Marco Vieira
João Carlos Cunha (Eds.)

LNCS 7869

Dependable Computing

14th European Workshop, EWDC 2013
Coimbra, Portugal, May 2013
Proceedings

Springer

Lecture Notes in Computer Science 7869

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Marco Vieira
João Carlos Cunha (Eds.)

Dependable Computing

14th European Workshop, EWDC 2013
Coimbra, Portugal, May 15-16, 2013
Proceedings

Springer

Volume Editors

Marco Vieira
University of Coimbra
Department of Informatics Engineering
Pólo II, Pinhal de Marrocos, 3030-290 Coimbra, Portugal
E-mail: mvieira@dei.uc.pt

João Carlos Cunha
Polytechnic Institute of Coimbra
Coimbra Institute of Engineering
Rua Pedro Nunes, 3030-199 Coimbra, Portugal
E-mail: jcunha@isec.pt

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-38788-3 e-ISBN 978-3-642-38789-0
DOI 10.1007/978-3-642-38789-0
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013939190

CR Subject Classification (1998): C.2, F.2, G.2

LNCS Sublibrary: SL 2 – Programming and Software Engineering

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Welcome to the proceedings of the 14th European Workshop on Dependable Computing, held in Coimbra, Portugal.

EWDC is a privileged forum for discussion of current and emerging trends on dependability area, fostering a close and fruitful contact between researchers from both academia and industry. After the first series, started in 1989 and held on a yearly basis until 2000, EWDC was revived in 2009, running since then every two years, alternatively with the European Dependable Computing Conference (EDCC). In 2009 EWDC was held in Toulouse, and in 2011 in Pisa.

The University of Coimbra had the pleasure of hosting the 2013 edition of the workshop. Established in 1289, this is one of the oldest universities in Europe. It is located in Coimbra, a small city built around the university, again along the European tradition of university towns.

EWDC 2013 focused on dependability and security of software and services. This is a challenging theme, especially when considering that information systems are more and more based on complex, heterogeneous, dynamic software and services, which are characterized by demanding quality attributes. Interoperability in the presence of dependability and security guarantees, as well as techniques and tools to assess the impact of accidental and malicious threats, are among the crucial aspects to be addressed.

The technical program included nine full papers and six short papers, accepted from 24 submissions, with some interesting contributions from joint work between academia and industry. All these papers were blind-reviewed by the Program Committee (PC) members and external reviewers. A total of 72 reviews were received, and each paper was reviewed by three referees.

For the first time in EWDC, some authors were invited to submit fast-abstracts, having the opportunity to present at the workshop work in progress or new ideas addressing very interesting issues in the dependability area. Six fast-abstracts were accepted.

We thank the PC members and external reviewers for their invaluable contribution in the rigorous review process. A special appreciation to the EWDC Steering Committee members for their precious guidance.

We hope EWDC 2013 yields a positive impact in the research projects and careers of all participants, and that these proceedings will be a valuable source of knowledge for future readers.

May 2013

Marco Vieira
João Carlos Cunha

Organization

Organization

University of Coimbra

Committees

Workshop Chair

Marco Vieira	University of Coimbra, Portugal

Publication Chair

João Carlos Cunha	Polytechnic Institute of Coimbra, Portugal

Local Arrangements Chair

Nuno Laranjeiro	University of Coimbra, Portugal

Publicity Chair

Bruno Cabral	University of Coimbra, Portugal

Steering Committee

Karama Kanoun	LAAS-CNRS, France
Algirdas Avizienis	Vytautas Magnus University, Lithuania
Jean-Charles Fabre	LAAS-CNRS, France
Felicita Di Giandomenico	ISTI, Italy
Ricardo Jimenez-Peris	Technical University of Madrid, Spain
Johan Karlsson	Chalmers University of Technology, Sweden
Henrique Madeira	University of Coimbra, Portugal
Miroslaw Malek	University of Lugano, Switzerland
Jaan Raik	Tallinn University of Technology, Estonia
Brian Randell	Newcastle University, UK
Luca Simoncini	University of Pisa, Italy
Janusz Sosnowski	Warsaw University of Technology, Poland

Program Committee

Alexander Romanovski	University of Newcastle, UK
Ana Rugina	ESA, Netherlands
António Casimiro	Faculdade de Ciências da Universidade de Lisboa, Portugal

Corrado Leita	Symantec, France
Domenico Cotroneo	University of Naples, Italy
Elena Troubitsyna	Åbo Akademi University, Finland
Erland Jonsson	Erland Jonsson, Sweden
Felicita Di Giandomenico	ISTI, Italy
Felix Salfner	Humboldt-Universität zu Berlin, Germany
Frank Ortmeier	University of Magdebourg, Germany
Hans-Peter Schwefel	FTW, Austria
Ilir Gashi	City University, UK
István Majzik	Budapest University of Technology and Economics, Hungary
Janusz Górski	Gdansk University of Technology, Poland
José Orlando Pereira	University of Minho, Portugal
Juan-Carlos Ruiz	Technical University of Valencia, Spain
Marco Serafini	Yahoo! Research, Spain
Marta Patiño-Martínez	University of Madrid, Spain
Michael Paulitch	EADS IW, Germany
Nicolas Rivière	LAAS-CNRS, France
Paolo Lollini	University of Florence, Italy
Simin Nadjm-Tehrani	Linköping University, Sweden
Tomas Bures	Charles University, Czech Republic
Vincent Nicomette	LAAS-CNRS, France

External Reviewers

Ivan Studnia	LAAS-CNRS, France
Jeremie Guiochet	LAAS-CNRS, France
José Luís Nunes	Polytechnic Institute of Coimbra, Portugal
Linas Labinis	Åbo Akademi University, Finland
Raul Barbosa	University of Coimbra, Portugal
Thibaut Probst	LAAS, France

Table of Contents

Dependable and Secure Computing

Fast Abstracts

Enhancing Intrusion Detection in Wireless Sensor Networks through Decision Trees

Alessia Garofalo, Cesario Di Sarno, and Valerio Formicola

Department of Technology, University of Naples Parthenope, Naples, Italy
{alessia.garofalo,cesario.disarno,
valerio.formicola}@uniparthenope.it
http://www.dit.uniparthenope.it/FITNESS/

Abstract. Wireless Sensor Networks (WSNs) are being increasingly adopted also in very sensitive applications where it is of paramount importance to ensure that the sensor network is protected from cyber-security threats. In this paper we present a new IDS architecture designed to ensure a trade-off between different requirements: high detection rate is obtained through decision tree classification; energy saving is obtained through light detection techniques on the motes. A dataset including sinkhole attack has been created and employed to evaluate the effectiveness of the proposed solution. Such a dataset has been made available, and will facilitate future comparisons of alternative solutions.

Keywords: Decision Tree, Anomaly Detection, Misuse Detection, Wireless Sensor Networks, Intrusion Detection Systems.

1 Rationale and Contribution

Wireless Sensor Networks (WSNs) are being increasingly adopted also in very sensitive applications such as forest fire detection [1], power transmission and distribution [2], localization [3], military applications [4], Critical Infrastructures (CIs) [5], underwater infrastructures monitoring (Underwater Wireless Sensor Networks) [6].

In such a context, it is of paramount importance to ensure that the sensor network is protected from cyber−security threats. Unfortunately achieving this objective is made particularly challenging by a number of characteristics of WSNs, the most relevant being: limited computational resources, preventing the implementation of strong cryptographic mechanisms; and their deployment in wild unattended environments, where it is easy for the attacker to physically access the devices (e.g. to read cryptographic keys directly from the memory).

Since it is commonly agreed that attacks cannot be always avoided or prevented, intrusion detection is thus needed as an additional line of defense. Detecting intrusions is the goal of Intrusion Detection Systems (IDSs) that already represent a key tool for ensuring cyber security in traditional computer based systems. Besides detection capabilities IDSs can also offer additional mechanisms, such as diagnosis [7] and prevention [8]. IDSs architectures for WSNs are

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 1–15, 2013.

currently being investigated and many solutions have been proposed in the literature. Unfortunately, no widely agreed solutions are currently available and this is due to many reasons, the most important ones being i) the high heterogeneity of technologies and protocols behind the term WSN; ii) the lack of means for actually comparing the effectiveness of different IDS solutions.

As for i) it is of great importance to develop solutions that can be applied to the standards (e.g. protocols, routing algorithms, operating systems) mostly adopted by industries, and to develop systems that can be used independently of the specific network topology. A first contribution of this paper is the implementation of a novel IDS solution for WSNs. The proposed solution is based on the architecture presented in [9] and exploits statistical models and cognition based detection techniques [10], in particular [11] Decision Trees, to achieve high detection rates and low power consumption. Our solution specifically addresses mesh networks. For such topology in WSNs, only few works are available. Existing solutions often suggest IDS architectures, and the related characteristics, exploiting, and thus taking advantage, of components offered by the specific network topology. If no network topology is mentioned, then a hierarchical detection mechanism is often obtained through election mechanisms [12]. On the contrary, our purpose is to investigate a specific network topology and at the same time to provide solutions that can be used in other topologies. So, the best choice to us was to consider all nodes in the network as peers, thus as part of a mesh network (more details are provided in following sections). The developed solution targets the Ad hoc On Demand Distance Vector (AODV) [13] routing protocol. This choice was made since AODV is both widely adopted and it is also the underlying routing protocol for the Zigbee standard specification.

With respect to ii), as an example there is no WSN dataset including security attack traces that can be used for comparing detection capabilities of different IDS architectures or implementations. Currently, publicly available WSN datasets typically contain sensor readings and/or functional parameters [14] [15] [16]. On the contrary, to the best of our knowledge, no dataset is currently available about WSN cyber security issues, i.e. about WSN routing attacks. A second contribution of this paper is therefore the production of the dataset, including sinkhole attack, that was employed for setting up the detection parameters of our IDS. The dataset is further described in Sect. 4 and is made available for further researches and for comparisons to our work [17].

The paper is organized as follows. Section 2 provides an overview of intrusion detection, reviews related work and compares available IDS solutions to our work. Section 3 describes the IDS architecture used in this work, its components and corresponding detection activities. Section 4 presents the experimental setup and Sect. 5 describes the cyber attack that was implemented. Section 6 presents the experimental results obtained, and Sect. 7 discusses the results achieved.

2 Background and Related Work

The National Institute of Standards and Technology (NIST) [18] identifies two main approaches to intrusion detection, *misuse detection* and *anomaly detection*.

In misuse detection, only known attacks can be detected: also, all known attack patterns need to be previously described e.g. through separate signatures. In anomaly detection, the system activity is monitored and an attack is detected whenever the current behaviour of the system is considered to be different than the one expected when no attack occurs.

Learning techniques for intrusion detection activities can be divided in *supervised*, *unsupervised* and *semi-supervised*. In supervised learning, a training set has to be provided where each information is typically composed by several attributes that describe the features of a target system. In the provided training set, a target attribute has to be given that represents the goal to be learned − e.g. if the goal is to detect cyber attacks, the target attribute may have the values "no attack", "under DoS attack", "under sinkhole attack". So, in supervised learning the IDS learns how to detect previously labeled attacks through available attributes. Instead, in unsupervised learning a normal behaviour is described with no a priori knowledge: when the behaviour of the system does not match the expected normal behaviour it is marked as abnormal − e.g. in the case of intrusion detection from cyber attacks, the system is detected as under attack, but nothing can be said about which specific attack is occurring. Semi−supervised learning is an hybrid approach between supervised and unsupervised learning [19] [20] [21].

Our work proposes a novel IDS for WSNs that aims to achieve high detection accuracy and light detection mechanisms on motes. This work improves aspects of the architecture proposed in [9], where an IDS is proposed and validated for WSNs within CIs. In [9], a hybrid architecture is proposed where intrusion detection activities are performed both by a Central Agent and a number of Local Agents. However, details are not provided about the implementation of the Central Agent: this aspect is instead investigated in our work.

In [22], the same architecture as [9] is used; additionally, an implementation of the Central Agent is proposed and it makes use of Hidden Markov Model (HMM). However, configuration of HMM is not simple and it is also based on a priori knowledge of several characteristics of the target system, but these information can be unavailable.

In our work, sinkhole [23] attack was considered on a WSN simulated through NS−3 [24]; the implementation proposed for Central Agent uses Decision Trees (DT): advantages and characteristics of this approach are detailed in following sections. Our experimental campaign shows that different DT techniques do not achieve comparable detection capabilities, so the choice of a specific learning method is an important task when the IDS is featured by DT learning.

Recently, several intrusion detection techniques were developed in order to find suitable solutions for security issues of WSNs. They are called *centralized* when a base station provides all intrusion detection capabilities, whereas *decentralized* solutions make use of the sensors' capabilities for performing intrusion detection activities.

Currently, IDSs rarely choose fully centralized approaches, since the IDS would represent a single point of failure, and a slow reaction time would also be experienced. Also, all motes should often send information to the central IDS, thus consuming additional resources. This issue could be mitigated by lowering the frequency of communications, but this would also imply in less effective detection mechanisms. Moreover, centralized solutions are vulnerable to routing level attacks. In [25], a centralized approach is chosen because of the computation and resource constraints that would be experienced by deploying an IDS on motes. While low resource consumption is claimed, it is not proved: as an example higher communication costs due to the IDS are not considered. From the point of view of intrusion detection accuracy, the IDS in [25] claims to obtain high detection rates but the work cannot be compared to ours since it does not provide data used for their experiments.

More research work can be found on decentralized solutions: i.e. intrusion detection mechanisms are used only on WSN motes. Typically those solutions imply either high energy consumption or low detection accuracy. Also, the proposed architectures should take into account the low availability of resources, otherwise for instance the intrusion detection activities could discharge batteries very quickly and the service provided by the WSN would soon become not available. In [26], a fully decentralized architecture for anomaly detection is proposed: high detection and low false positive rates are claimed to be achieved, but we are not able to compare our work to theirs, since test data are not made available for comparisons. In [26] no considerations or experimental tests were made about energy consumption: also, each node is assumed to have enough resources in order to perform the required computations. Authors are aware that the technique they propose cannot be applied to all sensor networks: we can say that the proposed detection mechanisms can hardly be used on existing WSNs. Our solution is instead energy-aware since detection activities were chosen according to the low availability of computational resources on motes.

As discussed in next section, DT learning has significant advantages and the most relevant issues can be mitigated. DTs have already been demonstrated to be effective, while compared to other detection techniques, for traditional IP-based networks. As an example, in [27] a DT algorithm is compared with a different supervised learning method: experimental results show that the detection rate of the decision tree is higher than the other one for almost all the cyber attacks the techniques were tested for. However, we are not aware of application of DT to WSNs.

As we already introduced, the main purpose of our solution is to enhance cyber security on WSNs and at the same time to take into account the specific characteristics of WSNs. So, our target is to reach acceptable solutions for a set of requirements, including low energy consumption for motes, high detection rate, low false positive rate.

3 Intrusion Detection Architecture for Wireless Sensor Networks

The IDS architecture proposed in [9] and considered for improvements in this work is shown in Fig. 1. The IDS is composed by a Central Agent (CA) and several Local Agents (LAs): each LA is deployed on a WSN node and the CA is deployed on a server that acts as a base station for the WSN. Aspects about fault tolerance are beyond the scope of the architecture.

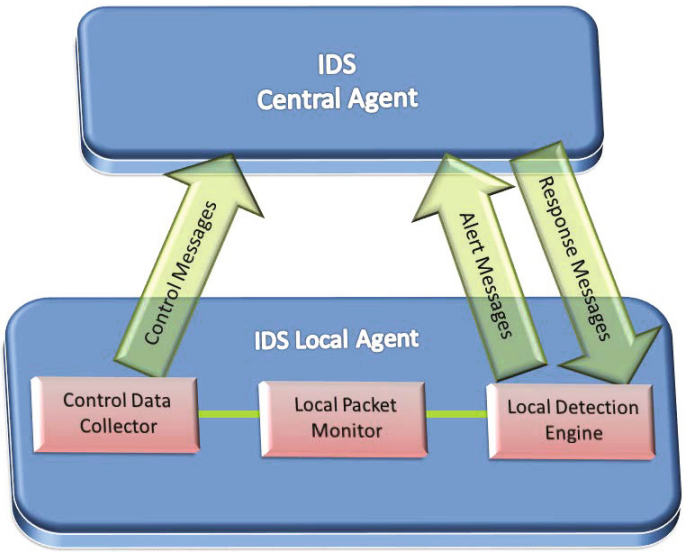


Fig. 1. IDS architecture

3.1 IDS Local Agent

The main components of the LAs are:

1. *Local Packet Monitor*: it monitors the traffic flowing through the node where the LA is deployed
2. *Control Data Collector*: it performs measures of parameters that have to be sent to the CA
3. *Local Detection Engine*: it performs local detection activities, raises security alerts, receives response messages from the CA, and is in charge of recovery activities when required

The Local Detection Engine performs detection activities by making use of data gathered by the Control Data Collector and related to monitored nodes - the choice of the nodes to be monitored by each LA is performed by the CA. When a potential attack is detected, a 'weak' detection event occurs: in that case, the

potentially compromised node is added to a blacklist and all routes containing that node are replaced so to forward packets through a different node. Also, the LA sends a security alert to the CA and waits for a response confirming or rejecting the alert. If the response is not received, the LA repeats the process and forwards data (including the alert) through different nodes until a response is received by the CA. When this happens, the blacklist is emptied or freezed according to the decision of the CA. Also, after the alert is confirmed the LA performs reactions chosen by the CA, otherwise the LA returns to regular mode. Further details can be found in [9]. We highlight that there is no interaction between LAs, since the final decision about a potential attack is performed through interactions with the CA (as shown in the following section). The computation of the final decision is only demanded to the CA, and this allows to reduce the computation on the mote and also LAs are fully independent from each other, so attacks to a specific LA do not have direct consequences on other LAs.

The chosen anomaly detection technique makes use of threshold metrics [10]: that is, specific events are counted over a given time window. If the number of events exceeds a given threshold, an alarm is raised. In this work, events related to cyber security of the system are chosen and counted. The events are counted through Exponential Moving Average (EMA): in its expression, a factor called *smoothing factor* is defined that gives higher weight to most recent data.

$$EMA_t = EMA_{t-1} + \alpha \left(M_t - EMA_{t-1} \right), \quad \alpha = 2/\left(N+1\right) \tag{1}$$

In (1), EMA_t is the current value for EMA, EMA_{t-1} is the last value obtained through EMA, α is the smoothing factor, N is the window size, M_t is the current value for the parameter being counted.

Equation (1) was chosen over Moving Average (MA) since it only requires the storage of last value obtained at run time, instead the expression of MA would require to store all data collected in the given window. The expression for MA is:

$$MA_t = \frac{1}{N} \sum_{i=0}^{N-1} M_{t-i} \tag{2}$$

being MA_t the current value of the MA, M_{t-i} the value for the parameter being counted at time $t-i$, N the size of the considered window. Equation (2) shows that the choice of MA would imply storage of all data in the moving window: specifically, space requirements grow linearly as the window size increases.

In (1) instead, the required storage size is not dependent on the size of the chosen window. So, even if a very large window size is chosen, space requirements of (1) are not so high to let this method be not usable on the mote. Also, the computations required by (1) are not highly resource-consuming: for each time slot, the only activity performed by a LA during intrusion detection activities is to update values in (1), compare the obtained result with the estimated threshold, and eventually send an alert to the CA. So, usage of EMA allows higher energy saving on the mote with reference to MA.

3.2 IDS Central Agent

The CA improves the detection capabilities of the proposed architecture through the validation of security alerts raised by LAs when an attack is detected. In this work we assume that the CA and LAs are free from faults or failures because we are focusing on intrusion detection capabilities of IDS.

The activities performed on the CA make use of a supervised technique and they are composed of two stages, called *profiling* and *misuse detection*. In the preliminary (*profiling*) stage, the CA collects data related to all nodes and gains knowledge of the specific characteristics of the network. This allows to define rules for detection of previously chosen cyber attacks. Also, in this stage the CA defines the nodes to be monitored by each LA (election mechanisms are beyond the scope of this work). In the *misuse detection* stage, the CA periodically receives information collected by each sensor for detection purposes: also, the CA validates or rejects incoming security alerts raised by LAs. In this stage, detection activities are performed at runtime by comparing current information to the definition of security attacks that was made in the previous stage.

Detection activities on the CA are based on *Decision Trees* and they are detailed in the following.

Decision Trees. As introduced in Sect. 2, Decision Tree (DT) learning is a promising approach for performing intrusion detection activities. DTs make use of supervised training: they need to be provided a training dataset and the target feature to be learned. The purpose of DTs is to split a given dataset into homogeneous subsets: specifically, the subsets to be created represent homogeneous data with reference to a specific parameter that belongs to the dataset. In the case of intrusion detection activities, the DT searches the available dataset for the features that better describe the conditions under given cyber attacks and under no attack.

In DTs, the attribution of a node to a certain class depends on the technique selected. Each technique is characterized by different parameters that aim to describe how homogeneous are data within a candidate subset (a branch or a leaf) of the DT. When the tree is built, the sequences of the conditions allowing to attribute data to certain classes can be used, e.g. in the form of if-else conditions, to detect if cyber attacks are occurring in the target system. It is thus clear that the DT is only able to forecast all cyber attacks that were provided during the training.

A disadvantage of DTs is *overfitting*. Overfitting occurs when an overcomplex model is built, thus the built model fits the dataset *too much* and incurs in loss of generality. This can be avoided through *pruning* techniques: when *postpruning* is used, nodes that cause overfitting in the tree are removed after the tree is built completely; instead, in *prepruning* the tree is built only partially according to some given criteria that mitigate overfitting. Further details can be found in [28].

Intrusion Detection through Decision Trees. In this work, detection activities on the CA are performed through DTs. The parameters chosen during our experimental campaign are shown in Sect. 6: all parameters are estimated through EMA as described in the previous section for LAs.

As already described, several techniques can be used to attribute nodes to specific classes: some of these techniques make use of similar parameters since they represent improvements of the same algorithm over time. In this work, techniques with different parameters were compared: they are *Classification And Regression Tree* (CART) [29], *CHi-squared Automatic Interaction Detection* (CHAID) [30], *C5.0* [31].

In the *profiling* stage the Decision Tree is built by using the attributes in the provided dataset that identify at best the target cyber attacks and the normal network state (that is, when no attack is occurring). As discussed, the tree built contains information to detect cyber attacks, so the CA is able to distinguish all the types of attack that were already provided in the reference dataset. After that, in the *misuse detection* stage the collected information is used to compare the behaviour of the system with the features of attacks previously learned and the normal behaviour.

4 IDS Experimental Setup

In this work, a WSN was simulated to perform preliminary tests of the proposed architecture. The simulation was made through ns-3 [32]: the simulations reproduce conditions when no attack is occurring and when a cyber attack is performed. Specifically, a network is simulated where monitoring activities are performed. Under these conditions, data are periodically collected and forwarded to a target destination, so route update requests are periodically performed when the collected data have to be sent. During a given attack window, the attacker sends two different malicious packets to try to perform the attack described in the following. In the dataset, 4 hours of simulated data are collected for 20 nodes: the attacker is supposed to have successfully compromised one of these nodes, so he/she tries to perform malicious activities within the given attack window. The attacked routing protocol is AODV. The dataset used in this work is available on our research group website [17]: Table 1 shows the setup parameters that were used to generate the dataset.

5 Attack Model and Implementation

The attack implemented in the provided dataset is a routing attack known as *sinkhole*. The purpose of sinkhole is to induce attacked nodes to send traffic through a compromised node: when the attack is successful, more severe attacks can be launched through the compromised node, e.g. collected data can be tampered, selectively dropped and so on.

So, when a sinkhole attack is performed, the attacker makes a route through the compromised node to a given destination look attractive to attacked nodes:

Table 1. Simulation parameters for the test dataset

Network Simulator	NS-3 [32]
Implemented Attacks	Sinkhole [23]
Routing Protocol	AODV [13]
Number of devices simulated	$N_d = 20$
Simulation time [hours]	4
Number of attackers	1
Sinkhole attack time window	$t \in [6400, 8000]s$

to do this, e.g. the attacker advertises a route to the WSN base station through a packet containing information about high quality routing metrics of the compromised node, such as low distance from the destination (the metrics are related to the specific routing protocol used in the network). When the attacked node receives the packet, it compares the information obtained to its route: if the new route is more attractive than his, it is replaced with the old one. In that case, the attack is successful: from this moment on, all the traffic directed from the attacked node to the WSN base station is sent through the compromised node.

In the simulated dataset, the routing protocol is Ad hoc On demand Distance Vector (AODV) [13]. AODV makes use of four message types: RREQ (Route REQuest), RREP (Route REPly), RERR (Route ERRor), RREP-ACK (Route REPly ACKnowledgement). These messages are used respectively to request a route to a given destination, to reply to a RREQ message by providing information about the requested route, to notify e.g. that a route has to be deleted, to return an acknowledgment when a RREP message is received.

Sinkhole attack to AODV routing protocol was launched as follows:

```
if (isAttackTimeWindow)
     then SendSinkholeRREQ(myIP) and SendSinkholeRREP(myIP)
```

In this work, the attacker is assumed to have already compromised a legitimate node in the WSN: periodically, when routing activities are performed by the compromised node, a check is made whether actual time belongs to the attack time window. When the condition is met, the attacker launches the sinkhole attack, so a legitimate routing request is performed for a path to the WSN base station through a RREQ message. When no attack is being performed, RREQ messages are sent only when knowledge of a specific route is necessary. So, the attacker does not need to alter the format of this message for its purposes, however the message is forced to be sent.

After the RREQ message is sent, the attacker itself sends a route response through a RREP message. The message is tampered with high quality route metrics in agreement with AODV protocol: according to RREP message format, the message has to contain a low distance from the route destination (expressed as low number of hops) and a recent timestamp to prove that the route is the most recent (through a high sequence number). When the tampered RREP is

sent, other nodes check the new route: if the new route is more attractive than the one they own, their old route is replaced with the new one and the attack is successful.

6 Experimental Campaign and Results Analysis

This section shows the preliminary results obtained through the dataset described in Sect. 4. In following tables, detection capabilities were measured through *False Positive Rate* (*FPR*), *False Negative Rate* (*FNR*), *True Negative Rate* (*TNR*), *True Positive Rate* (*TPR*), *Accuracy* (*ACC*). FPR represents the rate of cases of no attack identified incorrectly, FNR represents the rate of cases of attack identified incorrectly, TNR represents the rate of cases of no attack identified correctly, TPR represents the rate of cases of attack identified correctly, ACC is the total rate of correct detections.

$$FPR = \frac{FP}{FP + TN}, \quad FNR = \frac{FN}{FN + TP}, \quad TNR = \frac{TN}{TN + FP}, \tag{3}$$

$$TPR = \frac{TP}{TP + FN}, \quad ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

So, FPR and FNR represent rates of incorrect detections, whereas TPR and TNR represent rates of correct detections: finally, ACC represents the overall accuracy of the detection activity - in (3) and (4), TP = true positive, FN = false negative, FP = false positive, TN = true negative.

Table 2 shows the setting parameters for the simulation of LAs. Table 3 shows the parameters monitored for both Local Agents and Central Agent through EMA. A higher number of parameters was initially taken into account by evaluating the characteristics of AODV routing algorithm (that was detailed in previous section). After that, tests were performed on the target dataset and they revealed the presence of redundant information, so they were excluded.

Table 2. Setting parameters for each Local Agent

Number of monitored nodes per Local Agent	$N_n = 3$
Sliding window size N for EMA	3
EMA update delay [s]	1
Training Data	50% of dataset [17]
Validation Data	50% of dataset [17]

As for simulations of CA, in the experimental campaign a number of tests were made about detection through Decision Tree. As introduced in previous sections, Decision Trees can be built through different methods, and different results are obtained according to the specific method chosen. So, different Decision Trees techniques were tested, and after that they were compared in order to choose

the one with better results. Also, in Decision Trees different parameters describe each specific learning method: so, each method was tested with different values of its specific parameters, then the final values were chosen for each method with reference to the best results in terms of the metrics shown in (3) and (4).

The final values for each decision tree algorithm were obtained by performing several tests and choosing the settings which produce the best results in terms of the metrics defined in (3) and (4). The settings are shown in Table 4. The compared techniques were *Classification And Regression Tree* (CART) [29], *CHi-squared Automatic Interaction Detection* (CHAID) [30], *C5.0* [31]: as already introduced, the specific parameters shown in the table are related to the specific learning characteristics of each decision tree.

Table 3. Parameters monitored by Local Agents and Central Agent

EMA_{RREQ}	Mean number of exchanged RREQ messages per second
EMA_{RREP}	Mean number of exchanged RREP messages per second
EMA_{RERR}	Mean number of exchanged RERR messages per second
EMA_{DROP}	Mean number of dropped messages per second
EMA_{FWD}	Mean number of forwarded messages per second
EMA_{ROUTE}	Mean number of route update per second
EMA_{hf}	Mean number of hop count change frequency per route per second
EMA_{sf}	Mean number of sequence number update frequency per route per second
EMA_{hr}	Mean hop count update range per route per second
EMA_{sr}	Mean sequence number update range per route per second
MON	Monitored nodes

Table 4. Setting parameters for comparison of data mining techniques on Central Agent

CART [29]	Maximum tree depth	20
	Maximum surrogates	8
	Impurity measure	Gini index
	Minimum impurity change	10^{-4}
CHAID [30]	Maximum tree depth	20
	Alpha for splitting	0.05
	Alpha for merging	0.05
	Epsilon for convergence	10^{-3}
	Maximum iterations for convergence	100
C5.0 [31]	Pruning severity	100
	Number of folds for cross-validation	10
	Minimum records per child	2
All techniques	Training Data	50% of available dataset [17]
	Validation Data	50% of available dataset [17]
	Sliding window size N for EMA	3
	EMA update delay [s]	1

Table 5 shows the final results achieved in our experimental campaign through the different learning methods: the most effective technique was CART. As a matter of fact, it is the only technique that achieves both low FPR and low FNR. Also, CART has high TPR and high TNR: that means that this technique detects correctly attacks and no attack much better then CHAID and C5.0. These two methods have been estimated as more accurate as they have higher ACC, but they rarely detect the sinkhole attack (TPR is low and FNR is high with reference to CART). Through this experimental campaign, CHAID was not capable of detecting attacks: conditions under no attack are correctly detected (TNR = 1, FPR = 0); also, attacks are never detected (TPR = 0), instead they are always detected as no attack (FNR = 1).

In Table 5, experimental results presented in [33] are also shown. In this work, sinkhole attack was simulated on AODV routing protocol by extending an available simulation package, then performances of the proposed architecture are tested through these simulations. As already discussed, to the best of our knowledge neither the work [33] nor other works provide an attack dataset for comparisons. However, we take into consideration results in [33] as a preliminary mean for understanding the orders of magnitude of performances that are currently obtained through architectures similar to ours. In Table 5, we can see that our IDS that makes use of CART has not only comparable, but also higher performances than the solution proposed in [33].

Table 5. Performances of the proposed IDS through different decision tree techniques and comparisons to PCADID architecture proposed in [33]

Architecture Tested	Decision Tree	*FPR*	*FNR*	*ACC*	*TPR*	*TNR*
IDS proposed in Fig.1	CART	0.02	$7*10^{-4}$	0.978	0.999	0.978
IDS proposed in Fig.1	CHAID	0	1	0.994	0	1
IDS proposed in Fig.1	C5.0	$3*10^{-6}$	0.968	0.995	0.032	0.999
PCADID [33]	-	0.15	-	-	0.932	-

7 Summary and Conclusions

In this paper, an IDS for WSNs was presented that aims not to optimize one specific requirement, but to address several existing issues and to achieve an acceptable trade-off among them - our main targets are high intrusion detection rate, implementation of anomaly detection techniques on WSNs that allow low energy consumption, implementation of intrusion detection techniques on mesh networks. As a matter of fact, the proposed architecture can be especially addressed for applications whose main purpose is to monitor and keep under control environmental parameters, as in environmental monitoring. In such cases, ensuring at any time the minimal working conditions for WSNs is a key aspect in order to keep the monitoring activities always available.

Since datasets are not currently available for comparing existing works with our results, this work also provides a reference dataset [17] created through ns-3: the dataset contains normal and malicious network traffic and it was used to obtain the experimental results shown. This dataset is available for comparisons with future works. Even though comparisons cannot be made through a reference dataset, we made considerations about a work [33] which tests intrusion detection capabilities of a IDS against the same cyber attack and on the same routing protocol as the ones we considered: comparisons to our experimental results show that better performances are obtained in our work both in terms of correct detection of attacks and in terms of low false positives.

We plan to enhance intrusion detection activities by introducing additional approaches that are already proved to be successful in other different environments. For instance, the IDS could improve its detection capabilities by making use of additional specific information whether the specific WSN allows it; also, information for detecting attacks could be collected at different architectural levels [34] or through diverse sources [35]. Our work also aims at further improvements about the aspect of energy consumption: for instance, an estimation of the effective resources consumption on motes for the proposed techniques can be made. After that, threshold metrics on LAs could be investigated further in order to define at best the normal behaviour and the alert threshold i.e. if communications are only made when necessary, a higher amount of energy is saved.

Acknowledgments. This work has been partially supported by the TENACE PRIN Project (n. 20103P34XC) funded by the Italian Ministry of Education, University and Research.

The research leading to these results has received funding from the European Commission within the context of the Seventh Framework Programme (FP7/2007-2013) under Grant Agreement No. 257644 (MAnagement of Security information and events in Service Infrastructures, MASSIF Project).

References

1. Díaz-Ramírez, A., Tafoya, L.A., Atempa, J.A., Mejía-Alvarez, P.: Wireless sensor networks and fusion information methods for forest fire detection. Procedia Technology 3, 69–79 (2012)
2. Isaac, S., Hancke, G., Madhoo, H., Khatri, A.: A survey of wireless sensor network applications from a power utility's distribution perspective. In: AFRICON 2001, pp. 1–5 (2011)
3. Mao, G., Fidan, B., Anderson, B.D.: Wireless sensor network localization techniques. Computer Networks 51(10), 2529–2553 (2007)
4. Durisic, M., Tafa, Z., Dimic, G., Milutinovic, V.: A survey of military applications of wireless sensor networks. In: 2012 Mediterranean Conference on Embedded Computing, MECO, pp. 196–199 (2012)
5. Afzaal, M., Di Sarno, C., Coppolino, L., D'Antonio, S., Romano, L.: A resilient architecture for forensic storage of events in critical infrastructures. In: 2012 IEEE 14th International Symposium on High-Assurance Systems Engineering, HASE, pp. 48–55 (2012)

6. Wahid, A., Kim, D.: Connectivity-based routing protocol for underwater wireless sensor networks. In: 2012 International Conference on ICT Convergence, ICTC, pp. 589–590 (2012)
7. Coppolino, L., D'Antonio, S., Elia, I.A., Romano, L.: From intrusion detection to intrusion detection and diagnosis: An ontology-based approach. In: Lee, S., Narasimhan, P. (eds.) SEUS 2009. LNCS, vol. 5860, pp. 192–202. Springer, Heidelberg (2009)
8. Mudzingwa, D., Agrawal, R.: A study of methodologies used in intrusion detection and prevention systems (idps). In: 2012 Proceedings of IEEE Southeastcon, pp. 1–6 (2012)
9. Coppolino, L., D'Antonio, S., Romano, L., Spagnuolo, G.: An Intrusion Detection System for Critical Information Infrastructures using Wireless Sensor Network technologies. In: 2010 5th International Conference on Critical Infrastructure, CRIS, pp. 1–8. IEEE (2010)
10. Jyothsna, V., Prasad, V.V.R., Prasad, K.M.: Article: A review of anomaly based intrusion detection systems. International Journal of Computer Applications 28(7), 26–35 (2011)
11. Safavian, S., Landgrebe, D.: A survey of decision tree classifier methodology. IEEE Transactions on Systems, Man and Cybernetics 21(3), 660–674 (1991)
12. Bao, F., Chen, I.R., Chang, M., Cho, J.H.: Trust-based intrusion detection in wireless sensor networks. In: 2011 IEEE International Conference on Communications, ICC, pp. 1–6 (June 2011)
13. Perkins, C., Royer, E., Das, S.: RFC 3561 Ad hoc On-Demand Distance Vector (AODV) Routing. Technical report (2003)
14. Bodik, P., Hong, W., Guestrin, C., Madden, S., Paskin, M., Thibaux, R.: Intel lab data (2004), `http://db.csail.mit.edu/labdata/labdata.html` (last accessed November 27, 2012)
15. Suthaharan, S., Alzahrani, M., Rajasegarar, S., Leckie, C., Palaniswami, M.: Labelled data collection for anomaly detection in wireless sensor networks. In: 2010 Sixth International Conference on Intell. Sens., Sens. Netw. and Inf. Processing, ISSNIP, pp. 269–274 (December 2010), Dataset available online at `http://www.uncg.edu/cmp/downloads/` (last accessed November 27, 2012)
16. Hackmann, G., Chipara, O., Lu, C.: Art experimental data (2009), `http://wsn.cse.wustl.edu/index.php/ART_Experimental_Data` (last accessed November 28, 2012)
17. FITNESS Research Group: Cyber security datasets for wireless sensor networks (2012), `http://fitnesslab.altervista.org/index.php/it/?option=com_content&view=article&id=71` (last accessed November 27, 2012)
18. Bace, R., Mell, P.: NIST special publication on intrusion detection systems (2001)
19. Chapelle, O., Schölkopf, B., Zien, A. (eds.): Semi-Supervised Learning. MIT Press, Cambridge (2006)
20. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM Comput. Surv. 41(3), 15:1–15:58 (2009)
21. Cherkassky, V.S., Mulier, F.: Learning from Data: Concepts, Theory, and Methods, 1st edn. John Wiley & Sons, Inc., New York (1998)
22. Coppolino, L., Romano, L., Bondavalli, A., Daidone, A.: A hidden markov model based intrusion detection system for wireless sensor networks. Int. J. Crit. Comput.-Based Syst. 3(3), 210–228 (2012)
23. Karlof, C., Wagner, D.: Secure routing in wireless sensor networks: attacks and countermeasures. Ad Hoc Netw. 1, 293–315 (2003)

24. Henderson, T.R., Lacage, M., Riley, G.F.: Network Simulations with the ns-3 Simulator. In: ACM SIGCOMM 2008, p. 527. ACM (August 2008)
25. Kaplantzis, S., Shilton, A., Mani, N., Sekercioglu, Y.: Detecting Selective Forwarding Attacks in Wireless Sensor Networks using Support Vector Machines. In: 3rd International Conference on Intelligent Sensors, Sensor Networks and Information, ISSNIP 2007, pp. 335–340 (December 2007)
26. Loo, C.E., Ng, M.Y., Leckie, C., Palaniswami, M.: Intrusion detection for routing attacks in sensor networks. International J. of Distributed Sens. Netw. 2(4), 313–332 (2006)
27. Ektefa, M., Memar, S., Sidi, F., Affendey, L.: Intrusion detection using data mining techniques. In: 2010 International Conference on Information Retrieval Knowledge Management, CAMP, pp. 200–203 (March 2010)
28. Alpaydin, E.: Introduction to Machine Learning. 2nd edn. The MIT Press (2010)
29. Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and Regression Trees. Wadsworth and Brooks, Monterey (1984)
30. Kass, G.V.: An exploratory technique for investigating large quantities of categorical data. J. of the R. Statistical Society. Ser. C (Appl. Statistics) 29(2), 119–127 (1980)
31. Quinlan, R.: c5.0 (2007), `http://rulequest.com/` (last accessed November 28, 2012)
32. National Science Foundation, Planète group: ns-3 (2012), `http://www.nsnam.org/` (last accessed November 28, 2012)
33. Livani, M., Abadi, M.: A PCA-based distributed approach for intrusion detection in wireless sensor networks. In: 2011 International Symposium on Computer Networks and Distributed Systems, CNDS, pp. 55–60 (February 2011)
34. Ficco, M., Romano, L.: A generic intrusion detection and diagnoser system based on complex event processing, pp. 275–284 (2011)
35. Coppolino, L., D'Antonio, S., Esposito, M., Romano, L.: Exploiting diversity and correlation to improve the performance of intrusion detection systems. In: International Conference on Network and Service Security, N2S 2009, pp. 1–5 (2009)

Middleware Support for Adaptive Real-Time Applications in Wireless Sensor Networks*

João Alves, António Casimiro, and Luís Marques

Faculdade de Ciências da Universidade de Lisboa, Portugal
{jalves,lmarques}@lasige.di.fc.ul.pt, casim@di.fc.ul.pt

Abstract. This position paper describes initial efforts and ideas for the development of a middleware framework to support the operation of adaptive Wireless Sensor Networks applications with real-time and dependability requirements. We identify a set of underlying services that need to be implemented as part of this framework, explaining why they are needed and what they provide. In order to illustrate how this middleware can be used and its potential benefits, we consider the well-known LQER routing protocol to show how it must be changed to incorporate probabilistic real-time requirements and meet them in a dependable way.

Keywords: Wireless sensor networks, dependability, timeliness, middleware.

1 Introduction

Wireless sensor networks (WSNs) have a number of unique capabilities which make them suitable for many different classes of applications. This includes weather monitoring, industrial monitoring and quality control, object tracking and medical monitoring. But the nice features of WSN, like small size, reduced price and ease of use and deployment in the field, also pose a set of challenges to the application/system developer. WSNs have limited power supply, limited memory and processing power, and may need to operate under harsh conditions on unpredictable environments, making them more susceptible to external faults. The primary focus of the research community has been on addressing these challenges, designing protocols and techniques which try to be as energy efficient as possible, and also robust against the potential faults.

A problem that has been far less considered is that of achieving dependable solutions. Dependability can be defined as "the ability to deliver a service that can justifiably be trusted" [2], and can be broken down into six attributes: availability, reliability, safety, confidentiality, integrity and maintainability. These attributes, and dependability in general, are particularly important in WSN applications involving the control of critical infrastructures, real-time coordination of robots or other vehicles, or involving life-care services.

* This work was partially supported by the EU through the KARYON project (FP7-288195) and the FCT through the Multiannual Funding Program.

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 16–23, 2013.

Since applications using sensor networks rely on the availability and timeliness of the collected sensor data, one way of looking at dependability in this context is to reason in terms of these attributes. In fact, timeliness is not by itself an attribute of dependability, but instead the probability that timeliness requirements are met (i.e., that the system is reliable with respect to timing faults). In [5] the authors explore the concept of perception quality, and observe that as time passes, the quality of our perception of a sensed environment variable diminishes. In other words, the similarity between the real observed event and its systems internal representation tends to grow weaker the longer the sensing took place. As such, a dependable network is understood as one that is able to deliver the sensed data within a temporal bound, and do that with a desired probability.

In this paper we provide initial ideas for a middleware aimed at supporting dependable applications and services with timeliness requirements, running over wireless sensor networks. Achieving hard-real time properties in WSNs is an unrealistic objective [10]. Therefore, our work is intended to support adaptive services and applications, which are able to adjust their behavior at run-time to meet available resources and environment conditions. Dependability objectives are achieved through the adaptation process rather than by securing some fixed temporal bound. In practice, assumed bounds will be secured with a given probability, and the middleware will bring awareness about the relation between bounds and the probability of securing them at any given moment. The resulting programming model will be adequate to develop these applications in environments that exhibit uncertain temporal bounds with some degree of stability.

The paper is organized as follows. In Section 2 we go over related work on the topics of dependability and timeliness in WSNs. Then, Section 3 addresses the middleware design, underlying assumptions, goals and architecture. In Section 4 we provide a brief overview of how the middleware can be used to address timeliness requirements, considering a routing protocol as a toy example. Section 5 concludes the paper and refers to future work.

2 Related Work

2.1 Dependability in WSNs

A lot of research has been done on solving particular issues inherent to wireless sensor networks, such as energy conservation [1] or traffic reduction [8], a topic that has not received as much attention has been that of dependability. The dependability of a system reflects the trust a user has in that system, i.e., that the system will operate as expected [2]. Although dependability has several attributes, availability and reliability are the more relevant ones to our focus on timeliness. For a system to be highly available it means that its downtime is very small (which is usually expressed as a percentage of uptime in a given time frame). Reliability stands for the probability that in a time interval [0,t] the system will operate properly and continuously. In the context of wireless sensor networks, a dependable, reliable network, is a network where for a long sensing

period the data is sensed, sent and delivered to the destination with a very small (or no) downtime and with a very small loss of data [12]. When considering also timeliness requirements, as we do, timing faults must be avoided, and this need must be reflected in the notion of dependability.

In [12], the authors propose an event-based middleware service to improve the dependability of the network. This service uses the publish-subscribe scheme, and divides the network into clusters, where the cluster-heads serve as event brokers to the remaining nodes in a cluster (the publishers, or event sources) and to the base-station, which acts as subscriber. When a cluster-head fails, a network reconfiguration phase is triggered and the other cluster-heads have the responsibility of "adopting" the nodes of the failed cluster.

In contrast with the previous example, we want to provide support for awareness of the timeliness of the network operation, as a means to allow applications to adapt when some nodes fail or stop responding and when the communication latency varies over time (e.g., due to interferences, mobility, or other factors). We thus provide enriched dependability guarantees.

2.2 Timeliness in WSN

Timeliness has been often overlooked in detriment of energy consumption. But as WSNs delve evermore into the realm of time-sensitive applications and scenarios, it becomes increasingly important to address timeliness requirements in the context of these networks. One issue that must be addressed is the very definition of timeliness. In classical real-time computing, timeliness usually stands for the execution of actions in a bounded and fixed, well-known, amount of time. However, strictly satisfying deadlines requires a set of assumptions and system models that do not hold for WSNs [9]. In WSNs, by their very nature, guarantees about communication latency cannot be given in a strict sense. The network does not exhibit deterministic behavior due to the open environment and resources sharing. Communication is subject to varying and uncertain message delays and loss, which makes worst case analysis very difficult or even impossible [9]. Thus, the singular constraints of WSNs require a different characterization for the notion of timeliness.

In [9], a generalized notion of timeliness is introduced, which takes into account the particular nature of WSNs. This notion is built on top of the idea that applications should not request infeasible degrees of performance from the network (a request for a strict deadline on an individual message, is an example of such a request, mainly because of the uncertainty associated with end to end latency in WSNs). The authors define formally, that the generalized notion of timeliness is composed by a time interval, which delimits the target end-to-end transmission interval for a sequence of messages, and by a level of confidence, the probability of successful end-to-end transmissions within the time interval.

We have also previously exploited the idea of providing probabilistic guarantees in alternative to strict hard real-time guarantees [6]. We argue that no matter the approach used to make WSNs more predictable, perturbations are still likely to occur, and as such, instead of specifying fixed upper bounds on

system variables (latency, maximum number of omissions, etc.) monitoring and adaptation techniques should be used to characterize and deal with the uncertainty. Therefore, we proposed a technique based on non-parametric statistics, a lightweight method of statistical inference, which can be used to characterize environment conditions or, in other words, the state of the network. This information can then be used for adaptation purposes. Our current work exploits the availability of such monitoring service, including it in a middleware layer designed to support the development of adaptive real-time applications on WSNs.

3 Middleware Description

The middleware is designed with the following two principles in mind:

Principle 1: Realistic assumptions about WSNs technology and deployment.
Principle 2: Probabilistic timeliness as opposed to hard real-time guarantees.

One of the problems with previous work done on the topic of timeliness and real-time in WSNs has been that it is based on naive assumptions that severely restrict their applicability to real-world systems [10]. Following Principle 1 we will stay clear of such assumptions, whether they are about network and environment conditions, the goals of the architecture, or the applications using it.

Another reason to stay clear of restrictive assumptions is to allow the middleware to be applicable to a broader range of environments. The middleware design should, as much as possible, be independent of the underlying network topology and dynamics and of the expected applications and services running on top of it. Lets consider, for example, routing protocols as potential services running on top of the middleware. Both a flat, multi-route protocol, and an hierarchical cluster-based protocol should be able to take advantage of the middleware although it was not designed specifically for either of them.

One assumption about the environment that is necessary, is that its behaviour, while uncertain, has limited dynamics, i.e. the environment does not change to rapidly in relation to the perception capabilities of the system. The results in [7] validate this assumption, showing that adaptation can be done which is very close to the theoretical perfect adaptation. This implies that the environment dynamics are limited, otherwise this match would not occur, since the bound values would be very different by the time the adaptation was complete.

As stated earlier, achieving hard real-time guarantees is an unrealistic goal for wireless sensor networks. Therefore, in accordance with Principle 2, the design takes from the work in [6] and [9] and exploits the notion of probabilistic timeliness guarantees.

With these principles in mind, we propose a middleware design that can be described as a set of components or services that should run at each node in the network and that provide other applications and services with useful information about the runtime state of the network to support adaptation

The main component is the monitoring service. This service is responsible for monitoring system metrics, such as communication latency or node connectivity, node energy or packet loss rate. The monitoring service itself can be designed as a set of components, one for each monitored metric, and a communication interface. The other components are in essence auxiliary to the monitoring component, providing services it needs to accomplish its goal. These components are the clock service and the non-parametric statistics service.

The clock service is explicitly included as an independent service, as it constitutes a fundamental abstraction for distributed monitoring of time intervals or, in this case, of communication latencies. Without a notion of global time it would be impossible for the monitoring system to perform measurements of end-to-end latencies, as required to provided the intended service. We are aware that techniques based on round-trip delay measurements can be used to measure communication latencies, which could lead to the idea that no global notion of time is needed. However, these techniques requiring message exchanges are just implicit forms of clock synchronization, even if no clocks are explicitly synchronized. On the other hand, a clock service providing global time can be implemented without the need to exchange messages, therefore standing as a building block on its own. For instance, it might be possible that nodes have GPS-synchronized clocks, or that they use the power lines for clock synchronization. If these methods are not available then a clock synchronization algorithm, such as the one in [11] could be used.

The non-parametric statistics service [6] is the component responsible for realizing the statistical operations over sample data provided at its input interface. The monitoring service collects the required sample data, that is, measurements of message delays for messages exchanged with neighbor nodes, and feeds the statistics service to obtain the probabilistic distribution for these message delays. Based on that, it will be possible to answer questions like "what is the probability that a message will be delivered within X time units?". Of course, it may be possible to disseminate latency measurements to other nodes and hence feed the statistics service with the necessary data to raise awareness about the (probabilistic) latency to any other node in the network. In many cases, namely if there is a sink node in the network, the relevant latency measurements will concern the communication between nodes and the sink.

The monitoring component we have just described will be accessed by other services or applications through an interface. As an example, consider the interface for end-to-end latency estimates:

```
(latency l) <- getLatency (node n, probability p)

(probability p) <- getProbability (node n, latency l)
```

These two simple functions provide the basic service of the monitoring component. In the first function a node can inquire the monitoring service about the latency between itself and a node n that will hold with a given probability p. For example, a sensor node that is sending temperature measurements to a sink

node can become aware of the the latency bound that will hold with a certain probability, say, 0.98. If this latency is too high (which can imply that the sink may sometimes have a temporally inconsistent view of the temperature), then maybe the node will decide to increase the frequency of updates in order avoid such temporal inconsistencies with the indicated probability (0.98).

The second function is the complement of the first. In this case the application will inquire the middleware about the probability that a given latency l to a node n will hold. For example, the same temperature sensor node may want to know how probable it is that the latency to the sink will be, at most, 200ms. If the probability happens to be very low, then this may trigger some reconfiguration of adaptation of the node behavior in order to achieve a more predictable behavior.

A similar style of interface can be designed for other monitored metrics, and more powerful constructs can be built, in addition to these operations and maybe using them. The tradeoff between providing a richer middleware is the overhead in terms of needed resources, which are scarce in WSNs. This is why a non-parametric approach, which is light-weight, is considered for the statistics service.

4 Sample Application

In the following paragraphs we provide an example of how the support middleware could be used to enhance an existing service, by allowing this service to perform in a probabilistic real-time way, instead of being simply best-effort.

We consider a routing service, and focus on the modification of the LQER (Link Quality Estimation based Routing for Wireless Sensor Networks) protocol [3]. LQER is inspired by MCR (Minimum Cost Routing) [13] and MHFR (Minimum Hop Field based Routing), and utilizes the concept of a dynamic sliding window of length k for storing historical data of link quality, i.e., the success or failure of the last k transmissions on a given link.

The LQER protocol starts with the minimum hop field establishment, which has the goal of setting up the optimal path to send data to the sink, for each node. In this stage the sink broadcasts an ADV (advertisement) message which contains the hop count to the sink (0 at the sink). This message will be propagated through the network using the flooding algorithm and when a node n receives an ADV message from a node m, it will compare its hop count (h_n) to the advertised hop count of node m (h_m). If $h_m + 1$ is smaller than h_n then h_n is set to $h_m + 1$ and n broadcasts the ADV message with hop count equal to h_n. If $h_m + 1$ is equal to h_n then n adds m to its forwarding table but does not broadcast the ADV message. If $h_m + 1$ is bigger than h_n, then n simply ignores the message. At the end of this stage each node should be able to calculate the minimum hop count to the sink and have a forwarding node set.

Once the minimum hop field for each node has been established, nodes can start routing messages to the sink. When a node needs to send a message to the sink it will choose the node from its forwarding table with the best link quality, that is the node which has the largest value of $\frac{m}{k}$ where m is the number of

successful transmissions among the last k transmissions in the sliding window for that link. After the message is forwarded, the sliding window is updated to account for the success or failure of that transmission.

LQER uses loss as a link quality metric, but the use of the support middleware allows us to consider other time related metrics, such as the probability of success to meet a certain deadline (expressed as the end-to-end latency to the sink) or the estimated end-to-end latency for a desired probability of success. The necessary adaptations for the protocol to operate with these metrics, take place in the Link Quality Table Maintenance Algorithm and the Link Quality Estimation Routing Algorithm. Bellow we provide an overview of the baseline algorithms and of the necessary adaptations for each of the two discussed metrics to be used.

The majority of the changes happens in the Link Quality Table Maintenance Algorithm. Instead of storing the success or failure of a given transmission on some link, the algorithm first queries the middleware (calls the function getProbability) for the probability of sending data through that link within some deadline (which may be a configuration parameter of the routing protocol). Then, it stores the returned value, by first deleting the oldest value on the table and inserting the new one. When choosing a node from the forwarding table, the algorithm will choose the node with the highest average of the probability values stored in the Link Quality Table.

The changes necessary to use the second metric are very similar to the above, but instead of querying the middleware for the probability of meeting a deadline, the algorithm provides a probability and receives from the middleware an estimate of the best possible end-to-end latency for that probability (calls getLatency). When choosing a node from the forwarding table, the algorithm will choose the node with the smallest average of the end-to-end latency values in the Link Quality Table.

5 Conclusions

In this paper we proposed a support middleware for applications with dependability and timeliness requirements. We defined the middleware as a set of components, a main monitoring service and a suite of auxiliary components, which provide, through a programming interface, useful functions to applications running on WSNs. We showed what those functions might look like and provided an example to illustrate how they may be used to in a protocol that takes into account probabilistic timeliness requirements instead of simply exhibiting best-effort behavior.

Our future work will focus on implementing the proposed middleware and an example application, so as to study and evaluate the achievable benefits in terms of the capability to effectively support adaptive real-time applications. We intend to measure the overhead introduced by the middleware, in comparison to a baseline implementation providing best-effort service.

References

1. Anastasi, G., Conti, M., Di Francesco, M., Passarella, A.: Energy conservation in wireless sensor networks: A survey. Ad Hoc Netw. 7(3), 537–568 (2009)
2. Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 1(1), 11–33 (2004)
3. Chen, J., Lin, R., Li, Y., Sun, Y.: Lqer: A link quality estimation based routing for wireless sensor networks. Sensors 8(2), 1025–1038 (2008)
4. Garces-Erice, L., Rooney, S.: Dependable actuation in wireless sensor networks. In: 2007 IEEE Globecom Workshops, pp. 1–5 (November 2007)
5. Marques, L., Casimiro, A.: Data validity and dependable perception in networked sensor-based systems. In: 2010 29th IEEE Symposium on Reliable Distributed Systems, October 31-November 3, pp. 358–362 (2010)
6. Marques, L., Casimiro, A.: Lightweight dependable adaptation for wireless sensor networks. In: 4th International Workshop on Dependable Network Computing and Mobile Systems (DNCMS 2011), Proceedings of the 30th IEEE International Symposium on Reliable Distributed Systems Workshops, Madrid, Spain, pp. 26–35 (October 2011)
7. Marques, L., Casimiro, A.: Evaluating lightweight dependable adaptation in 802.15.4 wireless sensor networks. Technical Report 2012;04, Faculdade de Ciências da Universidade de Lisboa (2012)
8. Ngai, E.C.-H., Gelenbe, E., Humber, G.: Information-aware traffic reduction for wireless sensor networks. In: IEEE 34th Conference on Local Computer Networks, LCN 2009, pp. 451–458 (October 2009)
9. Oliver, R.S.: An Approach to Timeliness in Wireless Sensor Network Communications. PhD thesis, Technische Universität Kaiserslautern (September 2010)
10. Oliver, R.S., Fohler, G.: Timeliness in wireless sensor networks: Common misconceptions. In: Proceedings of the 9th International Workshop on Real-Time Networks, RTN 2010, Brussels, Belgium (July 2010)
11. Sichitiu, M.L., Veerarittiphan, C.: Simple, accurate time synchronization for wireless sensor networks. In: 2003 IEEE Wireless Communications and Networking, WCNC 2003, vol. 2, pp. 1266–1273 (2003)
12. Taherkordi, A., Taleghan, M., Sharifi, M.: Dependability considerations in wireless sensor networks applications. Journal of Networks 1(6) (2006)
13. Ye, F., Chen, A., Lu, S., Zhang, L.: A scalable solution to minimum cost forwarding in large sensor networks. In: Proceedings of the Tenth International Conference on Computer Communications and Networks, pp. 304–309 (2001)

Formal Analysis of Dynamic Domain Establishment Protocol in Cloud Logging Service

Wei Hu and Dongyao Ji

State Key Laboratory of Information Security
Graduate University of the Chinese Academy of Sciences
Institute of Information Engineering, Chinese Academy of Sciences
No.89 Minzhuang Road, Haidian District, Beijing, 100195, P.R. China
whu@is.ac.cn

Abstract. We present a formal analysis of the dynamic domain establishment protocol in the Cloud logging service. The protocol is used to establish a trust channel between the log as a service client agent (LCA) and the log as a service server agent (LSA). Formal specification and verification have been carried out using the specification language HLPSL and AVISPA, a state-of-the-art verification tool for security protocols. AVISPA has revealed two main security flaws, one of which (previously unheard of, up to our knowledge) allows an intruder to impersonate the LCA to join the dynamic domain, and may launch a denial-of-service attack. To address this problem, we propose to use explicit identity information in one's signature. The other one is the information leakage problem, to solve this problem we propose a modification of the protocol by adding a key update protocol. After these modifications, this protocol has been verified with AVISPA to be safe from these two attacks.

Keywords: Cloud computing, dynamic domain establishment, security protocol, formal analysis.

1 Introduction

In Cloud computing, there are enormous processes running at distributed and heterogeneous resources[1]. They produce a huge number of log records (in a variety of different forms), stored in many different places. So it is very difficult to handle these log records, let alone to manage them safely and automatically. But logging services are very important in Cloud computing. For example, in order to establish trust in Clouds, we can link together log records that are produced by multiple devices so that we can reconstruct the complete history of an event or result. Besides, logging services facilitate the communication and recording of diagnostic audit trails, as well as provide means to help achieve a number of security-related objectives[2]:

1. reconstruction of events—audit trails are used to reconstruct events when a problem occurs, while damages are assessed by analysing audit trails of system activity to identify how, when and why operations have stopped;

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 24–38, 2013.

2. intrusion detection—again, to identify malicious attempts to penetrate a system and gain unauthorised access to resources;
3. problem analysis—status of processes running in critical applications or services can be monitored with real-time auditing.

However, there are few people that put emphasis on the security of logging. So in the area where logging services are needed, a lot of problems appear. For example, Cloud provenance is based on logging services and it is very important for establishing trust in Clouds. In Cloud provenance, there are some limitations associated with it. For instance, current provenance mechanisms are object specific, that is, they do not automate the process of managing different log records and linking dependent log records together. Log and audit records are not reasonably protected, which in turn affects the credibility of provenance in the Cloud. Moreover, Cloud provenance mechanisms are deployed and fully controlled by Cloud providers, that is, Cloud users do not have control over such mechanisms, and neither can they access log and audit records. In order to overcome these limitations, Imad M. Abbadi has proposed a framework for establishing trust in Cloud provenance [3,4]. In the Abbadi's framework, he addressed the above problems by 1. Move log records from their originating distributed processes to a centralized repository; 2. Make log records be easily queried using standard mechanisms; 3. Associate individual log records with metadata.

Most importantly, Imad M. Abbadi has proposed a dynamic domain[5] concept to securely and easily manage log records. The most common threat on a logging service arises when an attacker compromises it, gains unauthorised access to all log data being forwarded from log generators (e.g. systems, software, middleware services), and modifies it to record fabricated data. To mitigate this type of attack, the logging service needs to be deployed independently from any parent application, on a strongly isolated compartment that provides robust memory protection. It needs to be a small and simple software designed to resist such attacks, and this should make attestation between the log generators and the logging service more feasible. The logging service needs to be able to verify the software configurations of all log generators. Such a verification mechanism is required to protect the logging service from an attacker submitting arbitrary logs, performing denial of service attacks, and also to filter out untrustworthy logging requests from a compromised log generator. So it is very useful to apply a dynamic domain to supply logging services.

A dynamic domain represents a group of devices that need to securely share a pool of content[6]. Each dynamic domain has a unique identifier i_D, a shared unique symmetric key k_D and a specific PKL_d composed of all devices in the dynamic domain. k_D is shared by all authorized devices in a dynamic domain and is used to protect the dynamic domain contents whilst in transit. This key is only available to devices that are members of the domain. Thus, only such devices can access the pool of content bound to the domain. Each device is required to securely generate for each dynamic domain a symmetric key k_C, which is used to protect the dynamic domain contents when stored in the device. Imad

M. Abbadi has defined a dynamic domain called LaaSD (Log as a Service Domain). In an LaaSD, k_{laas} is used to protect log records when transferred within the LaaSD and $k_{laas-cca}$ is used to protect log records when transferred from Cloud entities to the LaaSD. LaaSD consists of a platform that hosts Cloud LaaS applications. The architecture of an LaaSD is described in Fig. 1 as follows.

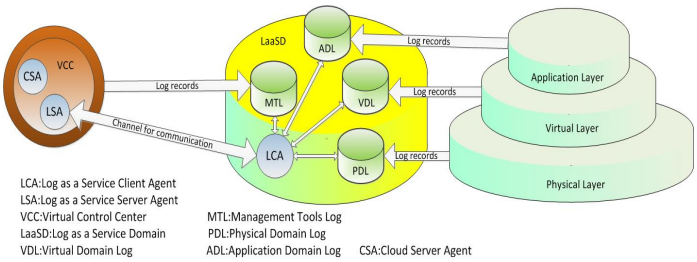


Fig. 1. The structure of an LaaSD

The LaaS application is used to move disperse log records to a centralized provenance system. Before an LaaSD can be taken into use, there are some operations needed to be performed. For example, the LaaS server agent initialization, the LaaS client agent initialization, the LaaSD establishment, LaaS client agents and an LaaS server agent mutual authentication, LaaS client agents joining the domain. Imad M. Abbadi has provided some protocols to accomplish these operations in his paper. We've found some faults in his protocols and these faults will result in the failure of the Abbadi's goals. These attacks we discover are new. We believe that the attacks we discover demonstrate that our approach is a successful way to automatically detecting errors in security protocols.

1.1 Structure of This Paper

The rest of this paper is organised as follows. In the next section we introduce the dynamic domain protocol. In Section 3 we provide the protocol formalization. In Section 4 we present the authentication flaw and the information leakage vulnerability we've found in the protocols. In Section 5 we present our solutions to these flaws and in Section 6 we draw some final remarks.

2 The Dynamic Domain Establishment Protocol

In this section we describe the dynamic domain establishment protocol[5]. Actually, there is no official description of this protocol; hence, in order to understand it, we have analyzed the whole protocol. The whole protocol includes seven algorithms. From Algorithm 1 to Algorithm 5, there are mainly some initialization steps in them, and these steps don't include the interactions between the LSA

and the LCA. Algorithm 6 and Algorithm 7 are initiated to add the LCA devices into the LaaSD managed by the LSA. The LSA sends an attestation request to the LCA to prove its trustworthiness, the LCA then sends the attestation outcome to the LSA. These steps are achieved using Algorithm 6. Adding a device into the dynamic domain uses Algorithm 7, which starts upon successful completion of Algorithm 6. The objective of Algorithm 7 is to securely transfer the key k_{laas} and $k_{laas-cca}$ to the LCA. Both keys are sealed on the device hosting the LCA, so that they are only released to the LCA when its execution environment is as expected. If the execution status of the device running the LCA is trusted, the LSA checks if the device's public key is included in the public key list of the domain. If so, it securely releases the domain-specific key k_{laas} and the LCA-CCA-specific key to the LCA using Algorithm 7. The keys are sealed on the LCA's device, so that they are only released to the LCA when its execution environment is as expected. Upon the successful completion of the above algorithms, the LaaS client and server agents establish a trust secure communication channel that is used to transfer the LaaSD key and policy to the LCA. These two algorithms include the following two protocols which are the core protocols in the Abbadi's framework. The TPM (Trusted Platform Module) is very important in these two protocols. It is a hardware module that provides an interface to help users to produce and store keys. It can help users to utilize the keys stored in it to encrypt or decrypt messages. It also has protected storage and protected capabilities. The protocols are described as follows.

Protocol 1:

1. $LSA \rightarrow TPM_{LSA} : TPM_{GetRandom}.$
2. $TPM_{LSA} \rightarrow LSA : N_1$
3. $LSA \rightarrow TPM_{LSA} : TPM_{LoadKey2}(Pr_{LSA-AIK})$
4. $LSA \rightarrow TPM_{LSA} : TPM_{Sign}(N_1)$
5. $TPM_{LSA} \rightarrow LSA \rightarrow LCA : N_1||Cert_{LSA}||Sign_{LSA}(N_1)$
6. $LCA \rightarrow TPM_{LCA} : TPM_{GetRandom}.$
7. $TPM_{LCA} \rightarrow LCA : N_2$
8. $LCA \rightarrow TPM_{LCA} : TPM_{LoadKey2}(Pr_{LCA-AIK})$
9. $LCA \rightarrow TPM_{LCA} : TPM_{CertifyKey}(SHA1(N_2||N_1||A_{LSA}||i_{laas}), Pu_{LCA})$
10. $TPM_{LCA} \rightarrow LCA : N_2||N_1||A_{LSA}||Pu_{LCA}||S_{LCA}||i_{laas}||Sign_{LCA}(N_2||N_1 ||A_{LSA}||i_{laas}||Pu_{LCA}||S_{LCA})$
11. $LCA \rightarrow LSA : N_2||N_1||A_{LSA}||Pu_{LCA}||S_{LCA}||i_{laas}||Cert_{LCA}||Sign_{LCA} (N_2||N_1||A_{LSA}||i_{laas}||Pu_{LCA}||S_{LCA})$

In Protocol 1, the LSA first requests its TPM to produce a random number, so the LSA sends to its TPM a command: $TPM_{GetRandom}$ (message 1). The LSA's TPM produces a random number N_1 and sends it to the LSA (message 2). In order to provide a signature, the LSA asks its TPM to load its private AIK (Attestation Identity Key) through the command $TPM_{LoadKey2}(Pr_{LSA-AIK})$ (message 3). After verifying that the current PCR (Platform Configuration Register, a register in the TPM) value matches the one associated with $Pr_{LSA-AIK}$ (the private key of AIK), the LSA's TPM loads that key. After its TPM has loaded an AIK, the LSA sends a command $TPM_{Sign}(N_1)$ to its TPM, to get

its signature on N_1 (message 4). The LSA's TPM sends its signature and a certificate to the LSA and then the LSA forwards it to the LCA (message 5). The LCA requests its TPM to produce a random number, so the LSA's TPM produces a random number N_2 and sends it to the LSA (message 6, message 7). The LCA asks its TPM to load its private AIK (message 8). The LCA requires its TPM to produce $SHA1(N_2||N_1||A_{LSA}||i_{laas})$ and the signature on this data, so the LCA's TPM produces that data according to the command and sends it to the LCA (message 9, message 10). Finally, the LCA sends the data produced by its TPM to the LSA (message 11). In this protocol, TPM_{LCA} is the TPM of the device running the LCA; TPM_{LSA} is the TPM of the device running the LSA; A_{LSA} is an identifier for the CSA (Cloud Server Agent) device included in $Cert_{LSA}$; i_{laas} is an laaSD-specific identifier; $Cert_{LSA}$ is the LSA device certificate and $Cert_{LCA}$ is the joining LCA device certificate; S_{LCA} is the platform state at release as stored in the PCR inside the TPM_{LCA}; Pu_{LCA} is the public key of the LCA and $Pr_{LCA-AIK}$ is the LCA'private AIK.

Protocol 2:

$1.LSA \rightarrow TPM_{LSA} : TPM_{LoadKey2}(Pr_{LSA})$

$2.LSA \rightarrow TPM_{LSA} : TPM_{Unseal}(k_{laas}||k_{laas-cca}||i_{laas}||PKL)$

$3.LSA \rightarrow TPM_{LSA} : TPM_{CertifyKey}(SHA1(N_2||A_{LCA}||e_{Pu_{LCA}}(k_{laas}||k_{laas-cca})))$

$4.LSA \rightarrow LCA : N_2||A_{LCA}||Pu_{LSA}||S_{LSA}||e_{Pu_{LCA}}(k_{laas}||k_{laas-cca})||Sign_{LSA}(N_2||A_{LCA}||e_{Pu_{LCA}}(k_{laas}||Pu_{LSA}||S_{LSA}))$

In Protocol 2, the LSA first requests its TPM to load its private key Pr_{LSA} (message 1). Then the LSA requests its TPM to unseal $k_{laas}||k_{laas-cca}||i_{laas}||PKL$ (message 2). After that, the LSA requests its TPM to sign $SHA1(N_2||A_{LCA}||e_{Pu_{LCA}}(k_{laas}||k_{laas-cca}))$ (message 3). At last, the LSA sends $N_2||A_{LCA}||Pu_{LSA}||S_{LSA}||e_{Pu_{LCA}}(k_{laas}||k_{laas-cca})||Sign_{LSA}(N_2||A_{LCA}||e_{Pu_{LCA}}(k_{laas}||Pu_{LSA}||S_{LSA}))$ to the LCA, mainly to allocate two keys to the LCA (message 4). In this protocol, A_{LCA} is an identifier for the LaaS client device included in $Cert_{LCA}$; k_{laas} is the LaaSD-specific content protecting key and $k_{laas-cca}$ is the LCA-CCA-specific key for protecting content transferred between CCA and LaaS and to establish trust between both entities; PKL is the public key list used in the domain; S_{LSA} is the platform state at release as stored in the PCR inside the TPM_{LSA}; Pr_{LSA} is the non-migratable private key of the LSA and it is bound to TPM_{LSA} and to the platform state S_{LSA}; Pu_{LSA} is the non-migratable public key of the LSA; $e_{Pu_{LCA}}(Y)$ denotes the asymmetric encryption of data Y using key Pu_{LCA}.

3 Protocol Formalization

In this section, we give a formal specification of the dynamic domain establishment protocol first in Alice-Bob notation and then in the form of HLPSL (High Level Protocol Specification Language). We notice that there are some internal operations in Abbadi's protocols. These operations are included in the steps where users' TPMs take part in. We remove these steps in the protocols because

they have no influence on the interactions between the LCA and the LSA. This simplification is based on the assumption that the channels between users and their TPMs are secure. By this means, we derive a precise model that ignores some unnecessary states and still captures the original protocols. Finally, we formally specify the security properties to be verified later in Section 3.2.

3.1 Alice-Bob Formalization

For the protocols described in Section 2, we can define a protocol formalization in Alice-Bob notation by removing the steps that are irrelevant for our analysis. Our formal model is built upon the protocols between the LCA and the LSA. We notice that in these protocols there are several redundant steps, which can be actually omitted without altering the security analysis. To make the protocols more clear, we capture the most important steps of the protocols. These simplifications allow for a shorter and simpler formalization, which is presented in Protocol 3 as follows.

Protocol 3:

$1. LSA \rightarrow LCA : N_1||Cert_{LSA}||Sign_{LSA}(N_1)$

$2. LCA \rightarrow LSA : N_2||N_1||A_{LSA}||Pu_{LCA}||S_{LCA}||i_{laas}||Cert_{LCA}||Sign_{LCA}(N_2||N_1||A_{LSA}||i_{laas}||Pu_{LCA}||S_{LCA})$

$3. LSA \rightarrow LCA : N_2||A_{LCA}||Pu_{LSA}||S_{LSA}||e_{Pu_{LCA}}(k_{laas}||k_{laas-cca})||Sign_{LSA}(N_2||A_{LCA}||e_{Pu_{LCA}}(k_{laas}||Pu_{LSA}||S_{LSA}))$

The steps that either the LSA or the LCA interacts with the TPM don't affect the mutual authentication between the LSA and the LCA, so we remove them. It is important to notice that this formalization still captures the original protocols; in fact, we've formalized and analyzed also the longer version of the protocols obtaining the same result. By analyzing the longer protocols, we've found another kind of attack and we will talk about it later.

3.2 HLPSL Formalization

For a protocol to be verified with AVISPA's[7] back-end model checking engines, it must first be encoded in HLPSL—an expressive, modular, role-based formal language that allows for the detailed specification of the protocol in question. An HLPSL[8] model typically includes the roles played in the security protocol, as well as the environment role and the security goals that have to be satisfied. The conceptual model of our HLPSL formalization is not very complex, we just use four roles to model the protocol. Translating the protocol into HLPSL is not very difficult, since it is written in Alice-Bob notation.

A role in HLPSL uses channels defined by the environment role for sending and receiving messages. The message sequences between each role have a one-to-one mapping to the Alice-Bob notation defined in the previous section. AVISPA analyzes protocols under the assumptions of a perfect cryptography and that the protocol messages are exchanged over a network controlled by a Dolev-Yao intruder. That is, the intruder can intercept, modify, and generate messages under any party name, but he cannot break cryptography without the decryption key.

We first specify the sequence of actions of each kind of protocol participant in a module, which is called a basic role. Each basic role contains a set of state transition definitions and local variables. In addition, each basic role contains a set of shared constants defined by the environment role to model the shared knowledge between different roles. In our protocol, there are two basic roles, which are called *ddp_Init* and *ddp_Resp*. These roles describe what information the participants can use initially(parameters), their initial states, and ways in which the states can change (transitions).

The basic roles are composed together in a composed role called *session*. *Session* role has no transition section, but rather it instantiates the two basic roles, glues them together so they can execute together. *Session* role describes sessions of the protocol. Each transition represents the receipt of a message and the sending of a reply message, and the local variables are set during a state transition. The transition section of our protocol model contains a set of transitions. A transition consists of a trigger, or precondition, and an action to be performed when triggering event occurs.

The last role to be defined in the protocols is the environment role, which is a top-level role that contains global constants and a composition of the sessions, where the intruder may play some roles as a legitimate user. There is also a statement which describes the initial knowledge of the intruder.

An HLPSL model is a state machine, and an AVISPA model checking engine tries to reach all possible states of the protocol to find an insecure state that violates at least one of the protocol's safety properties—referred as "security goals" in AVISPA. There are two types of security goal supported by HLPSL—secrecy and authentication(includes weak authentication and strong authentication)[9]. Each security goal, declared with a unique constant identifier, is an invariant that must hold true for all reachable states. Three special statements in HLPSL are used to specify the condition of a desired security goal. For secrecy goals, the secret statement specifies which value should be kept secret among whom; and if the intruder learns the secret value, then he has successfully attacked the protocol. For authentication goal, a pair of statements (witness and request) are used to check that a principal is right in believing that his intended peer is presented in the current session, and agrees on a certain value.

Our HLPSL model specifies five security goals based on the Alice-Bob formalization in Section 3.1. The security goals of our HLPSL model are specified as follows: *authentication_on n1, authentication_on n2, weak_authentication_on n1, weak_authentication_on n2, secrecy_of sec_k1, sec_k2*. For the secrecy goal, we use *secrecy_of sec_k1, sec_k2* which specifies that if the intruder learns a secret value that is not explicitly a secret between him and someone else, then the intruder has successfully attacked the protocol. For the authentication goal, we mean to check that a principal is right in believing that his intended peer is present in the current session, has reached a certain state, and agrees on a certain value, which is typically fresh. Weak authentication is violated whenever there is a request (b,a,id,m) but no matching witness event witness (a,b,id,m). It means that a party b believes a message m to come from a, but a has never sent m,

at least not for this purpose. Strong authentication is violated whenever weak authentication is, or whenever a request event occurs more frequently than the corresponding witness event.

In general, there is no bound on the number of parties and sessions of the protocol that can be executed in parallel. While one can bound the number of parties, by the argumentations or by the symbolic sessions technique of OFMC, the problem of an unbounded number of sessions cannot be solved in general since it gives rise to undecidability. Moreover, there are two similar problems of unboundedness in the protocol: there is no bound on the number of payload messages to be exchanged or on the number of new message sequences that can be started, i.e., the protocol contains unbounded loops. All these problems give rise to an unbounded number of steps of honest parties, while OFMC currently requires analysis settings with bounded numbers of steps of honest parties. There is also no bound on the complexity of messages that the adversary can generate. However, OFMC implements the lazy intruder technique, which uses a symbolic representation to avoid explicitly enumerating the possible messages that the Dolev-Yao adversary can generate, and which allows for an analysis without restricting this parameter of the problem.

We have therefore analyzed the protocol with OFMC under the following execution and analysis settings: there are at most two parallel protocol sessions, the client can start at most two message-sending sequences per protocol session. OFMC hasn't reported any attacks on the protocol for these analysis settings. Our protocol model specifies five security goals given above based on the Alice-Bob formalization. We've first checked the goal of mutual authentication, then we check the goal of secrecy. We've found that these goals are not achieved and the results are showed as follows. The source code is listed in Appendix A.

4 Attacks on the Dynamic Domain Establishment Protocol

In this section we present and discuss the results obtained by analyzing the HLPSL formalization presented in Section 3.2 using AVISPA (and in particular the OFMC model checker). This analysis has shown that the dynamic domain establishment protocol is subject to a masquerade attack and there exists an information leakage problem and the Cuckoo attack.

Masquerade Attack. The goal of this protocol is to establish a mutual authentication between the LSA and the LCA. After that, the LSA allocates the domain keys k_{laas} and $k_{laas-cca}$ to the LCA. We use SPAN and AVISPA to verify these goals. The attack is found when we require the *authentication_on n2* goal to be satisfied, in a scenario represented by an environment role with two parallel sessions between the two agents. The result is shown as the first alternative in the message sequence chart in Fig.2 and then in the attack trace as shown in Fig.3. Indeed, in the sequence of events found by OFMC (shown in Fig.2), the intruder reuses the message produced by the lca to fool the lsa to believe its forged identity of the lca, gaining the access to the dynamic domain.

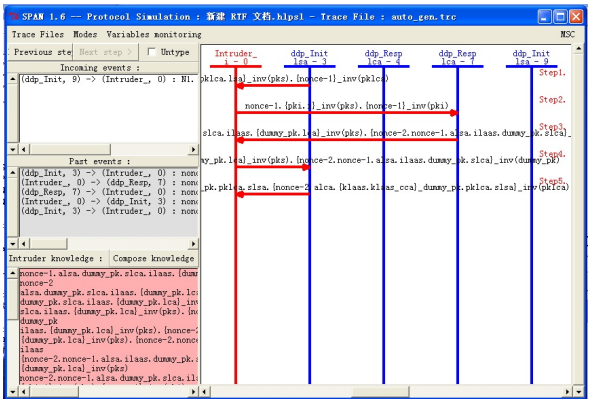


Fig. 2. The result produced by SPAN

From the results produced by the tools, we can get the attack steps as follows. (We use I (X) to stand for I impersonates the identity of X)

$1.I(LCA) \rightarrow LSA : start$
$2.LSA \rightarrow I(LCA) : N_1||Cert_{LSA}||Sign_{LSA}(N_1)$
$1'.I(LSA) \rightarrow LCA : N_1||Cert_{LSA}||Sign_{LSA}(N_1)$
$2'.LCA \rightarrow I(LSA) : N_2||N_1||A_{LSA}||Pu_{LCA}||S_{LCA}||i_{laas}||Cert_{LCA}||Sign_{LCA}(N_2||N_1||A_{LSA}||i_{laas}||Pu_{LCA}||S_{LCA})$
$3.I(LCA) \rightarrow LSA : N_2||N_1||A_{LSA}||Pu_{LCA}||S_{LCA}||i_{laas}||Cert_{LCA}||Sign_{LCA}(N_2||N_1||A_{LSA}||i_{laas}||Pu_{LCA}||S_{LCA})$
$4.LSA \rightarrow I(LCA) : N_2||A_{LCA}||Pu_{LSA}||S_{LSA}||e_{Pu_{LCA}}(k_{laas}||k_{laas-cca})||Sign_{LSA}(N_2||A_{LCA}||e_{Pu_{LCA}}(k_{laas}||Pu_{LSA}||S_{LSA}))$

We've found that, an attacker can successfully break the authentication between the LSA and the LCA. After these steps are executed, the LSA thinks it is the LCA who starts this operation, and accepts its identity. But in fact, the LCA doesn't start this operation and it is still waiting for the LSA (which is I in fact) to finish this operation. This attack can make the logging services crash down. The probable utilize of this vulnerability is that, an intruder can pretend to be a legal user to join the LaaSD. After he joins in the domain, he can do some malicious operations within the domain. For example, he can launch a denial-of-service attack after joining in the dynamic domain.

Information Leakage Problem. By analyzing the whole protocols, we've found that the key distributing protocol is not safe. With the help of AVISPA, we've found that an information leakage problem appears, when the LSA allocates the same keys to a newly joined LCA device. The newly joined LCA device can utilize these keys to decrypt some messages shared within the domain long time ago, then it can learn some information that shouldn't be known by it. Besides, when a device departs from the domain, information leakage problem

may appear again. After an LCA device departs from the domain, it shouldn't own the domain's secret keys, if no measures are taken, it will be very dangerous for the domain's members to communicate using those keys.

Cuckoo Attack. The Abbadi's framework is based on trusted computing, and the TPM plays an important role in his protocol. At present, there is a famous attack called the Cuckoo Attack[10] that can attack a computer with a TPM in it.

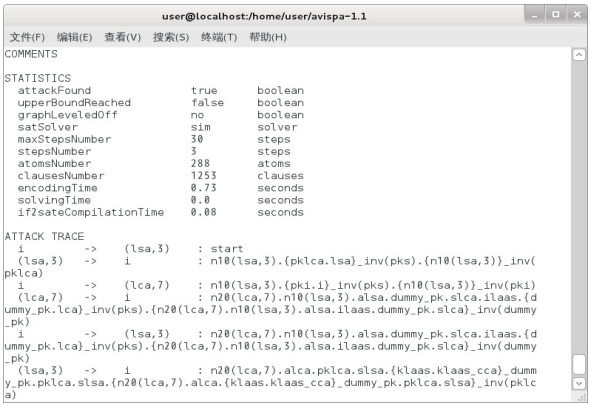


Fig. 3. The result produced by AVISPA

In one implementation of the cuckoo attack, malware on the user's local machine sends messages intended for the local TPM (TPM_L) to a remote attacker who feeds the messages to a TPM (TPM_M) inside a machine the attacker physically controls. Given physical control of TPM_M, the attacker can violate its security guarantees via hardware attacks. Thus, at a logical level, the attacker controls all communication between the verifier and the local TPM, while having access to an oracle that provides all of the answers a normal TPM would, without providing the security properties expected of a TPM. Then the attacker can pretend to be the LSA by forging its certificate. Certificate is very important in Abbadi's protocol, so this attack is really a big disaster to the dynamic domain.

5 Fixing the Protocol

In this section, we discuss how to repair the security flaws described above.

The way to deal with the masquerade attack is to change the identity label from A_{LSA} and A_{LCA} to the LSA and the LCA. Because identity is very important in authentication protocol, so it is necessary to add this information in one's signature. After this fixing, we've checked the protocol again, and this time there is no authentication error in the protocol.

The way to deal with the information leakage problem is to add a key update protocol in the dynamic domain. In Abbadi's dynamic domain establishment

protocol, every device in the domain will be allocated with two symmetrical keys k_{laas} and $k_{laas-cca}$ when it joins the domain. These keys are used within the domain. If every time a device joins the dynamic domain, these keys are different, then the information leakage problem can be settled. Besides, after a device leaves, there needs a way to update these keys too, so that no one can make use of these risky keys. Following Abbadi's method, we put forward a key update protocol as follows.

Protocol 4.

$1.LSA \rightarrow TPM_{LSA} : TPM_{GetRandom}.$
$2.TPM_{LSA} \rightarrow LSA : k'_{laas}.$
$3.LSA \rightarrow TPM_{LSA} : TPM_{GetRandom}.$
$4.TPM_{LSA} \rightarrow LSA : k'_{laas-cca}.$
$5.LSA \rightarrow TPM_{LSA} : TPM_{LoadKey2}(Pr).$
$6.LSA \rightarrow TPM_{LSA} : TPM_{Seal}(k'_{laas}||k'_{laas-cca}||i_{laas}||PKL_{laas}).$
$7.LSA \rightarrow TPM_{LSA} : TPM_{CertifyKey}(SHA1(LCA_1||e_{Pu_{LCA_1}}(k'_{laas}||k'_{laas-cca})), Pu_{LSA})$
$8.LSA \rightarrow LCA_1 : LCA_1||Pu_{LSA}||S_{LSA}||e_{Pu_{LCA_1}}(k'_{laas}||k'_{laas-cca})||Sign_{LSA}(LCA_1||e_{Pu_{LCA_1}}(k'_{laas}||k'_{laas-cca})||Pu_{LSA}||S_{LSA})$
$9.LSA \rightarrow LCA_2 : LCA_2||Pu_{LSA}||S_{LSA}||e_{Pu_{LCA_2}}(k'_{laas}||k'_{laas-cca})||Sign_{LSA}(LCA_2||e_{Pu_{LCA_2}}(k'_{laas}||k'_{laas-cca})||Pu_{LSA}||S_{LSA})$
...
$n.LSA \rightarrow LCA_n : LCA_n||Pu_{LSA}||S_{LSA}||e_{Pu_{LCA_n}}(k'_{laas}||k'_{laas-cca})||Sign_{LSA}(LCA_n||e_{Pu_{LCA_n}}(k'_{laas}||k'_{laas-cca})||Pu_{LSA}||S_{LSA})$

Through this protocol, the LSA can assign two new keys k'_{laas} and $k'_{laas-cca}$ to every LCA that belongs to the dynamic domain. When a new LCA device requires to join the domain and after the successful completion of Protocol 3, Protocol 4 should be executed to update the domain symmetric keys. When an LCA device, for example LCA_i, requires to apart from this domain, Protocol 4 should be executed except one step that assigns the domain symmetric keys to LCA_i, that is: i.$LSA \rightarrow LCA_i : LCA_i||Pu_{LSA}||S_{LSA}||e_{Pu_{LCA_i}}(k'_{laas}||k'_{laas-cca})||Sign_{LSA}(LCA_i||e_{Pu_{LCA_i}}(k'_{laas}||k'_{laas-cca})||Pu_{LSA}||S_{LSA})$. We've checked this protocol with AVISPA, finding it is safe in secrecy. It is necessary for the domain to adopt this protocol to assure it works well.

The solutions to the Cuckoo attack are provided as follows. (1) Removing network access. It seems that the Cuckoo attack can be prevented by severing the connection between the local malware and the adversary's remote PC. The assumption is that without a remote TPM to provide the correct responses, the infected machine must either refuse to respond or allow the true TPM to communicate with the user's device. (2) Eliminating malware. Another approach is to try to remove the malware on domain user's local computer. Unfortunately, this approach is both circular and hard to achieve. (3) Establishin a secure channel. Such a secure channel may be established using hardware or cryptographic techniques. This solution removes every opportunity for user errors, does not require

the preservation of secrets, and does not require software updates. Unfortunately, the cost and industry collaboration required to introduce a new interface make it unlikely to be deployed in the near future.

6 Conclusions

In this paper, we have presented a formal analysis of the dynamic domain establishment protocol. We have given a protocol formalization first in an Alice-Bob notion and then in HLPSL. By using AVISPA, we've found two main security flaws, including a masquerade attack: an intruder can intercept user session information, and reuse it to illegitimately join the domain. In order to fix this problem, we have corrected the protocol by adding an identity in one's signature. In order to fix the other problem, we have put forward a key update protocol. In our opinion, the key update protocol can be used in many occasions as well as the dynamic domain. As we know, key management is very important in information security area, but many people ignore the importance of key updating. These solutions have been formally verified using AVISPA.

Our approach can be applied in the area where security protocols are used. We believe that this is a practical and useful way of analyzing security protocols, allowing for the fast exploration of the state space, and capable of discovering new attacks. The limitations of our approach is that it can only verify authentication and secrecy goals, and the concurrency numbers of protocol sessions and protocol parties are limited. Our approach doesn't support channels beside Dolev-Yao ones and we assume the channels between users and their TPMs are secure. Besides, we haven't considered attacks on the encryption methods used, only on the protocol itself. All of these limitations in our approach can be studied in the future.

References

1. Armbrust, M., Fox, A., Joseph, A.D., Katz, R., et al.: A view of Cloud computing. Communications of the ACM 53(4), 50–58 (2010)
2. Huh, J.H., Martin, A.: Trusted logging for grid computing. In: Third Asia-Pacific Trusted Infrastructure Technologies Conference, pp. 30–42. IEEE Computer Society (2008)
3. Abbadi, I.M., Alawneh, M.: A framework for establishing trust in the Cloud. Computers and Electrical Engineering 38, 1073–1087 (2012)
4. Abbadi, I.M., Martin, A.: Trust in Cloud. Information Security Technical Report 16, 108–114 (2011)
5. Abbadi, I.M.: A framework for establishing trust in Cloud provenance. International Journal of Information Security 12(2), 111–128 (2013)
6. Abbadi, I.M.: Clouds' infrastructure taxonomy, properties, and management services. In: Abraham, A., Mauri, J.L., Buford, J.F., Suzuki, J., Thampi, S.M. (eds.) ACC 2011, Part IV. CCIS, vol. 193, pp. 406–420. Springer, Heidelberg (2011)
7. Hussein, M., Seret, D.: A comparative Study of Security Protocols Validation Tools: HERMES vs AVISPA. In: Proceedings of IEEE International Conference on Advanced Communication Technology, ICACT 2006, pp. 497–502. IEEE Computer Society (2006)

8. Chevalier, Y., Compagna, L., Cuellar, J., Drielsma, P.H., et al.: A High Level Protocol Specification Language for Industrial Security-Sensitive Protocols. In: Proc. SAPS 2004, pp. 281–285. Austrian Computer Society (2004)
9. Sun, S.T., Hawkey, K., Beznosov, K.: Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures. Computers & Security 31(4), 465–483 (2012)
10. Parno, B.J.: Trust extension as a mechanism for secure code execution on commodity computers. Thesis for the Ph. D. Degree, School of Electrical and Computer Engineering Carnegie Mellon University, pp. 62–70 (2010)

Appendix A. DDP Source Code

```
role ddp_Init (   LSA,LCA:agent,
                  Pklsa,Pks:public_key,
                  Klaas_cca,Klaas:symmetric_key,
                  Snd,Rec:channel(dy))
played_by LSA
def=

   local State            :nat,
         Pklca            :public_key,
         N1,N2,Ilaas      :text,
         SLCA,ALSA        :text

   const alca,slsa       :text,
         sec_k1,sec_k2   :protocol_id

   init State:=0

   transition

1.State=0
/\Rec(start)
=|>
State':=1
/\N1':=new()
/\Snd(N1'.{Pklsa.LSA}_inv(Pks).{N1'}_inv(Pklsa))
/\witness(LSA,LCA,n1,N1')
2.State=1
/\Rec(N2'.N1.ALSA'.Pklca.SLCA'.Ilaas'.{Pklca.LCA}_inv(Pks)
                  .{N2'.N1.ALSA'.Ilaas'.Pklca.SLCA'}_inv(Pklca))
=|>
State' :=2
/\Snd(N2'.alca.Pklsa.slsa.{Klaas.Klaas_cca}_Pklca.Pklsa.slsa.
        {N2'.alca.{Klaas.Klaas_cca}_Pklca.Pklsa.slsa}_inv(Pklsa))
/\request(LSA,LCA,n2,N2')
```

```
/\secret(klaas,sec_k1,{LCA,LSA})
/\secret(klaas_cca,sec_k2,{LCA,LSA})
end role

role ddp_Resp ( LCA,LSA:agent,
                Pklsa,Pks:public_key,
                Klaas,Klaas_cca:symmetric_key,
                Snd,Rec:channel(dy))
played_by LCA
def=

   local  State                 :nat,
          Pklca                 :public_key,
          N1,N2                 :text,
          SLSA,ALCA             :text

   const alsa,slca,ilaas        :text,
         sec_k1,sec_k2          :protocol_id

init State:=0



transition

1.State = 0
  /\Rec(N1'.{Pklsa.LSA}_inv(Pks).{N1'}_inv(Pklsa))
=|>
State':=1
/\N2':=new()
/\Snd(N2'.N1'.alsa.Pklca.slca.ilaas.{Pklca.LCA}_inv(Pks).{N2'.N1'
                                .alsa.ilaas.Pklca.slca}_inv(Pklca))
/\witness(LCA,LSA,n2,N2')


2.State=1
/\Rec(N2.ALCA'.Pklsa.SLSA'.{Klaas.Klaas_cca}_Pklca.Pklsa.SLSA'
     .{N2.ALCA'.{Klaas.Klaas_cca}_Pklca.Pklsa.SLSA'}_inv(Pklsa))
=|>
State':=2
/\request(LCA,LSA,n1,N1)
/\secret(Klaas,sec_k1,{LCA,LSA})
/\secret(Klaas_cca,sec_k2,{LCA,LSA})

end role
```

```
role session ( LSA,LCA:agent,
               Klaas,Klaas_cca: symmetric_key,
               Pklsa,Pklca,Pks:public_key) def=

local SC,RC,SS,RS: channel (dy)

composition

   ddp_Init(LCA,LSA,Pklsa,Pks,Klaas,Klaas_cca,SC,RC)
/\ ddp_Resp(LSA,LCA,Pklca,Pks,Klaas,Klaas_cca,SS,RS)

end role

role environment() def=

const n1,n2                :protocol_id,
      lsa,lca,i            :agent,
      klaas,klaas_cca      :symmetric_key,
      pklsa,pklca,pks,pki  :public_key

intruder_knowledge={lsa,lca,pki,inv(pki),pks,alsa,alca,slsa,slca,
             {pki.i}_inv(pks)}

composition

session(lca,lsa,klaas,klaas_cca,pklca,pklsa,pks)
/\session(lca,i,klaas,klaas_cca,pklca,pki,pks)
/\session(i,lsa,klaas,klaas_cca,pki,pklsa,pks)

end role

goal

authentication_on n1
authentication_on n2
weak_authentication_on n1
weak_authentication_on n2
secrecy_of sec_k1, sec_k2

end goal

environment()
```

Model-Driven Evaluation of User-Perceived Service Availability

Andreas Dittrich and Rafael Rezende

ALaRI Advanced Learning and Research Institute
Università della Svizzera italiana (USI)
Via G. Buffi 13, CH-6904 Lugano, Switzerland
{andreas.dittrich,rafael.ribeiro.rezende}@usi.ch
http://www.usi.ch/

Abstract. Service-oriented architecture (SOA) has emerged as an approach to master growing system complexity by proposing services as basic building elements of system design. However, it remains difficult to evaluate dependability of such distributed and heterogeneous functionality as it depends highly on the properties of the enabling information and communications technology (ICT) infrastructure. Moreover, every specific pair service client and provider can utilize different ICT components, constituting for the *user-perceived* view of a service.

We provide a model-driven methodology to automatically create reliability block diagrams of such views. Given a service description, a network topology model and a pair service client and provider, it identifies relevant ICT components and generates a user-perceived service availability model (UPSAM). We then use this UPSAM to calculate the steady-state availability of different views on an exemplary mail service deployed in the network infrastructure of University of Lugano, Switzerland.

Keywords: Service networks, Service dependability, Availability, Quality of service, Service network management, Modeling, Object oriented modeling, Design engineering.

1 Introduction

Growing functional and non-functional requirements have increased IT system complexity significantly during the last decade. At the same time, modern business operation is relying ever more on IT services and thus, predictable service delivery with time, performance and dependability constraints. In order to tame complexity and enable efficient design, operation and maintenance, various modeling techniques have been proposed. *Service-Oriented Architecture* (SOA) proposes a formalism where services are the basic building elements of system design. [4]

Meeting non-functional property requirements is crucial for successful service provision. However, non-functional properties like service availability are highly dependent on the properties of the underlying information and communications

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 39–53, 2013.

technology (ICT) infrastructure. This work focusses on *user-perceived service availability*: Given an ICT infrastructure with a set of providing service instances and a set of service clients. The user-perceived availability is the probability for a service provided by one or more of these instances to perform its required function when requested from a specific client.

To assess the user-perceived service availability for any client within the network, information about the system availability is not sufficient, because every specific pair service requester and provider can utilize different ICT components. This is why, although services are usually well-defined within business processes, assessing service availability remains uncertain. The underlying infrastructure varies according to the position of the *service requester* – represented by a person or even an information and communications technology (ICT) component – and the concrete *providing service instance*. Evaluation of user-perceived service availability should employ a model of the ICT infrastructure where service properties are linked to component properties.

One important concept in service-oriented architecture is composition, the possibility to combine the functionality of multiple services to provide more complex functionality as a composite service with a single interface. If the individual services within the composition are indivisible entities regarding their functionality, they can be called *atomic services*. For instance, an *email* service can be divided into atomic services *authenticate*, *send mail* and *fetch mail*. In this sense, *email* corresponds to a *composite service* constituted by the atomic services *authenticate*, *send mail* and *fetch mail*.

This paper provides a methodology to evaluate user-perceived service availability. Given a set of input models that describe the service network topology, its services and actors, it generates and solves specific *user-perceived service availability models* (UPSAM) for different user perspectives. These models are expressed as *reliability block diagrams* (RBD) and evaluate steady-state service availability. The evaluation can be useful when designing a service network to estimate the expected quality of service provision. After deployment, it can be used to detect bottlenecks in the network or to evaluate the impact of planned changes to the ICT infrastructure. The methodology could be extended to provide evaluation of different dependability properties that also cover dynamic network behaviour during service usage.

The following section provides an overview of related work. Section 3 states the scientific problem of evaluating user-perceived service availability, followed by an approach to solve it in Section 4. Given a service description, a network topology model and a pair service provider and requesting client, we employ a methodology to automatically identify relevant ICT components and from that generate the UPSAM. Finally, Section 5 demonstrates the feasibility of our approach in a representative case study by applying it to an exemplary email service within parts of the service network infrastructure of University of Lugano, Switzerland. We extract the UPSAM of that service for different service clients and calculate and compare their availability. Section 6 summarizes our work by pointing out the main contributions and remaining open issues.

2 Related Work

Service-Oriented Architecture (SOA) [4] provides a set of methodologies where system components are designed as interoperable services. In this paper, a service is defined according to [6] as "an abstraction of the infrastructure, application or business level functionality. It consists of a contract, interface, and implementation. [...] The service interface provides means for clients to connect to the service, possibly but not mandatory via network." The same authors define a composite service as a composition of basic indivisible services called atomic services, which are shaped according to their business functionalities. Their definition focuses on the optimal re-usability, in order to avoid redundant atomic services with similar or the same purpose. Milanovic et al. also propose a methodology for the automatic generation of service availability models based on run-time monitoring [7,5,6]. In their methodology, a configuration management database system collects information about the network topology for further service deployment and steady-state availability analysis.

A different definition of services is proposed by the Service Availability Forum (SAF) [11] through the *Availability Management Framework* (AMF). In their specification, AMF components are the basic entities of the framework and consist of a set of software or hardware resources. In contrast to Milanovic et al. in [5], where infrastructure and services are modeled independently, SAF describes AMF components as intrinsic service providers, which can be grouped into bigger logical units called service units (SU).

Salehi et al. [10] proposes a *UML-based AMF configuration language* (UACL) to facilitate the generation, analysis and management of the AMF configurations. The language has been implemented by means of a *Unified Modeling Language* (UML) [9] profile. *Dependability Analysis Modeling* (DAM) [2] consists also of a UML profile for dependability modeling. It correlates service and ICT components, and describes them with a complete set of properties, although no transformation is provided by the methodology.

The authors of [17] provide a stochastical model to assess user-perceived web servide availability and demonstrate that there can be significant differences between the system and user-perceived perspectives. In [12] a new status-based model to estimate user-perceived availability proposed. Both works do not model the providing infrastructure in detail, however.

In order to assess the service dependability from different user perspectives, an extraction of relevant network parts is presented in [3]. Given a model of the network topology, a service description and a pair service requester and provider, a model-to-model transformation is applied to obtain a *user-perceived service infrastructure model* (UPSIM): Given an ICT infrastructure that contains a providing service instance p_i and a service client c_j. The UPSIM is that part of the infrastructure which includes all components, their properties and relations hosting the atomic services used to compose a specific service provided by p_i for c_j. The approach in [3] uses a subset of UML elements as well as UML profiles and stereotypes to impose specific dependability-related attributes to ICT components.

The paper at hand builds on the work in [3] with a methodology to obtain as output a specific availability model expressed as *reliability block diagram* (RBD) to evaluate service availability for different user perspectives. The case study in Section 5 uses an implementation of that methodology that is extensively described in [8].

3 Problem Statement

The availability of a service, as any non-functional property, depends on the underlying ICT infrastructure required for service execution. Moreover, a service may require a different set of ICT components for each user perspective within the infrastructure, as the service can be invoked for different pairs of service requester and provider. Also, the topology and services may change due to reconfiguration, addition or removal of components, upgrades and so on.

This dynamicity represents one of the main challenges of availability evaluation, especially during run-time, when changes need to be instantly considered in the availability models. A methodology is needed to support the model-driven evaluation of user-perceived service availability. The methodology should include:

1. A model to describe the ICT infrastructure, including availability properties for each component.
2. A model to describe services in hierarchical manner.
3. A formalism to relate an abstract service to parts of a concrete deployed infrastructure for any specific user perspective.
4. A mechanism to generate and solve a *user-perceived service availability model* (UPSAM) for such a user perspective.

The complete methodology should be automated as much as possible to support quick model updates in dynamic environments and to eliminate human errors during update or upgrade procedures. Preferably, the methodology should be defined and implemented using well-known standards and open-source tools to support external verification and to facilitate its dissemination.

4 Methodology

Since user-perceived non-functional service properties depend on the underlying infrastructure, the methodology consists of generating a *user-perceived service availability model* UPSAM from a service description, a network topology and a mapping between them. The UPSAM is then evaluated using an external tool for availability analysis. Infrastructure and services are represented in UML models as in [3]:

– *Class diagrams* are used to describe structural units of the network (e.g.: routers, clients, servers), their properties and relations in distinct classes.

– *Object diagrams* describe a deployed network structure/topology composed of *class* instances, namely objects with all properties of the parent class, and *links* as instances of their relations.
– *Activity diagrams* are used for service description and represent the service as a flow of actions.

We are using the input model specifications from [3] but enhance the described workflow. Instead of generating a *user-perceived service infrastructure model* (UPSIM), we output a *reliability block diagram* (RBD) for that part of the network that is relevant for service provision for a specific pair service requester and provider. Figure 1 presents an overview of the methodology.

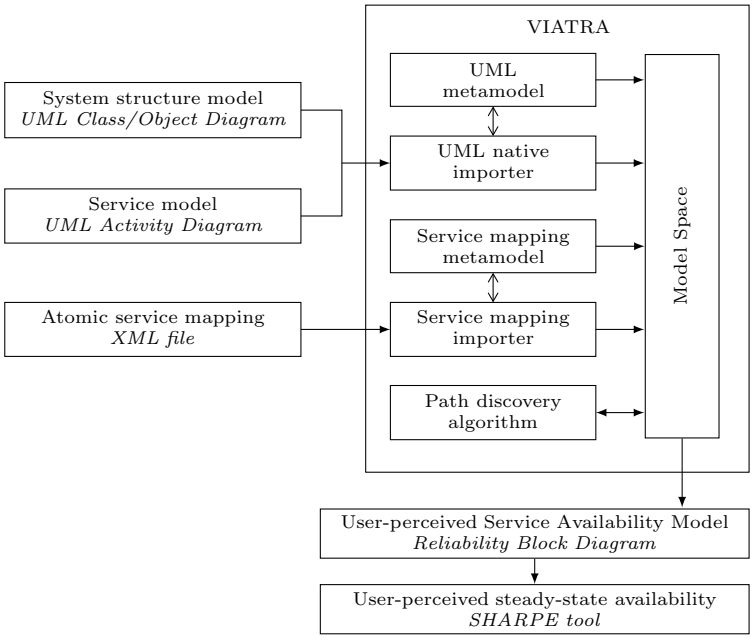


Fig. 1. Implementation of the model transformation

Following is a step-by-step description of the methodology. Steps 1 to 7 have been adapted from [3] for the scope of this work where necessary. Steps 8 to 10 are the main contribution of this work and are described in more detail in Section 4.1. Apart of the last Step 10, the workflow is based on the open-source development tool Eclipse [13], using both the UML2-compliant [9] modeling tool Papyrus [14] and the model transformation plug-in VIATRA2 [15]. Extensive details about the implementation of all steps can be found in [8].

1. Identify ICT components and create respective UML classes for each type. For subsequent availability analysis, an elementary UML availability profile

(see Figure 2) is applied to classes. This results in a class diagram containing the description of every ICT component.
2. Model the complete ICT infrastructure using UML object diagrams with instances of the classes from Step 1.
3. Identify and iteratively describe services using UML activity diagrams with atomic services as building blocks (Actions). This step results in a collection of service models with no correlation to the infrastructure.
4. Generate service mapping pairs by mapping atomic services from Step 3 to respective requester and provider ICT components from the infrastructure object diagram (Step 2).
5. Import ICT infrastructure and service UML models to the VIATRA2 model space. VIATRA2 creates entities for model elements, their relations and for atomic services.
6. Import service mapping pairs to the VIATRA2 model space using a custom service mapping importer.
7. For each atomic service, discover all acyclic paths between requester and provider, provided by the mapping in Step 4. Resulting paths are stored separately in the model space for further manipulation.
8. Generate atomic UPSAMs. For each atomic service, Paths extracted from Step 7 are merged into a single network topology, corresponding to the user-perceived service infrastructure. The atomic UPSAM is obtained as an RBD from that infrastructure.
9. Generate composite UPSAM. According to the service model from Step 3, the atomic UPSAMs are combined into a single RBD.
10. Calculate the user-perceived availability with the *Symbolic Hierarchical Automated Reliability and Performance Evaluator* (SHARPE)[16] using the composite UPSAM from the previous step.

Steps 1 to 3 are done manually using a UML modeling tool like Papyrus [14] and kept unaltered as long as the ICT infrastructure and services descriptions do not change. The mapping (Step 4) is a simple XML structure where changes will eventually be performed in order to analyze different user-perspectives on a service. This can be done manually or automated. Steps 5 through 10 are then fully automatable.

The availability profile presented in Figure 2 contains elementary properties required for stead-state availability analysis: *mean time between failures* (MTBF) and *mean time to repair* (MTTR). Additionally, the *redundantComponents* property specifies internal redundancy, which can be used to implicitly define a large set of ICT components into a single object in the infrastructure model.

4.1 User-Perceived Service Availability Model Generation

Since all the atomic services within a given composite service may be executed, the paths found in Step 7 are merged into one model which corresponds to the

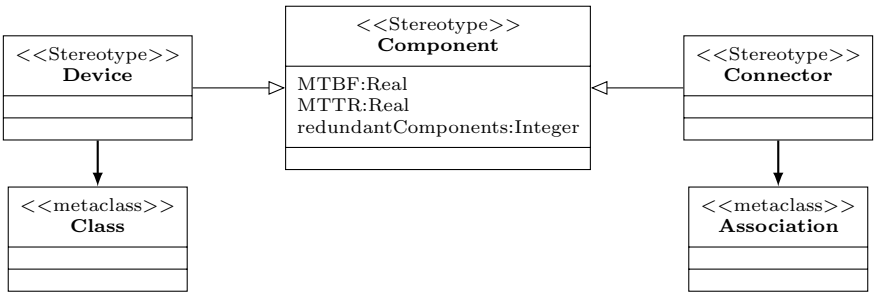


Fig. 2. Elementary availability profile

partial infrastructure required for proper service delivery of a given service pair. The UPSAM is a transformation of that partial infrastructure. The *instanceSpecifications* of the components within that partial infrastructure have the same signature as in the original ICT infrastructure from Step 1. Therefore, they maintain the same set of properties as the classes they instantiate. It is thus guaranteed that a subsequent availability analysis will find specific required properties for every element of the UPSAM.

As a consequence of Step 7, each atomic service has its own set of paths. All ICT components forming the path are translated into serialized blocks inside the RBD, given that all of them must be working in order to traverse the path. If an ICT component has n redundant components, the RBD will have n parallel blocks with the same characteristics. This corresponds to the *redundantComponents* property of the profile.

An atomic service is available if all ICT components of at least one of its paths are available. This introduces path redundancy inside the service network, and is represented within the RBD by placing blocks related to these paths in parallel. Identical blocks within those parallel paths are then merged into a single block. Let us demonstrate this using Figure 3 as an example of an ICT infrastructure model.

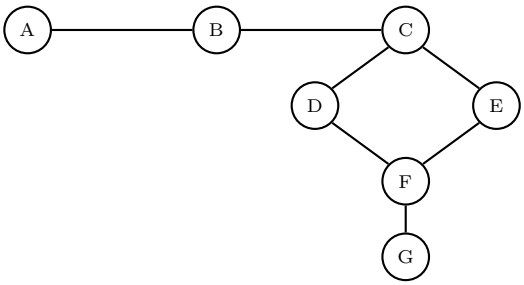


Fig. 3. ICT Infrastructure model example

All components have an internal redundancy of 0, only component B has an internal redundancy of 2 (`redundantComponent=2`). The following paths are identified from component A to G:

$A \rightarrow B \rightarrow C \rightarrow D \rightarrow F \rightarrow G$
$A \rightarrow B \rightarrow C \rightarrow E \rightarrow F \rightarrow G$

Components A, B, C and F, G are represented as a series of blocks, as they are common for both paths. Blocks D and E are in series within their respective paths but parallel to each other. Thus, they are represented as a pair of parallel blocks in between the two sequences obtained previously. Knowing that B has an internal redundancy of two extra components, it is represented as three parallel blocks B. The resulting RBD is presented in Figure 4. Note that the order of the blocks does not affect the resulting steady-state availability.

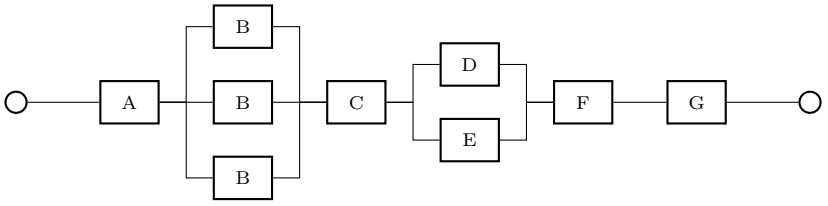


Fig. 4. Reliability Block Diagram of the example ICT infrastructure model

5 Case Study

We will now demonstrate the evaluation of user-perceived availability using the methodology from Section 4 with an exemplary *Send mail* service. It consists of resolving the *mail exchanger* (MX) address via the *domain name system* (DNS) and then sending an email message over that MX by means of the common *simple mail transfer protocol* (SMTP). During SMTP communication, the MX checks the credentials provided by the client with an external authentication server. This service represents a widespread use-case in today's service networks. In detail, the service is composed of three atomic services: *Resolve mail server address*, *Dispatch email via SMTP* and *Check authentication*. The UML activity diagram representing the *Send mail* flow of actions of the composite service is shown in Figure 5.

We simplify the fault model by taking only the steady-state availability of ICT components into account. This means we assume that all faults from classes *fail stop* to *byzantine*[1] are combined in the steady-state availability of the individual ICT components. We also disregard service discovery: The DNS server address is known a priori to the client as is the authentication server address to the MX.

[1] An ordered fault classification can be found in [1].

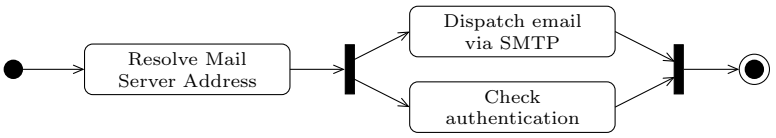


Fig. 5. Send mail service represented in UML activity diagram

The underlying network on which the service is deployed is based on the network of University of Lugano, Switzerland. The network core consists of the central switches with redundant connections and is nearly identical to the real infrastructure, while the tree-formed peripheral parts connected to the core have been reduced for demonstration purposes. As described in Section 4, the ICT infrastructure is represented by a UML object diagram, where each node is an instance of a specific ICT component class described in a UML class diagram. The links between nodes are also represented as instances of associations from the UML class diagram. For simplification purposes, associations are given the maximum availability of 1 – meaning that they are always available – so that their respective RBD blocks can be omitted in latter illustrations without affecting the steady-state availability. The full topology is shown in Figure 6.

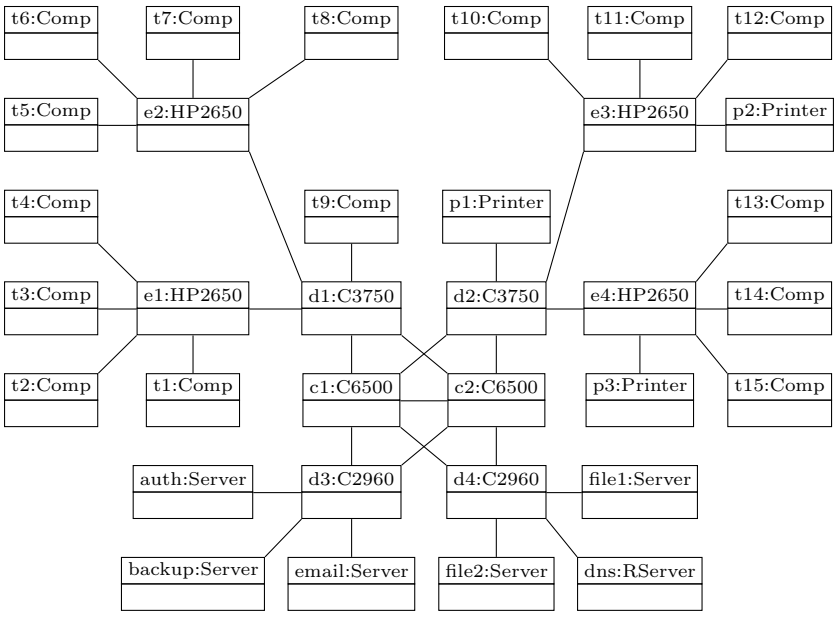


Fig. 6. Network infrastructure presented in UML Object Diagram

As an example for an ICT component description in UML, Figure 7 shows a fraction of the UML class diagram (Step 1 of the methodology) containing the description of the devices and their connections as, respectively, classes and associations. The type `RServer` represents a server containing an internal redundancy of one extra component (`redundantComponents=1`) which signifies that there are actually two servers with one server to fail over. Type `C2960` represents a switch.

Fig. 7. Predefined network elements represented in UML Class Diagram

The complete list of ICT components including relevant availability data is presented in Table 1. In the UML object diagram of the real topology in Figure 6 (Step 2 of the methodology) each node is represented by a unique identification and the respective type, in the format `id:Type`. Types *HP2650*, *C3750*, *C6500* and *C2960* are switches, the other types should be self-explanatory. Given that `RServer` has `redundantComponent=1`, the *dns* is then known to have redundancy although represented by a single node.

Table 1. Specification of ICT components

Type	Manufacturer	Model	MTBF(hours)	MTTR(hours)	RC*
C2960	Cisco	Catalyst 2960-48FPD-L	183498	0.5	0
C6500	Cisco	Catalyst 6500	61320	0.5	0
C3750	Cisco	Catalyst 3750G-24TS	188575	0.5	0
HP2650	Hewlett-Packard	ProCurve 2650	199000	0.5	0
Server	Dell	PowerEdge T620	60000	0.1	0
RServer	Dell	PowerEdge T620	60000	0.1	1
Comp	HP Compaq	DC7800	3000	24.0	0
Printer	Canon	IR3245N	2880	1.0	0

*redundantComponent

In this case study, the ICT components *t1* and *backup* were chosen as clients to compare two views on a composite service as perceived by different clients. Components *dns*, *email* and *auth* play the roles of dns server, mail server and authentication server. The mappings between atomic services and the ICT infrastructure (Step 4 of the methodology) for the clients *t1* and *backup* are given in Table 2 and Table 3, respectively. It can be seen that only minor changes to

Table 2. Service mapping pairs of the *Send mail* service for client *t1*.

Atomic Service	Requester	Provider
Resolve mail server address	t1	dns
Dispatch email via SMTP	t1	email
Check authentication	email	auth

Table 3. Service mapping pairs of the *Send mail* service for client *backup*.

Atomic Service	Requester	Provider
Resolve mail server address	backup	dns
Dispatch email via SMTP	backup	email
Check authentication	email	auth

the input models are necessary to change the user-perceived view on a service: Only the requesting instance in the mapping is changed, the network model and service description remain untouched.

In the following, we will demonstrate how to generate the UPSAM for the first atomic service, *Resolve mail server address*. Generation for the subsequent atomic services is omitted but will follow the exact same procedure. Starting from *t1* in the network shown in the infrastructure model of Figure 6, the path discovery algorithm (Step 7 in the methodology) identifies eight acyclic ways to reach *dns*:

t1→*e1*→*d1*→*c1*→*d4*→*dns*
t1→*e1*→*d1*→*c1*→*c2*→*d4*→*dns*
t1→*e1*→*d1*→*c1*→*d2*→*c2*→*d4*→*dns*
t1→*e1*→*d1*→*c1*→*d3*→*c2*→*d4*→*dns*
t1→*e1*→*d1*→*c2*→*d4*→*dns*
t1→*e1*→*d1*→*c2*→*c1*→*d4*→*dns*
t1→*e1*→*d1*→*c2*→*d2*→*c1*→*d4*→*dns*
t1→*e1*→*d1*→*c2*→*d3*→*c1*→*d4*→*dns*

Paths are then merged and transformed into a single reliability block diagram, the UPSAM, shown in the upper part of Figure 8. Basically, this procedure – corresponding to Step 8 of the methodology – reduces common nodes of different paths and excludes those which do not affect the overall availability of the service. For instance, in order to pass through *d2*, nodes *c1* and *c2* must be available, in addition to the common nodes *t1*, *e1*, *d1*, *d4* and *dns*. However, their availability implies that there is already at least one path guaranteed to be available between *d1* and *d4*. This is because associations have an availability of 1 and there are associations between *c1* and *d1*,*d4* as well as between *c2* and *d1*,*d4*. For this reason, the node *d2* does not affect the overall availability and is

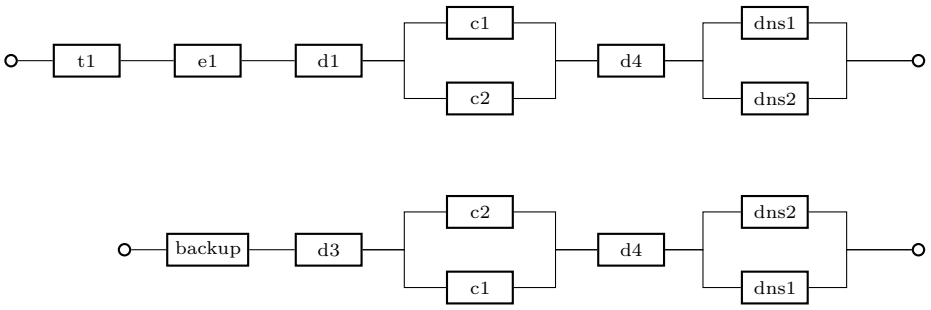


Fig. 8. *User-perceived service availablity models* (UPSAM) of atomic service *Resolve mail server address* for requesters *t1* and *backup*.

excluded from the UPSAM. Furthermore, redundant components are expanded: The *dns* component is converted into a pair of parallel blocks *dns1* and *dns2*. Figure 8 shows the UPSAM for requester *t1* side by side with the analogously created UPSAM for requester *backup*. We see only minor differences in the two models because to reach *dns*, both requesters have to use almost the same part of the network. Both have to traverse the network core, only the entry points are different. The next atomic service *Dispatch email via SMTP* paints a different picture. To reach the mail exchanger, requester *backup* does not need to traverse the network core, drastically reducing the number of blocks in the reliability block diagram. The UPSAM are depicted in Figure 9.

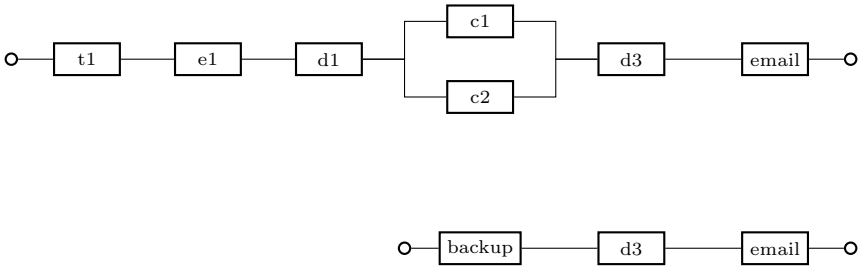


Fig. 9. *User-perceived service availablity models* (UPSAM) of atomic service *Dispatch email via SMTP* for requesters *t1* and *backup*.

Now, a composite UPSAM is created from the atomic UPSAMs according to the service description in Figure 5. Although the *Send mail* service described in the activity diagram contains a parallel execution, every single atomic service must be concluded in order to accomplish the execution of the composite service. For this reason, the resulting UPSAMs of the individual atomic services are put in series to compose the overall UPSAM of the composite service *Send mail*,

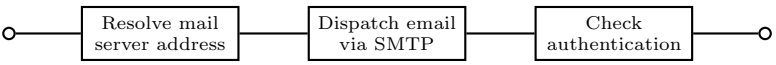


Fig. 10. Service availablity model of the *Send mail* service

as presented in Figure 10. This corresponds to Step 9 of the methodology. For the sake of clarity, atomic services have been combined into single blocks in the figure.

As the last step, we use SHARPE [16] to solve the obtained UPSAM to calculate the steady-state availability for the composite service *Send mail*. Results are shown in Table 4. We included results for the same service as requested by client *backup* to show how two different user perspectives on the same service differ in their availability.

Table 4. Service availability of *Send mail* service from different user perspectives

Service	Requester *t1*	Requester *backup*
Resolve mail server address (atomic)	0.999912118	0.999992884
Dispatch email via SMTP (atomic)	0.999910452	0.999993942
Check authentication (atomic)	0.999993942	0.999993942
***Send mail* (composite)**	**0.999816521**	**0.999980768**

In fact, altough the availability is reasonably high for both clients, it is ten times higher when the same service is requested by client *backup* instead of client *t1* (1.6 hours downtime per year for client *backup* versus 10 minutes for client *t1*). These differences are expected to be of a much higher magnitude in more heterogeneous networks with a significant variability in availability of the various component types, especially when taking into account different link qualities. This justifies the approach of considering user-peceived service availability.

6 Conclusion and Outlook

Assessing non-functional service properties like availability remains challenging. This is because service dependability depends highly on the properties of the providing ICT infrastructure. This infrastructure, however, changes for every different client. Thus, every client has another view on the service's availability. This is especially true in today's heterogeneous and widespread networks where the variability of availability among clients can be very high. Assessing system or service availability with aggregation functions might give an overview but falls short of providing a realistic picture of a service's dependability for specific clients.

We provided an automated methodology that evaluates user-perceived service availability. Given a set of input models – representing the network topology,

the service description and a pair requester, provider – that part of the ICT infrastructure providing the service for the given service pair is extracted and transformed into a reliability block diagram which is solved to obtain the steady-state availability of the given service. The methodology uses a hierarchical service model where on the highest level there is a composite service composed of atomic services which in turn map to ICT infrastructure components. The reliability block diagram for the client-specific infrastructure providing a composite service consitutes the *user-perceived service availability model* (UPSAM).

The case study demonstrates the feasibility of the approach by applying it to an exemplary mail service deployed on parts of the network of University of Lugano, Switzerland. We showed how the availability of the same service can differ considerably even in such a high-availability network when requested from two different users. The methodology is thus able to provide a fine-grained view on service availability as experienced from different points of the network.

Future work will focus on the complexity of the various methodology steps. Especially, the path discovery algorithm and creation of the composite reliability block diagram need optimization when applied to networks with a high degree of connectivity, such as wireless mesh networks. Also, combining sets of components with a low variability in user-perceived availability to reduce the size of the topology graph will be considered. Finally, extending the methodology to evaluate different dependability properties like interval availability, performability or responsiveness remains an open issue.

References

1. Barborak, M., Dahbura, A., Malek, M.: The consensus problem in fault-tolerant computing. ACM Computing Surveys 25(2), 171–220 (1993)
2. Bernardi, S., Merseguer, J., Petriu, D.: An UML profile for dependability analysis and modeling of software systems. Tech. Rep. RR-08-05, University of Zaragoza (May 2008)
3. Dittrich, A., Kaitovic, I., Murillo, C., Rezende, R.: A model for evaluation of user-perceived service properties. In: International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW). IEEE Computer Society (accepted for publication, May 2013)
4. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design, 1st edn. The Prentice Hall Service Technology Series from Thomas Erl. Prentice Hall PTR, Upper Saddle River (2005)
5. Malek, M., Milic, B., Milanovic, N.: Analytical availability assessment of IT services. In: Nanya, T., Maruyama, F., Pataricza, A., Malek, M. (eds.) ISAS 2008. LNCS, vol. 5017, pp. 207–224. Springer, Heidelberg (2008)
6. Milanovic, N., Milic, B.: Automatic generation of service availability models. IEEE Transactions on Services Computing 4(1), 56–69 (2011)
7. Milanovic, N., Milic, B., Malek, M.: Modeling business process availability. In: Congress on Services - Part I, pp. 315–321. IEEE Computer Society (July 2008)
8. Murillo, C.: Model-Driven Evaluation of User-Perceived Service Availability. Master thesis, Università della Svizzera Italiana (USI), Lugano, Switzerland (January 2013)

9. Object Modeling Group: Unified Modeling Language Infrastructure, version 2.4.1 (August 2011)
10. Salehi, P., Hamoud-Lhadj, A., Colombo, P., Khendek, F., Toeroe, M.: A UML-based domain specific modeling language for the availability management framework. In: 12th International Symposium on High-Assurance Systems Engineering (HASE), pp. 35–44. IEEE Computer Society (November 2010)
11. Service Availability Forum: Application Interface Specification (2011), http://www.saforum.org
12. Shao, L., Zhao, J., Xie, T., Zhang, L., Xie, B., Mei, H.: User-perceived service availability: A metric and an estimation approach. In: International Conference on Web Services (ICWS), pp. 647–654. IEEE Computer Society (July 2009)
13. The Eclipse Foundation: Eclipse development environment (March 2013), http://www.eclipse.org
14. The Eclipse Foundation: Papyrus UML modeling tool (March 2013), http://www.eclipse.org/modeling/mdt/papyrus
15. The Eclipse Foundation: VIATRA2, VIsual Automated model TRAnsformations (March 2013), http://www.eclipse.org/gmt/VIATRA2
16. Trivedi, K.S.: SHARPE (symbolic hierarchical automated reliability and performance evaluator) (February 2010), http://www.ee.duke.edu/~kst
17. Xie, W., Sun, H., Cao, Y., Trivedi, K.S.: Modeling of user perceived webserver availability. In: International Conference on Communications (ICC), vol. 3, pp. 1796–1800. IEEE Computer Society (May 2003)

Exploiting SDN Approach to Tackle Cloud Computing Security Issues in the ATC Scenario

Gabriella Carrozza[1], Vittorio Manetti[1], Antonio Marotta[2], Roberto Canonico[2], and Stefano Avallone[2]

[1] SESM s.c.a.r.l., Via Circumvallazione esterna di Napoli, Giugliano in Campania, 80014 Naples, Italy
{gcarrozza,vmanetti}@sesm.it

[2] University of Napoli Federico II, Dipartimento di Elettronica e Tecnologie dell'Informazione, Via Claudio, 21 - 80125 - Napoli, Italy
{stavallo,roberto.canonico,antonio.marotta}@unina.it

Abstract. Cloud Computing has been receiving great attention in the last few years due to the benefits it provides in terms of flexibility, scalability, virtualization and service provision. Nevertheless, many companies remain reluctant to such a cutting-edge technology due to the serious security issues affecting virtualized environments, especially in critical application scenarios where high safety and dependability levels are required. This work is aimed at discussing and presenting the main security threats for cloud computing infrastructures, as well as proposing a novel architecture in charge of reacting to security attacks in Infrastructure as a Service platforms. The basic idea is to migrate the attacked virtual appliance and to reconfigure the network by means of Software Defined Networking approach. The paper presents the architecture we have in mind and that will be deployed and validated against a real world distributed Air Traffic Control system, for which missing dependability and security targets would result in huge business and human losses.

1 Introduction

Cloud Computing (CC) is a model for enabling flexible and ubiquitous network access to a pool of shared computing resources. The Infrastructure as a Service (IaaS) service model has paved its way as a scalable, efficient and flexible solution since it allows consumers to deploy virtual resources such as networks, storage and virtual machines without any dependence on the physical infrastructure. In the case of private clouds, the cloud infrastructure and the shared resources are operated solely for a given organization which can even let cloud maintenance to third parties. By using the cloud, critical systems could be tested and reproduced, thus improving their dependability. Also, they can be rapidly reconfigured in case of failure by leveraging resources redundancy which characterizes any cloud infrastructure, with a direct return on business. Notwithstanding these incomparable benefits, CC adoption in critical industry is hampered by the security pitfalls it still exhibits, as well as by the the lack of mechanisms intended

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 54–60, 2013.

at increasing isolation and protection from internal and external threats. Cloud security issues have been widely studied by researchers in the last few years, in order to find effective solutions that can encourage critical systems industries to move towards such architectures.

This work comes from an industrial experience in the Air Traffic Control (ATC) field, in which a private cloud testbed has been set up according to the IaaS service model to reproduce real systems in house, as well as to investigate the possibility of interconnecting remote ATC centers through the cloud. It presents a novel architecture for detecting malicious activities and automatically implementing recovery strategies in the cloud infrastructure, when one of the virtual nodes is compromised. The idea is to define and realize mechanisms based on the so-called Software Defined Network (SDN) paradigm, thanks to which the forwarding plane is decoupled from the control plane, and the network behaviour can be easily programmed through a global view of the network itself. By automatically migrating virtual resources from the compromised node in a remote datacenter we are confident to increase the overall system security level.

The rest of the paper is organized as follows. Section II formalizes some of the most relevant cloud security issues, while section III briefly introduces the Software Defined Networking (SDN) approach and one of its implementation, namely the OpenFlow protocol. Section IV presents the state of the art and discusses the main related works that propose the use of such an approach for security purposes. Section V illustrates the proposed architecture for anomalies detection and for the implementation of the mitigation strategies. Last, section VI describes the real world critical system representing the case study for the architecture.

2 Security Issues in IaaS Cloud Computing Environment

A lot of papers in the literature propose interesting countermeasures to the most common security flaws in the IaaS Cloud Computing environments. One of the most discussed and well-known issues is data protection and availability: when the user entrusts his data to the cloud, he is not aware of the location where they will be stored and the way they will be treated. Different ways to use encryption are proposed, such as the one based on attributes: the decision of which users can decrypt a ciphertext is taken on the basis of the attributes and policies associated with the message and the user.

Although the great advantages that come along with the application of cloud computing, new security challenges related to the virtualization must also be taken into account. One of the most serious problem in the cloud infrastructure is related to the VMs image management risk: before even securing the running VM and the customer data, it is needed to be sure about the integrity of the virtual image which is about to be spawned in the cloud infrastructure. If an insider attacker gets access to the location where the VM images are stored, he has the chance to modify the way VMs will behave according to his malicious intention. That is why VMs images should be patched for security reasons and

scanned with the aim of finding malicious software. Another important task is the verification of the running VMs integrity and the behaviour of the components of the platform. The cloud infrastructure gives the chance to the end user to orchestrate, provision and manage all the implemented services through a set of APIs, that constitute the unique entry point for users to interact with the platform. Nevertheless, if the exposed APIs are vulnerable, there may arise lots of security threats, such as: clear text authentication, anonymous access, reusable token and weak access controls.

In this work, the focus is on a private cloud IaaS for which dependability and security issues are fewer than in public clouds. Here, indeed, network and data are stored and managed by third parties and off premises, differently from a private cloud environment. However, there still arise internal security threats: in this case, insider attacks come from the cloud platform users or the administrator himself.

3 OpenFlow and Software Defined Networking

Software Defined Networking (SDN) paradigm, which has rapidly changed the perspective of doing network research, is based on a sharp distinction between the infrastructure layer, composed by network devices, and the control layer where the network intelligence is deployed (the OpenFlow Controller). Using this approach, the forwarding plane is decoupled from the control plane, and the network behaviour can be easily programmed through a global view of the network. The OpenFlow Protocol constitutes the interface between the two layers, allowing to control and to define traffic management strategies to be performed by the switch devices in the infrastructure. OpenFlow is also attracting lot of interest in cloud computing platforms as a leading technology for implementing Networking as a Service. The need to have fully-virtualized networks becomes the main focus of the cloud computing community: being the hypervisor the heart of hardware-virtualization, it is needed to find solutions allowing to reach the same level of abstraction with physical network resources. Indeed OpenFlow can be used to guarantee the programmability of the networking level for the VMs. Concerning security requirements, the dynamic nature of CC systems makes the traditional solutions inefficient rising the need to find new approaches for protecting the infrastructure from different kind of attacks. OpenFlow can be considered as a leading technology for implementing an architecture that is aware of dynamic application security policies at a low cost.

4 Related Work

Some recent works propose OpenFlow as an effective solution for security and introduce OpenFlow-based platforms for implementing several security techniques. In [5], authors argue that the SDN paradigm can make the implementation of traffic anomaly detection easier by using the well-known NOX [6] OpenFlow controller in SOHO (small office/home office) networks. They implement four different anomaly detection algorithms as applications on the controller and point

out that SDN allows to implement line-rate detection of network vulnerabilities exploitation and also risk mitigation.

Braga *et al.* [7] face a well-known security issue, which is the Distributed Denial of Service. Their proposal aims at minimizing the overhead due to the extraction of network features used in the detection process, by requesting the flows to the OF switches.

Wang *et al.* [8] suggest a flexible security management architecture for large-scale production networks to overcome the drawbacks of the existing static solutions, by considering the peculiarities of data center networks. The architecture they propose is composed by a control component having the view of both the global state of the network, and the designed security policies, besides one or more security elements which are responsible for detecting anomalies in the traffic patterns. The programmability of the network flows enables the check of the global security policies and the fast reaction to alarms. However, such a distributed architecture requires a deep analysis to verify if the introduced latency (caused by the presence of the security elements) does affect user experience.

This work differs from the ones proposed in the literature, in the sense that it proposes an architecture in which the SDN paradigm is used to implement mitigation and recovery strategies in case of disasters and security breaches.

5 Description of the Proposed Architecture

Our idea is to design and implement an OpenFlow-based architecture in order to fit the dynamic nature of cloud computing infrastructures. It relies on the OpenvSwitch [9] technology, that is used as the virtual switch to provide connectivity to the virtual guests. OpenvSwitch implements a number of interesting features that led us to choose it since the architecture we are introducing relies on VLANs to guarantee level-2 isolation, and on the use of vNICs in bonding configuration for failover reasons. Moreover, the OpenvSwitch is OpenFlow 1.0 protocol compliant. We also evaluated some of the available open source OpenFlow Controllers and our choice fell on Floodlight [10], a Java event-based Controller. The features we took into account are the modularity of its core functionalities, the availability of REST APIs (which makes it consistent with the CC services provisioning) and its performances compared to the other available controllers.

As shown in Fig. 1, at the switch edge on the virtualization layer we use an Attack Detector agent, which is responsible of sniffing and analysing all the traffic coming from and arriving on the VLANs of the virtual networks. When abnormal activity is detected, the agent sends alerts through a secure channel (a Transport Layer Security socket) to a central Reaction Decider which is implemented on the OpenFlow Controller. The latter is also in charge of programming the flow tables of the OpenvSwitches and selecting the best countermeasure to adopt in relation to the severity level of the attack and the number of affected nodes. Starting from the assumption that the Cloud infrastructure is made of geographically distributed datacenters (as you can see in the cloud layer),

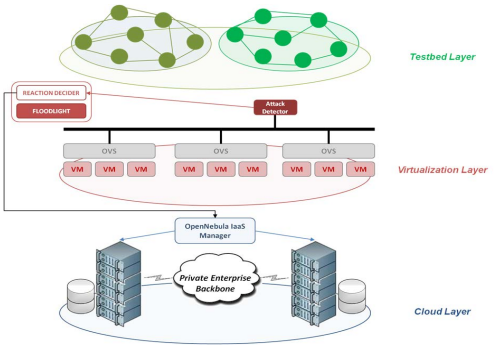


Fig. 1. Different views of the architecture

once an attack is detected, a mitigation and recovery strategy for the involved nodes is triggered. Such a strategy consists in dynamically activating a migration of the virtual guests under attack in a remote datacenter belonging to the same cloud infrastructure by interacting with the cloud platform manager. Once the migration process is terminated, the Floodlight Controller is in charge of programming the OpenvSwitches' flow tables in order to redirect the traffic related to the migrated node towards its new location. The aim is to guarantee transparency of the virtual appliance, so that a legitimate user or machine can access the services hosted on the attacked node without being aware of the migration process. Moreover, in order to further increase network security in the connection between the distributed data-centers, we use a mechanism that splits packets into parts and then redirects them to disjoint paths, so that an intruder is not able to reconstruct the flowing traffic. This is done by using a traffic engineering mechanism based on the MPLS (MultiProtocol Label Switching) technology [11]. The confidentiality is guaranteed because if a malicious user is able to intercept the traffic between two nodes of the network passing through different paths, he should be aware of the splitting mechanism in order to reconstruct the original message from the parts he has collected. Besides the mechanism we use is not overhead-heavy compared to the use of end-to-end encryption.

6 Case Study: A Testbed for Air Traffic Control

Air Traffic Control (ATC) are very demanding and software-intensive systems. They are safety critical, highly distributed and hard real time. Among the dimensional architectural requirements of this type of systems there are ultrahigh availability (6 nines), high performance, modifiability, scalability and usability. In the ATC field, ATC centers belonging to the same system are often deployed over different cities in a given country, either for fault tolerance purposes and remote connection needs at country level. For this reason, CC represents the key technology these industries need: first, setting up an extended enterprise private

CC platform allows to connect geographically distributed ATC centers for dependability purposes, e.g., by realizing a failover configuration among centers in order to increase overall system availability. Second, it can be leveraged in pre-operational phases by setting up testbed platforms in the cloud to perform distributed testing campaigns on complex systems from different premises, to reproduce real world scenarios in house and to validate the system in a number of operational use cases.

The use case scenario in which we intend to test and evaluate the proposed OpenFlow-based architecture is a Private Enterprise CC Infrastructure that hosts an Area Control Center (ACC), the main operative center in an ATC system. ACC is the infrastructure responsible for controlling aircraft in a particular volume of airspace at high altitudes between airport approaches and departures. By using this case study, we aim at validating our architecture on a very complex real world system which is one of the main industrial assets. In order to perform a preliminary assessment of the whole architecture, we implemented a proof of concept deploying an ACC (accounting for a total of 32 nodes) on a Private IaaS Infrastructure realized by using the OpenNebula [14] OpenSource solution. We simulated some Distributed Denial of Service attacks and used the Snort [12] Intrusion Detection system to trigger the mitigation strategy already presented. We are now investigating the presence of vulnerabilities of the testbed nodes which can be exploited as an attack surface. To this aim we exploited a linux-based distribution, namely Backtrack [13], which is widely used for penetration testing and security assessment.

7 Conclusions and Future Work

CC perfectly fits IT companies' needs for elasticity and scalability by extremely reducing CapEx/OpEx costs and datacenter start-up time. Anyway there are still open research issues about the security level and performances achievable when moving services in the cloud. In this work we built a private Enterprise CC platform to host an entire Air Control Center and then we designed an architecture with the aim of automatically reacting to attacks towards the ACC nodes. For the proof of concept, we used a simple DDoS attack which is detected by the agent and the raised alarm is sent to a central decider, which is in charge of triggering the mitigation strategy. The decider is implemented on an OpenFlow Controller which has a global view of the network and it interacts with the cloud manager in order to activate the migration of the attacked node in a remote data-center. In the connection between the two data-centers, confidentiality is guaranteed thanks to the use of an MPLS-based splitting mechanism which makes the eavesdropping of the packets very hard to possible attackers. Finally, the Controller can then reconfigure traffic flows in order to guarantee the transparency of the location of the node after the mitigation process. Our future work consists in an evaluation of the proposed architecture and the use of classical anomaly detection with other mechanisms aimed at identifying malicious patterns on a per-user or per-application basis.

Acknowledgements. This work has been partially supported by MIUR under Project "SVEVIA" (PON02_00485_3487758) of the public-private Laboratory "COSMIC" (PON02_00669).

References

1. McKeown, N., et al.: OpenFlow: Enabling Innovation in Campus Networks. ACM SIGCOMM Computer Communication Review Archive 38(2), 69–74 (2008)
2. Bindra, G.S., et al.: Cloud Security: Analysis and Risk Management of VM Images. In: 2012 International Conference on Information and Automation (ICIA), June 6-8, pp. 646–651 (2012)
3. Lombardi, F., Di Pietro, R.: Secure Virtualization for Cloud Computing. Journal of Network and Computer Applications 34(4), 1113–1122 (2011)
4. Yu, T.-T., Zhu, Y.-G.: Research On Cloud Computing and Security. In: 2012 11th International Symposium on Distributed Computing and Applications to Business, Engineering & Science (DCABES), October 19-22, pp. 314–316 (2012)
5. Mehdi, S.A., Khalid, J., Khayam, S.A.: Revisiting Traffic Anomaly Detection Using Software Defined Networking. In: Sommer, R., Balzarotti, D., Maier, G. (eds.) RAID 2011. LNCS, vol. 6961, pp. 161–180. Springer, Heidelberg (2011)
6. Nox Controller, `http://www.noxrepo.org/`
7. Braga, R., et al.: Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow. In: 2010 IEEE 35th Conference on Local Computer Networks (LCN), October 10-14, pp. 408–415 (2010)
8. Wang, K., et al.: LiveSec: Towards Effective Security Management in Large-scale Production Networks. In: 2012 32nd International Conference on Distributed Computing Systems Workshops (ICDCSW), June 18-21, pp. 451–460 (2012)
9. `http://openvswitch.org/`
10. Floodlight Controller, `http://floodlight.openflowhub.org/`
11. Stefano, A., et al.: A Splitting Infrastructure For Load Balancing and Security in an MPLS Network. In: 3rd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities, May 21-23, pp. 1–6 (2007)
12. Roesch, M.: Snort, Lightweight Intrusion Detection For Networks. In: 13th USENIX Systems Administration Conference (LISA 1999), Seattle, WA (November 1999)
13. `http://www.backtrack-linux.org/`
14. `http://opennebula.org/`

Intercept: Profiling Windows Network Device Drivers*

Manuel Mendonça and Nuno Neves

University of Lisboa, Faculty of Sciences, LASIGE, Portugal
manuelmendonca@msn.com, nuno@di.fc.ul.pt

Abstract. Device drivers account for a substantial part of the operating system (OS), since they implement the code that interfaces the components connected to a computer system. Unfortunately, in the large majority of cases, hardware vendors do not release their code, making the analysis of failures attributed to device drivers extremely difficult. Although several instrumentation tools exist, most of them are useless to study device drivers as they work at user level. This paper presents Intercept, a tool that profiles Windows Device Drivers (WDD) and logs the driver interactions with the OS core at function level. The tool helps to understand how a WDD works and can provide support for several activities, such as debugging, robustness testing, or reverse engineering. Experiments using Ethernet, Wi-Fi and Bluetooth device drivers show that Intercept is able to record function calls, parameters and return values, with small overheads even when the device driver under test is subject to a heavy workload.

Keywords: Device drivers, profiling, dependability.

1 Introduction

Device drivers (DD) play an important role in the computer industry as they are responsible for interfacing the multitude of devices that can be connected to a system. Therefore, their aggregated size can be a substantial part of modern operating systems. Nevertheless, most system administrators, users, and programmers still view them as an obscure and complex section of the operating system, which in part can be explained due to the DD necessity of addressing low level hardware details and OS internals. In the past, DD misbehavior has been pointed out as a prime cause for system crashes [3], and some researchers have showed that faults in DD can have a strong impact in the overall system dependability [4,5,6,7].

The recognized complexity associated with DD is aggravated as most vendors do not release openly the code, or even the hardware specifications. Therefore, the development, testing and analysis of DD becomes a complex task, and typically can only be achieved through the use of reverse engineering techniques and other forms of instrumentation. Although several instrumentation tools exist, most of them work at user level, making them useless to study the device drivers' behavior.

* This work was partially supported by the EC through project FP7-257475(MASSIF), by the FCT through the Multiannual program and project PTDC/EIA-EIA/113729/2009(SITAN).

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 61–75, 2013.

In the paper we present Intercept, a tool that instruments Windows Device Drivers (WDD) by logging data about the interactions with the OS core. It operates without access to the driver's source code and with no changes to the driver's binary file. As its name indicates, the tool intercepts all function calls between the DD and the OS core, ensuring that various data can be collected, such as the name of the functions that are invoked, their parameters and return values, and the content of particular areas of memory. Although simple in concept, it enables the users to expose a DD behavior and data structures, which provide a practical approach towards its understanding.

In the case of DD involved with communications, which are the focus of the paper, Intercept can be used as a building block of other tools by providing the contents of packets and the context of their arrival/departure. For this purpose, Intercept can log the network traffic information in the format used by Libpcap [17], which can then be analyzed by popular tools such as WireShark [18]. Intercept can be very helpful in debugging processes since it gives a higher level vision of what is happening between the OS core and the driver, and at the same time offering information on the parameter contents and address locations. Combined with debugging tools from Microsoft, such as WinDbg [20], this data is useful to reduce the time for locating functions, OS resources and global variables. Currently, we are using Intercept as a component of a testing tool for DD. Some preliminary results are presented at the end of the paper.

2 Related Work

In the past, several tools have been proposed for various types of code analysis. For example, CodeSurfer [11] can perform program slicing to support a better understanding of the code behavior. BitBlaze [10] combines dynamic and static analysis components to extract information from malware. Other tools like Coverity [12], Path Finder [14] or CoreDet [13] rely either on C or Java language constructs and LLVM compilers [15] to transform the source code to their analysis format. Unfortunately, these tools depend on the existence of the source code. Binary programs have been addressed by RevGen [16], which translates them to LLVM intermediate representations, enabling the code to be checked with off-the-shelf analysis tools.

These tools, although producing valuable information, only reveal a part of the scope of the analysis, which is the static organization of the software. To obtain a vision over the dynamic behavior of the component, it is usually necessary to resort to debuggers or instrumentation tools that are able to trace the execution and record the instructions that were run. SytemTap [21] and Ftrace [22] are examples of existing tracing tools, but they only support the Linux OS.

Detours [1] is a library for intercepting arbitrary Win32 binary functions on x86 machines. The interception code is applied dynamically at runtime by replacing the first few instructions of the target function with an unconditional jump to a user-provided detour function. The removed instructions from the target function are preserved in a trampoline function, which also has an unconditional branch to the remainder of the target function. The detour function can either completely replace the target function or extend its semantics by invoking the target function as a

subroutine through the trampoline. Detours experiments were based on Windows applications and DLLs, but were not applied to device drivers.

PIN [2] is a software system that performs run-time binary instrumentation of Windows applications. PIN collects data by running the applications in a process-level virtual machine. It intercepts the process execution at the beginning and injects a runtime agent that is similar to a dynamic binary translator. To use PIN, a developer writes a "Pintool" application in C++ using the PIN API consisting of instrumentation, analysis and callback routines. The "Pintool" describes where to insert instrumentation and what it should do. Instrumentation routines walk over the instructions of an application and insert calls to analysis routines. Analysis routines are called when the program executes an instrumented instruction, collecting data about the instruction or analyzing its behavior. Callbacks are invoked when an event occurs, such as a program exit. Several applications were instrumented using PIN, such as Excel and Illustrator. PIN executes in user level ring3, and therefore can only capture user-level code. DynamoRio [24] is an example of dynamic binary translation technique similar to the one used by PIN [2].

NTrace [23] is a dynamic tracing tool for the Windows kernel capable of tracing system calls, including the ones involving drivers. The used technique is based on code modification and injection of branch instructions to jump to tracing functions. It relies on the properties introduced by the Microsoft Hot patching infrastructure, which by definition start with a `mov edi, edi` instruction. NTrace replaces this instruction with a two-byte jump instruction. However, due to the space constraints, the jump cannot direct control into the instrumentation routine. It rather redirects to the padding area preceding the function. The padding area is used as a trampoline into the instrumentation proxy routine.

Intercept uses an alternative approach to instrument device drivers in Windows, which requires no changes to the binary code and supports callbacks. It uses a DD loader to point all imported functions from a driver to its own interception layer. Callback functions registered by the driver are also captured and directed to the interception layer. No extra code needs to be developed for normal operation --- a complete log is generated describing how the driver behaves as a result of the experiments. However, extensibility is achieved by changing the actions performed by the interception layer, allowing more complex operations to be carried out.

3 Device Drivers

DD are extensible parts of the OS, exporting interfaces that support the interactions with the hardware devices. They are called when either the OS requires some action to be carried out by the device or the other way around. Depending on the type of device, the DD can operate in two different ways. In the first one, the DD accesses the device in a periodic fashion (pooling) --- the DD programs a timer with a certain value and whenever the timer expires the device is checked to see if it needs servicing (and proceeds accordingly). In the second way, the device triggers an interrupt to request the processor's attention. Each interrupting device is assigned an identifier called the interrupt request (IRQ) number. When the processor detects that an

interrupt has been generated on an IRQ, it stops the current execution and invokes an interrupt service routine (ISR) registered for the corresponding IRQ to attend to the request of the device. In either case, these critical pieces of code must be quickly executed to prevent the whole system from being stopped.

The rest of this section provides context on the operation of WDD. Some of this information was obtained by reading available literature, while other had to be discovered by reverse engineering the operation of Windows.

3.1 Windows Device Drivers

The *Windows Driver Model (WDM)* defines a unified approach for all kernel-mode drivers. It supports a layered driver architecture in which every device is serviced by a driver stack. Each driver in this chain isolates some hardware-independent features from the drivers above and beneath it avoiding the need for the drivers to interact directly with each other. The WDM has three types of DD, but only a few driver stacks contain all kinds of drivers:

- *Bus driver* – There is one bus driver for each type of bus in a machine (such as PCI, PnP and USB). Its primary responsibilities include: the identification of all devices connected to the bus; respond to plug and play events; and generically administer the devices on the bus. Typically, these DD are given by Microsoft;
- *Function driver* – It is the main driver for a device. Provides the operational interface for the device, handling the read and write operations. Function drivers are typically written by the device vendor, and they usually depend on a specific bus driver to interact with the hardware;
- *Filter drivers* – It is an optional driver that modifies the behavior of a device. There are several kinds of filter drivers such as: lower-level and upper-level filter drivers that can change input/output requests to a particular device.

The WDM specifies an architecture and design procedures for several types of devices, like display, printers, and interactive input. For network drivers, the *Network Driver Interface Specification (NDIS)* defines the standard interface between the layered network drivers, thereby abstracting lower-level drivers that manage hardware from upper-level drivers implementing standard network transports (e.g., the TCP protocol). Three types of kernel-mode network drivers are supported in Windows:

- *Miniport drivers* - A *Network Interface Card (NIC)* is normally supported by a miniport driver that has two basic functions: manage the NIC hardware, including the transmission and reception of data; interface with higher-level drivers, such as protocol drivers through the NDIS library. The NDIS library encapsulates all operating system routines that a miniport driver must call (functions `NdisMXxx()` and `NdisXxx()`). The miniport driver, in turn, exports a set of entry points (`MPXxx()` routines) that NDIS calls for its own purposes or on behalf of higher-level drivers to send packets.
- *Protocol Drivers* - A transport protocol (e.g. TCP or IP) is implemented as a protocol driver. At its upper edge, a protocol driver usually exports a private interface to its higher-level drivers in the protocol stack. At its lower edge, a protocol driver

interfaces with miniport drivers or intermediate network drivers. A protocol driver initializes packets, copies data from the application into the packets, and sends the packets to its lower-level drivers by calling `NdisXxx()` functions. It also exports a set of entry points (`ProtocolXxx()` routines) that NDIS calls for its own purposes or on behalf of lower-level drivers to give received packets.

- *Intermediate Drivers* - These drivers are layered between miniport and protocol drivers, and they are used for instance to translate between different network media. An intermediate driver exports one or more virtual miniports at its upper edge. A protocol driver sends packets to a virtual miniport, which the intermediate driver propagates to an underlying miniport driver. At its lower edge, the intermediate driver appears to be a protocol driver to an underlying miniport driver. When the miniport driver indicates the arrival of packets, the intermediate driver forwards the packets up to the protocol drivers that are bound to its miniport.

Windows drivers expose functions that provide services to the OS. However, only one function is directly known by the OS, as it is the only one that is retrieved from the binary file when the driver is loaded. By convention, the function name is `DriverEntry()`. This function is called when the OS finishes loading the binary code of the driver, and its role is to initialize all internal structures of the driver and hardware, and indicate to the OS the exported driver functions by calling `NdisMRegisterMiniportDriver()`. Example exported miniport driver functions to NDIS are: `MPInitialize()` and `MPSendPackets()`.

Generically, a packet transmission is accomplished in a few steps with NDIS. The protocol driver sends the packet by calling NDIS function `NdisSendPackets()`, which in turn passes the packet to the miniport driver by invoking `MPSendPackets()` exported by the miniport driver. The miniport driver then forwards the packet to the NIC for transmission by calling the associated `NdisSendPackets()`. On the other way around, when a NIC receives a packet, it can post a hardware interrupt that is handled by NDIS or the NIC's miniport driver. NDIS notifies the NIC's miniport driver by calling the appropriate `MPXxx()` function. The miniport driver sets up the data transfer from the NIC and then indicates the presence of the received packet to higher-level drivers by calling the `NdisMIndicateReceivePacket()`. The upper level protocol driver then calls `NdisReturnPacket()` to retrieve the packet.

3.2 Windows Device Drivers File Structure

Windows normally organizes the information about a DD in several files. Files with the extension ".inf" contain plain text and are divided in several sections. They have relevant context data such as the vendor of the driver, the type and the compatibility with devices, and startup parameter values. They are used during driver installation to match devices with drivers and to find the associated ".sys" files. Files with the extension ".sys" are the binary executable images of the DD, and they are loaded to memory to provide services to the OS. The binary files follow the PEF file format [8], the same format used to represent applications and DLLs.

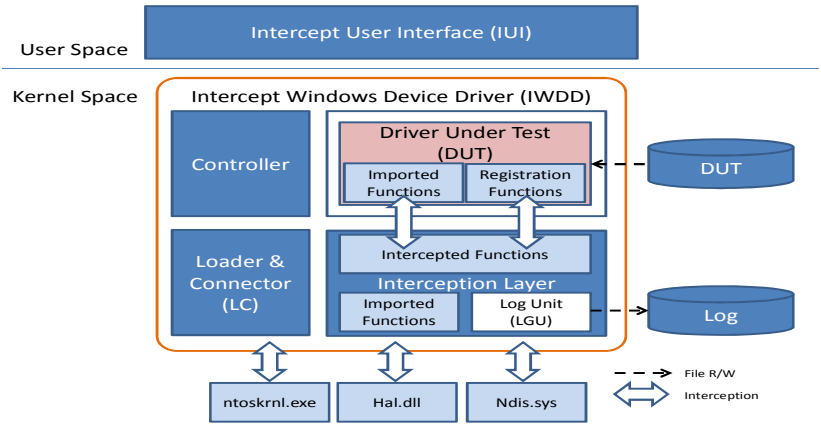


Fig. 1. Intercept architecture

The PEF file structure contains binary code and dependencies from other software modules (organized as tables). The binary code is mostly ready to be loaded into memory and run. However, since it can be placed anywhere in memory, there is the need to fix up the relative addresses of the function calls. Functions that refer to external modules are located in the imported functions table. This table contains the names of the external modules (DLLs, .sys, .exe), the function names and the address location in the memory of the running system. The addresses are resolved by the Windows Driver Manager when it loads the driver for execution.

The driver is placed in execution by calling the `DriverEntry()` function. The address of this function is also obtained from the PEF file, and is located in the `AddressOfEntryPoint` field of the Optional Header section.

4 Intercept

Intercept logs information about the interactions between the OS core and the device driver under test (DUT). The data is collected during the whole period of execution, starting when the driver is loaded and ending when it is uninstalled. It includes among others, the list of functions that are used, the order by which they are called, and parameter and return values. This information is quite comprehensive, and it helps not only to understand the driver-OS core interactions, but also to realize how drivers deal with the hardware in terms of programming and access to specific storage areas.

4.1 Architecture

The architecture of Intercept is represented in Fig. 1. It can be divided in two main components: the *Intercept Windows Device Driver (IWDD)* and the *Intercept User Interface (IUI)*. The first is a Windows driver that provides all the necessary functions to load, execute and intercept the DUT. The second is an application that allows users to setup the interception process and control the IWDD activity.

The components of IWDD are the following. The *Controller* provides an interface for the IUI application to control the behavior of the IWDD, allowing for instance the definition of the level of detail of logging and the selection of which functions should be logged. The *Loader & Connector* (LC) is responsible for loading the "DUT.sys" file into the memory space of IWDD. It also links all functions that the DUT calls from external modules to the functions offered by the Interception layer. The *Interception Layer* provides the environment for the DUT to run, and intercepts all calls performed by the OS to the DUT and the other way around. The *Log Unit (LGU)* receives the log entries from the Interception layer and saves them to a file. This is performed in a separate task to decouple the write delays from the remaining processing, and therefore increase the system performance.

Intercept is installed by replacing in the system the DUT with its own driver (the IWDD). When the OS attempts to load the DUT, in fact it ends up loading IWDD. Later on, IWDD brings to memory the DUT for execution. Setting up the interception of a DUT involves the following steps:

1. The user indicates the DUT of interest through the IUI interface, where a list of devices present in the OS is displayed;
2. The IUI locates the "DUT.inf" and "DUT.sys" files, and makes a copy of them to a predefined folder. A copy of the "IWDD.sys" file is also placed in the same folder;
3. The IUI replaces in the "DUT.inf" file all references to "DUT.sys" with "IWDD.sys". The IUI also removes references to the security catalogue, since IWDD is not currently digitally signed. This way, when the OS interprets the "DUT.inf" file, it will install "IWDD.sys" instead;
4. The Windows Device Manager (WDM) is used to uninstall the "DUT.sys", and then it is asked to check for new hardware, to detect that there is a device without a driver. At that time, the location of the predefined folder is provided, and Windows interprets the modified "DUT.inf" file. Since there is a match with the hardware identification of the device, it proceeds to load "IWDD.sys".

4.2 Start Up Process for Interception

After "IWDD.sys" is loaded, the following sequence of actions occurs:

1. The WDM calls the `DriverEntry(DriverObject *drvObj, PUNICODE_STRING RegPath)` function of IWDD, so that it can initialize and register the callback functions. Parameter `*drvObj` is a complex structure where some of the exported callback functions can be registered. Parameter `RegPath` is the path of the Windows Register location where the driver should store information. Since the DD functionality is to be provided by the original DUT implementation, at this stage the control is given to the LC unit to load the DUT's code;
2. The LC unit interprets the "DUT.sys" file contents, relocates the addresses, and goes through the table of imported functions to link them to the Interception layer. Technically this is achieved by having in the Interception layer a table containing entries with a "name" and an "address" for each function. The "name" is the Windows function name that can be found in the imported table of the DUT and the

"address" is a pointer to the code of the function. The address of the function in the Interception layer is placed in the imported function table of the DUT's. In the end, all imported functions of the DUT point to functions in the IWDD.

3. Next, the "DUT.sys" binary is merged and linked to the IWDD. The LC unit also finds the address of the DUT's `DriverEntry()`, which is then executed. As with any other driver, the DUT has to perform all initializations within this function, including running `NdisMRegisterMiniportDriver()` to register its exported functions to handle packets. However, since the DUT's imported functions were substituted by IWDD functions, a call to `NdisMRegisterMiniportDriver()` in fact corresponds to a call to `_IWDD_NdisMRegisterMiniportDriver()`[1]. In the particular case of this function, the DUT gives as parameters the callback functions to be registered in the NDIS library. In the Interception layer, the implementation of this function swaps the function addresses with its own functions, making the interception effective also for functions that will be called by the OS to the DUT.
4. When the DUT's `DriverEntry()` finishes, it returns a `drvObj` parameter containing potentially also some pointers to callback functions. Therefore, before giving control back to the OS, IWDD replaces all callback entries in `drvObj` with its own intercept functions, which in turn will call the DUT's routines. This way this type of callback function is also intercepted.

4.3 Tracing the Execution of the DUT

The DUT starts to operate normally, but every call performed by the OS to the DUT, and vice versa, is intercepted. The Interception layer traces all execution of the DUT, recording information about which and when functions are called, what parameter values are passed, which return values are produced and when the function exits. The log uses a plain text format and data is recorded to a file.

All functions implemented in the Interception layer make use of routines `_IWDD_DbgPrint()` and `_IWDD_Dump(char *addr, long size)`. The first works like the C language `printf()` function, and is used to write formatted data to the log file, such as strings and other information types. The second function is used to dump into the log file the contents of memory of a certain range of bytes starting at a given memory addresses. Together, these two functions can give a clear insight of the DUT's and OS's interaction.

Typically, the Interception layer creates a log entry both when entering and leaving a function. Whenever input parameter values are involved, they are also logged before calling the intended function, either in the DUT's code or in the OS. Output parameters and return values are saved before the function ends execution. Complex structures, such as `NetBuffers`, `NetBufferLists` or `MDLs` [9], are decomposed by specific routines so that the values in each field of the structure can be stored.

The interception of functions and the trace of its related information is a time consuming activity that may interfere with the DUT and the overall system performance.

[1] The prefix _IWDD_ is used to identify a function provided by the IWDD.

To reduce overheads, the storage process is handled by a separate thread. During the IWDD startup process, the LGU unit creates a queue and a dedicated thread (`DThread`), whose task is to take elements from the queue and write them into the log file. The queue acts as a buffer to adapt to the various speeds at which information is produced and consumed by the thread. The access to the queue is protected by a lock mechanism to avoid race conditions. A call to `_IWDD_DbgPrint()` or `_IWDD_Dump()` copies the contents of the memory to the queue, and signals the thread to wake up and store the information.

In the standard mode of operation, the log file is created when the thread is initiated. Each time the thread awakes, the data is removed from the queue and written to the file. When the file reaches a pre-determined value, it is closed and a new one is created. However, in case of a crash, the information in cache can be lost. To cope with this situation, the thread can also be configured to open, write synchronously and close the file each time it consumes data from the queue. However, this comes at the expense of a higher overhead.

5 Experimental Results

The objective of the experiments is twofold. First, we want to get some insights into the overheads introduced by Intercept, while a DD executes a common network task --- a file transfer by FTP. Second, we want to show some of the usage scenarios of the tool, such as determining which functions are imported by the drivers and what interactions occur while a driver runs.

5.1 Test Environment

The experiments were performed with three standard drivers, implementing different network protocols, namely Ethernet, Wi-Fi and Bluetooth. Table 1 summarizes the installation files for each DUT.

Table 1. Device drivers under test

Driver Type	Files
Ethernet	netrtx32.inf, rtlh.sys
Wi-Fi	netathr.inf, atrh.sys
Bluetooth	netbt.inf, btnetdrv.sys

The corresponding hardware devices were connected to a Toshiba Satellite A200-263 Laptop computer. The Ethernet and Wi-Fi cards were built-in into the computer, while the Bluetooth device was a SWEEX Micro Class II Bluetooth peripheral [19] linked by USB. In the tests, we have used Intercept both with Windows Vista and Windows 8.

The overhead experiments were based on the transmission of a file through FTP. The FTP server run in an HP 6730b computer. The FTP client was the Microsoft FTP client application, which was executed in the laptop together with Intercept. Different

network connections were established depending on the DUT in use. For the Ethernet driver an Ethernet network of 100Mbps using a TP-Link 8 port 10/100Mbps switch was setup to connect the two systems. For the Wi-Fi and Bluetooth drivers an ad-hoc connection was established.

5.2 Overhead of Intercept

To evaluate the overheads introduced by Intercept, we have run a set of experiments consisting on the transfer of a file of 853548 bytes length between a FTP server and a client. Any file could have been used for the transfer. We selected this file because it was the first log produced by Intercept during the experiments.

For each driver five FTP transfers were performed, and the average results are presented in the tables. Table 2 summarizes the results for the execution time and transfer speeds. Column "Driver ID" represents the DUT, either in Windows Vista (xx_Vista) or in Windows 8 (xx_Win8). The columns under the label "Intercept off" display the average transfer time and average speed when the Intercept tool is not installed in the client system. The columns under label "Intercept on" correspond to the case when the Intercept tool is being used.

The results between Intercept off and on show a performance degradation, which was expected as Intercept records all the activity of the drivers, and performs tasks such as decoding parameter structures and return values of all functions. Nevertheless, these overheads are relatively small: between 2% and 7% for the Ethernet driver, 2% to 3% for the Bluetooth driver and 14% to 15% for the Wi-Fi driver. These observations were more or less expected since the Wi-Fi drivers have more imported functions, are longer in size and require more processing when compared with the other drivers. The same Bluetooth driver was used in both OS which can explain the similarity of the degradation. The differences between the overheads on the Ethernet and Wi-Fi networks can be related to changes in the drivers, since we have used the standard drivers that came with the Windows installation.

Table 2. FTP file transfer time and speed values (Time in seconds; Speed in Kbytes/second)

Driver ID	FTP Transfer				
	Intercept off (average)		Intercept on (average)		Time Overhead
	Time	Speed	Time	Speed	
Eth_Vista	0,198	6238	0,202	6204	2%
Eth_Win8	0,136	6503	0,146	5963	7%
WiFi_Vista	9,300	97	10,650	84	15%
WiFi_Win8	0,276	3076	0,314	2872	14%
Bth_Vista	5,890	145	6,012	142	2%
Bth_Win8	5,612	152	5,760	148	3%

During the experiments we saw that for each transmitted byte, Intercept generated between 9 to 23Kbytes of data. Not surprisingly the Wi-Fi driver was the one that generated a higher amount of data, which can be interpreted as a synonymous of increased complexity.

Table 3. Top 5 most used functions by each driver

Function	Eth Vista	WiFi Vista	Bth Vista
NdisMSynchronizeWithInterruptEx	-	69301	-
InterruptHandler	880	33931	-
MiniportInterruptDpc	-	32774	-
NdisAcquireReadWriteLock	-	6345	-
NdisReleaseReadWriteLock	-	6345	-
NdisMIndicateReceiveNetBufferLists	-	-	1032
NdisAllocateMdl	1096	-	-
NdisFreeMdl	1096	-	-
NdisAllocateNetBufferAndNetBufferList	1024	-	-
NdisFreeNetBufferList	1024	-	-
NdisAllocateMemoryWithTagPriority	-	-	520
NdisFreeMemory	-	-	520
MPSendNetBufferLists	-	-	503
NdisMSendNetBufferListsComplete	-	-	503

5.3 Understanding the Dynamics of Function Calls

The dynamics of function calls during a driver's execution is determined by its work load. Intercept can support various kinds of profiling analysis about the usage of functions by a certain device driver under a specific load. For example, in our FTP transfer scenario, Table 3 represents the top 5 most called functions by each DUT from installation and until deactivation (in Windows Vista). Based on the number of function calls it becomes clear that the Wi-Fi driver is the one that shows more activity in the system. Focusing on the top 3 functions from this driver, the `NdisMSynchronizeWithInterruptEx` is the most used function. Drivers must call this function whenever two threads share resources that can be accessed at the same time. On a uniprocessor computer, if one driver function is accessing a shared resource and is interrupted, to allow the execution of another function that runs at a higher priority, the shared resource must be protected to prevent race conditions. On an SMP computer, two threads could be running simultaneously on different processors and attempting to modify the same data. Such accesses must be synchronized.

`InterruptHandler` is the second most executed function. This function runs whenever the hardware interrupts the system execution to notify that attention is required. From the 33931 interrupts, 32774 calls were deferred for later execution with `MiniportInterruptDpc`. By inspecting the remaining functions used by the Wi-Fi driver, which are lock related, it becomes evident that the driver is relying heavily on multithreading and synchronization operations.

Several other metrics can be obtained with Intercept, such as the minimum, average and maximum usage of each individual resource, DMA transfers, restarts, pauses, most used sections of the code, to name only a few. Intercept can also be employed when particular information needs to be collected. As an example, we wanted to find out what data is returned by the FTP server after the client connects. Fig. 2 shows a call performed by the DUT to the OS notifying NDIS that a new frame has just arrived. In this case it is possible to observe the banner received from the FTP server, i.e., "220-Welcome to Cerberus FTP Server".

```
ENTER - NdisMIndicateReceiveNetBufferLists
MiniportAdapterHandle..:0x895de190
NetBufferLists.........:0x87f86898
PortNumber.............:0x00000000
NumberOfNetBufferLists.:0x00000001
ReceiveFlags...........:0x00000002
NDIS_RECEIVE_FLAGS_RESOURCES
NetBuf.................: 0x87f86898
Dumping mdl.....: 0x882412b0
0x884a6a10  ff ff ff ff ff ff 00 27 13 4f 61 40 08 00 45 00 .......'.Oa@..E.
0x884a6a20  00 4e 01 87 00 00 80 11 24 ba c0 a8 c9 0d c0 a8 .N......$.......
0x884a6a30  c9 ff 00 89 00 89 00 3a f1 ff 91 92 01 10 00 01 .......:........
0x884a6a40  00 00 00 00 00 00 20 46 44 45 4e 46 44 46 50 46 ...... FDENFDFPF
0x884a6a50  44 45 4d 46 41 43 41 43 41 43 41 43 41 43 41 43 DEMFACACACACACAC
0x884a6a60  41 43 41 43 41 41 41 00 00 20 00 01 6f 6e 0d 0a ACACAAA.. ..on..
0x884a6a70  32 32 30 2d 55 4e 52 45 47 49 53 54 45 52 45 44 220-UNREGISTERED
0x884a6a80  0d 0a 32 32 30 2d 57 65 6c 63 6f 6d 65 20 74 6f ..220-Welcome to
0x884a6a90  20 43 65 72 62 65 72 75 73 20 46 54 50 20 53 65  Cerberus FTP Se
0x884a6aa0  72 76 65 72 0d 0a 32 32 30 20 43 72 65 61 74 65 rver..220 Create
0x884a6ab0  64 20 62 79 20 47 72 61 6e 74 20 41 76 65 72 65 d by Grant Avere
0x884a6ac0  74 74 0d 0a 00 56 00 03 00 01 00 00 00 02 00 32 tt...V.........2
0x884a6ad0  00 5c 4d 41 49 4c 53 4c 4f 54 5c 42 52 4f 57 53 .\MAILSLOT\BROWS
```

Fig. 2. Looking in detail at a particular packet (excerpt)

5.4 Understanding How Drivers Interact with the Hardware

Intercept can also help to understand how specific hardware interactions are performed. The NDIS Library provides a set of I/O functions that a miniport driver calls to access I/O ports. These calls provide a standard portable interface that supports the various operating environments for NDIS drivers. For instance, functions are offered for mapping ports, for claiming I/O resources, and for reading from and writing to the mapped and unmapped I/O ports. Taking the Wi-Fi driver as an example, one can use Intercept to learn how the hardware initialization process happens. It starts when the OS invokes the drivers' callback function `MPInitializeEx` (see Fig. 3).

The OS passes several parameters to this function. One of them is the `MiniportAdapterHandle` so that whenever there is the need for the driver to call for some function, the OS is able to know which hardware the driver is referencing to (in this case, the reference is 0x8a16b0e8). All subsequent functions related with this driver will use this reference.

```
ENTER - NewMPInitializeEx
MiniportAdapterHandle.....: 0x8a16b0e8
MiniportDriverContext.....: 0x00000000
MiniportInitParameters....: 0x8c35b6d8
Header.Revision.....: 0x00000001
Header.Size.....: 0x00000028
Header.Type.....: 0x00000081
Flags.....: 0x00000000
IMDeviceInstanceContext.....: 0x00000000
MiniportAddDeviceContext.....: 0x00000000
IfIndex.....: 0x00000041
NetLuid.....: 0x00000000
NetLuid.Info.....: 0x00000000
NetLuid.Value.....: 0x00000000
AllocatedResources.....: 0x8a1e269c
AllocatedResources->Version.....: 0x00000001
AllocatedResources->Revision.....: 0x00000001
AllocatedResources->Count.....: 0x00000003
AllocatedResources->PartialDescriptors[00000000].Type.....: 0x00000003
AllocatedResources->PartialDescriptors[00000000].ShareDisposition.....: 0x00000001
AllocatedResources->PartialDescriptors[00000000].Flags.....: 0x00000080
CmResourceTypeMemory
AllocatedResources->PartialDescriptors[00000000].u.Memory.Start.....: 0xd4000000
AllocatedResources->PartialDescriptors[00000000].u.Memory.Length.....: 0x00010000
```

Fig. 3. Call to MPInitializeEx to initialize the hardware (excerpt)

Another parameter is the resources allocated for the hardware. This allocation was performed automatically by the system according to the PCI standard, which releases the programmers from doing it. However, the driver only gets to know it when this function is called. In this example some of resources assigned to the Wi-Fi hardware were: Memory start: 0xd4000000 and Memory length: 0x00010000.

5.5 Understanding Particular (Complex) Interactions with the OS

Intercept can be used to comprehend how certain complex operations are performed by the driver. For example, in Windows, a driver can remain installed but disabled. By analyzing the log produced by Intercept during the disabling process, it is possible to observe that the OS first calls the drivers' `MiniportPause` to stop the flow of data through the device. Second, the OS calls `MiniportHalt` to obtain the resources that were being utilized. Both these two functions were registered during the initialization process, at the time using the `NdisMRegisterMiniportDriver` function. Finally, the OS calls the `Unload` function to notify the driver that is about to be unload. The `Unload` function was also registered by the driver in the OS when the `DriverEntry` routine returned, by setting the address of this function in the `DriverUnload` field of the `Driver_Object` structure. As soon as the `Unload` function starts it is possible to observe in the log that the driver calls the `MPDriverUnload` callback function. When this function ends the unload process ends and the driver is disabled.

Another example corresponds to uninstalling the driver. With the information logged by Intercept, it was found that there is no difference between disabling and uninstalling a driver, except from the fact that uninstalling the driver removes it from the system.

The detailed information stored by Intercept in the log also helps to determine if all resources allocated by the driver are returned to the OS core. This can assist for instance to detect drivers with bugs. Table 4 represents the use of five resources utilized by the Wi-Fi driver. It is possible to observe a match between the number of resource allocations and releases, which gives evidence that the driver released all those allocated resources.

Table 4. Top 5 allocation/release resources functions

Allocation function	#Calls	Release function	#Calls
_IWDD_NdisFreeIoWorkItem	1158	_IWDD_NdisAllocateIoWorkItem	1158
_IWDD_NdisMAllocateNetBufferSGList	1041	_IWDD_NdisMFreeNetBufferSGList	1041
_IWDD_NdisMAllocateSharedMemory	803	_IWDD_NdisMFreeSharedMemory	803
_IWDD_NdisAllocateNetBuffer	256	_IWDD_NdisFreeNetBuffer	256
_IWDD_NdisAllocateNetBufferList	256	_IWDD_NdisFreeNetBufferList	256

6 Using Intercept as a Component of a Testing Tool

Currently, we are developing a testing tool that uses Intercept as a building block. Due to its detailed logs, the tester can fully understand the driver's dynamics, and thus

plan and design tests that target specific and elaborate conditions. The new tool uses a file that describes the test pattern. Whenever a function is intercepted by the Interception Layer, it calls the `_Inject_decison` function to evaluate the conditions and execute the test accordingly.

As a demonstration of the results of this ongoing work, an experiment was performed during the initialization of the Wi-Fi driver in Windows 8. The test targeted the `NdisMMapIoSpace` function that maps a given bus-relative "physical" range of device RAM. When successful, this function returns `NDIS_STATUS_SUCCESS` and the value of the output parameter `VirtualAddress` contains the start of the memory map. Other outcomes are exceptions that should be handled quietly.

Four test scenarios were planned by returning to the driver three possible exceptional values (as described in the Microsoft documentation) `NDIS_STATUS_RESOURCE_CONFLICT`, `NDIS_STATUS_RESOURCES`, `NDIS_STATUS_FAILURE` and one unspecified value `(NDIS_STATUS_FAILURE+1)`, while maintaining `VirtualAddress` equal to `NULL`. The DUT handled correctly the tests and ended quietly, and appropriately deallocated all resources, as confirmed by the Intercept logs.

Four additional test scenarios were performed with the same return values but assigning a specific value to `VirtualAddress`. These tests all resulted in a crash with the DUT being the culprit. It was concluded that the driver is using the value of `VirtualAddress` before checking the return value, which is worrisome in case Windows does not clear the `VirtualAddressis` field.

7 Conclusions

The paper presents Intercept, a tool that instruments WDD by logging the driver interactions with the OS at function level. It uses an approach where the WDD binary is in full control and the execution traced to a file recording all function calls, parameter and return values. The trace is directly generated in clear text with all the involved data structures.

An experiment with three network drivers was used to demonstrate some of the instrumentation capabilities of Intercept. The performance of the tool was also evaluated in a FTP file transfer scenario, and the observed overheads were small given the amount of information that is logged, all below 15%.

As is, Intercept gives a clear picture of the dynamics of the driver, which can help in debugging and reverse engineering processes with low performance degradation. Intercept is also currently being used as a building block of a testing tool. Preliminary results show the ability to identify bugs in drivers, by executing tests based on the knowledge obtained from the driver's dynamics.

References

1. Hunt, G., Brubacher, D.: Detours: Binary Interception of Win32 Functions. In: Proc. of the Conf. of USENIX Windows NT Symposium (1999)
2. Skaletsky, A., Devor, T., Chachmon, N., Cohn, R., Hazelwood, K., Vladimirov, V., Bach, M.: Dynamic Program Analysis of Microsoft Windows Applications. In: Proc. of the Int. Symp. on Performance Analysis of Systems & Software (2010)

3. Chou, A., Yang, J., Chelf, B., Hallem, S., Engler, D.: An Empirical Study of Operating System Errors. In: Proc. of the Symp. on Operating Systems Principles (October 2001)
4. Mendonça, M., Neves, N.: Robustness testing of the Windows DDK. In: Proc. of the Int. Conf. on Dependable Systems and Networks (June 2007)
5. Albinet, A., Arlat, J., Fabre, J.-C.: Characterization of the Impact of Faulty Drivers on the Robustness of the Linux Kernel. In: Proc. of the Int. Conf. on Dependable Systems and Networks (June 2004)
6. Durães, J., Madeira, H.: Characterization of operating systems behavior in the presence of faulty drivers through software fault emulation. In: Proc. of the Pacific Rim Int. Symp. of Dependable Computing (December 2002)
7. Johansson, A., Suri, N.: Error Propagation Profiling of Operating Systems. In: Proc. of the Int. Conf. on Dependable Systems and Networks (July 2005)
8. Microsoft, Microsoft Portable Executable and Common Object File Format Specification (February 2005)
9. WDK 8.0 (July 2012), http://msdn.microsoft.com/en-US/windows/hardware/hh852362
10. Song, D., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, M.G., Liang, Z., Newsome, J., Poosankam, P., Saxena, P.: BitBlaze: A new approach to computer security via binary analysis. In: Sekar, R., Pujari, A.K. (eds.) ICISS 2008. LNCS, vol. 5352, pp. 1–25. Springer, Heidelberg (2008)
11. Balakrishnan, G., Reps, T., Kidd, N., Lal, A.K., Lim, J., Melski, D., Gruian, R., Yong, S., Chen, C.-H., Teitelbaum, T.: Model checking x86 executables with CodeSurfer/x86 and WPDS++. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 158–163. Springer, Heidelberg (2005)
12. Henri-Gros, C., Kamsky, A., McPeak, S., Engler, D.: A few billion lines of code later: using static analysis to find bugs in the real world. Communications of the ACM 53(2) (2010)
13. Bergan, T., Anderson, O., Devietti, J., Ceze, L., Grossman, D.: CoreDet: a compiler and runtime system for deterministic multi-threaded execution. In: Proc of the Int. Conf. on Architectural Support for Programming Languages and Operating Systems (March 2010)
14. Pasareanu, C., Mehlitz, P., Bushnell, D., Gundy-Burlet, K., Lowry, M., Person, S., Pape, M.: Combining unit-level symbolic execution and system-level concrete execution for testing NASA software. In: Proc. of the Int. Symp. on Software Testing and Analysis (July 2008)
15. The LLVM Compiler Infrastructure (February 2013), http://llvm.org/
16. Chipounov, V., Candea, G.: Enabling Sophisticated Analysis of x86 Binaries with RevGen. In: Proc. of the Int. Conf. on Dependable Systems and Networks (June 2011)
17. Libpcap file format (February 2013), http://wiki.wireshark.org
18. WireShark (February 2013), http://www.wireshark.org/
19. Sweex (February 2013), http://www.sweexdirect.co.uk
20. WinDbg (February 2013), http://msdn.microsoft.com/en-us/windows/hardware/gg463009.aspx
21. SystemTap (February 2013), http://wiki.eclipse.org/Linux_Tools_Project/Systemtap
22. Ftrace (February 2013), http://lwn.net/Articles/322666/
23. Passing, J., Schmitdt, A., Lowis, M., Polze, A.: NTrace: Function Boundary Tracing for Windows on IA-32. In: Proc. of the Working Conf. on Reverse Engineering (October 2009)
24. Bruening, D., Garnett, T., Amarasinghe, S.: An Infrastructure for Adaptive Dynamic Optimization. In: Proc. of the Int. Symp. on Code Generation and Optimization (March 2003)

The Challenge of Detection and Diagnosis of Fugacious Hardware Faults in VLSI Designs

Jaime Espinosa, David de Andrés, Juan-Carlos Ruiz, and Pedro Gil

Fault-Tolerant Systems Group (GSTF),
Instituto de Aplicaciones de las TIC Avanzadas (ITACA)
Universitat Politècnica de València (UPV),
Campus de Vera s/n, 46022, Valencia, Spain
{jaiesgar,ddandres,jcruizg,pgil}@disca.upv.es
http://www.stf.webs.upv.es

Abstract. Current integration scales are increasing the number and types of faults that embedded systems must face. Traditional approaches focus on dealing with those transient and permanent faults that impact the state or output of systems, whereas little research has targeted those faults being logically, electrically or temporally masked -which we have named fugacious. A fast detection and precise diagnosis of faults occurrence, even if the provided service is unaffected, could be of invaluable help to determine, for instance, that systems are currently under the influence of environmental disturbances like radiation, suffering from wear-out, or being affected by an intermittent fault. Upon detection, systems may react to adapt the deployed fault tolerance mechanisms to the diagnosed problem. This paper explores these ideas evaluating challenges and requirements involved, and provides an outline of potential techniques to be applied.

Keywords: Fault detection, transient faults, intermittent faults, permanent faults, fault diagnosis, VLSI design workflow.

1 Introduction

Current embedded VLSI systems are widespread and operate in multitude of applications in different markets, ranging from life support, industrial control, or airborne electronics to consumer goods. It is unquestionable that the former require different degrees of fault tolerance, given the human lives or great investments at stake, but it is not so obvious to admit that unexpected failures in consumer products can undermine their success in the marketplace [1]. Hence, there is great interest in protecting equipment from eventual faults, which in turn involves providing a certain degree of service reliability over the whole lifetime. Specifically this relies on controlled operation of both software and hardware. While it is clear that potential programming bugs will affect the software behaviour, recent studies on complete systems highlight the disastrous impact which even transient faults happening in the hardware may have in the code

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 76–87, 2013.

execution of critical applications [2]. Therefore in order to achieve dependable devices it is no longer possible to preclude hardware implications from software design.

In the design stage of a product it is foreseeable an evolution in its operational state, from an ideal scenario to another posing several dependability threats. Since a set of specifications has to be met, a conservative approach is taken and security margins are applied to compensate for expected negative effects which may hinder correct service delivery. But that evolution can be no longer predicted accurately enough [3], leaving the only alternative to adapt the system to unexpected changes during its service lifetime. Hence, it is a growingly important requisite to create an information flow from environment to hardware and finally software. A major source of such sudden changes in a system is the occurrence of faults.

To explain why faults appear, there are a number of reasons to be mentioned. For instance, manufacturing capabilities have been evolving at a fast pace, bringing a new breadth of improvements to embedded systems in terms of logic density, processing speed and power consumption. However, those benefits become threats to the dependability of systems, causing higher temperatures, shorter timing budgets and lower noise margins which increase fault proneness. In addition, deep-submicron technologies have both decreased the probability of manufacturing defect-free devices, and increased the likelihood of problematic events originated by wear-out. Moreover, the susceptibility of extremely integrated electronics to α-particles and neutrons, arriving from outer space or radioactive materials grows steadily, yielding a non-negligible degree of so called *soft errors* [4], which affect temporarily the correctness of processing.

The mentioned faults can be classified in permanent or transient categories. Research in the field has focused mainly in tackling permanent faults, disregarding transient faults when their effect is not visible as errors in the captured data. For instance, transient faults with short activation times (percentage of time in which it is affecting the system relative to clock period), which have been shown difficult to detect by conventional means [5], may not produce incorrect outputs at once, but are a good indication of a problematic environment. We have named them *fugacious*. According to [6], out-of-range supply voltages, abnormal noise, temperature, etc. are triggers for such transient faults, which if repeated are called *intermittent* faults. Whether the final nature of the fault is transient or intermittent will depend on the wear-out conditions. For that reason we must make an effort to be able to detect and diagnose such types of faults, because these will provide valuable information when taking decisions for the evolution of the system. An example would be to change the data codification in a bus to a more robust scheme in the hardware, or to enable additional processing iterations or variable checks in the software, for redundancy purposes. Studies devoted to detection and diagnosis of fugacious faults are scarce or non-existent. However, certain known detection techniques could be applied to fugacious faults with limited success [24], since only a reduced period of time is monitored.

The contribution of this paper is based in 2 major points: (i) to identify and ponder the challenges of detection and diagnosis of fugacious faults in VLSI systems and (ii) to provide insight on methods and technologies to cover such challenges.

The rest of the paper is structured as follows. Section 2 justifies the importance of on-line detection, and underlines the difficulties of detecting transient and intermittent faults with short activation times. Furthermore it presents the different fault models while providing an overview on diagnosis of such faults. A set of methods and technologies is presented in Section 3 to cope with the task. Finally, Section 4 indicates the following actions to be taken and related issues.

2 The Problem of Fast Fault Detection and Diagnosis

According to Avizienis [7] the basic criterion to catalogue faults in permanent or transient type is the persistence. This can however be an incomplete information to comprise the whole picture and thus, *activation reproducibility* is the concept introduced to better describe the observed situations. For permanent faults, different activation patterns lead to solid, hard faults when these are systematically reproducible or to elusive, soft faults when they are not. Depending on circumstances those soft faults can be intermittent in time. For transient faults, elusive activation is the most common but certain circumstances can likewise make them manifest intermittently.

Such differentiated activation patterns require tailored fault tolerant techniques of detection and diagnosis for dependability threats caused by faults and errors. In several situations including high availability or high performance systems, a concurrent detection (on-line) becomes critical. Next the existing scenario related to such concepts is explained.

2.1 On-Line Detection of Faults and Errors

In order to test proper development of the systems several methods have been described. From post-manufacture checking by means of test vectors or burn-in testing used to discard flawed units, to assigning slots of regular service time for test, for instance, many off-line techniques are currently employed. But the advantage of on-line detection is clear. A loose detection or notification latency, can have disastrous consequences in certain situations [8]. Besides, the longer a fault is present in the system without detection the higher the probability of facing a multiple fault situation. Provided that the latter is a problem of increased complexity we find justified interest in early detection.

There is long tradition in the dependability community to develop on-line error detectors. Typically, they are based in the use of special data codification or in the replication and comparison of outputs or state variables. But the relationship between a fault occurred at the processing network and an error manifested in the outputs or state variables is a limiting factor known as *observability* [9]. When the observability in an output is null for a given fault, no matter which

input data combination was applied the fault will not show at the output. Likewise if more than one output is observable for that fault, multiple alterations would be detectable at those outputs. This can be specially important when using encoded circuits, where several properties describe different types of such circuits according to the consequences of a set of faults [10].

- *Fault-Secure.* Circuits where in presence of a fault either the outputs are correct or they are not a valid code word.
- *Self-Checking.* If for every fault of the fault models and for every input either the output is correct or it is detected as error.
- *Self-Testing.* Circuits in which for every fault of the fault set there is at least one possible input which causes the error to be detected.
- *Totally Self-Checking.* Circuits that are simultaneously Fault-Secure and Self-Testing.
- *Code-disjoint.* Circuits where a non-code word input generates a non-code output. This allows detection of erroneous inputs or cascading of blocks.

Therefore when employing codification it will be desirable to maximise the observability in order to achieve a good fault coverage.

There are 3 possible causes for fault filtering: electrical, logical or temporal. When dealing with permanent faults, temporal causes are discarded and due to the nature of hard faults, logical filtering can only be short in time. For those reasons any checking methodology will obtain positive results with hardly any misses meanwhile the observability is good enough.

Nevertheless, detection of transient or elusive and intermittent faults is not so straightforward. On the one hand, and according to field data from digital systems [11], transient faults have been shown to account for up to 80% of failures. These can be caused by several reasons as it is known. Among those reasons, the arrival of α-particles, protons or neutrons from radiation is one of the most studied and popular. If we pay attention to the evolution of transient fault duration produced by one of these particles impacting a CMOS node, the result is directly proportional to the feature size of the electronics [12]. However, the operational frequency of devices has not been following the historic monotonic growing trend, due to well known power dissipation issues. Consequently transients produced by radiated particles and charge build-ups are narrower and narrower compared to the clock periods. This paves the way to believe that although the number of faults affecting a system may be high, chances are these would not be easily captured by clock edges at the storage elements (heavy temporal filtering, see Figure 1) . The derivatives of this are that the moment a fault is detected many more could have already happened and the available time for reaction could be too short. Therefore for self-awareness purposes it is desirable to detect them.

On the other hand, intermittent errors caused as studied by Nightingale [13] a total of 39% of all hardware errors which, according to reports by Microsoft from 950.000 computers, induced a crash in the operating system. This gives a hint on the number of intermittent faults that can be happening in the system if we consider that not all of them will end in an operating system crash. Other

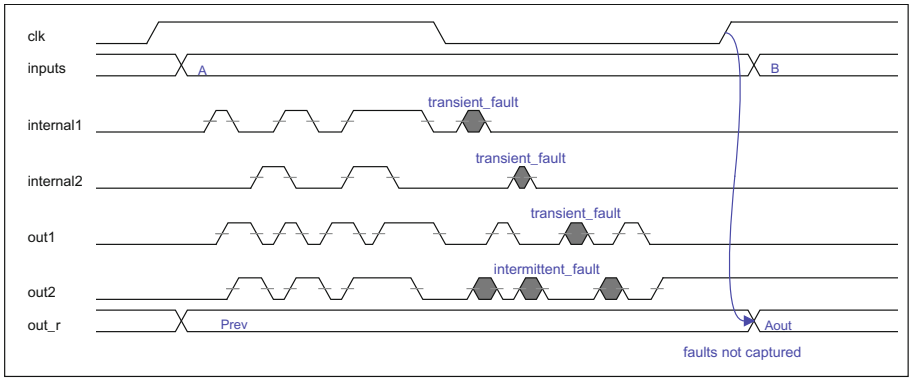


Fig. 1. Temporal filtering

examples of can be found in certain cruise control modules for vehicles [14] which hit a return rate of 96% due to undetected intermittent failures. These figures can be uneven depending on the context of operation since, as distinguished by Savir [15], random originated intermittent faults appear and disappear in an unpredictable fashion whether systematic intermittent faults evolution can be numerically characterized [16]. This enables a proper decision on the best moment to apply recovery actions to maximise availability. Such systematic intermittent faults start by small fluctuations which grow in time and intensity until their effect is severe [17]. In order to set focus on the considered problem, a description of fault models is required.

2.2 Considered Fault Models

According to the presented concept of activation time, and the activation reproducibility described earlier, we have established the name of *fugacious* faults to refer to a set of 3 different types of faults. The fugacious transient faults are defined as those which remain active less than a clock cycle of the system. Likewise, fugacious intermittent faults are those transient faults which activate at least twice in a clock period. Finally, permanent faults are active the whole time span of the clock period, that means for us a fault lasting more than one clock period will be considered permanently active.

2.3 Fault Diagnosis

Multiple efforts have been conducted towards an effective diagnosis of different types of faults based on their activation reproducibility. As demonstrated in [18] there are important benefits for the Mission Time Degradation and Mean Time To Failure (MTTF) Degradation associated to correctly discriminating transient from permanent faults. It is clear that no equal treatment has to be given to both of them. For instance, transient faults will require no corrective action at

all when hardware redundancy provides a voted fault tolerance. Disregarding the affected element for a certain period of time will negatively affect the dependability of the system. Furthermore, given the nature of intermittent faults and their proneness to become permanent, a proper distinction provides insight on the convenience to isolate or recover the functional unit. Intermittent faults diagnosis is a hot-topic in the field. An analytical model for a fault controller was presented in [19], using a thresholds-based α count methodology to discriminate transient from intermittent faults. Its Stochastic Activity Networks (SAN) analysis is specifically based on the time step, where transient faults last for less than one step and intermittent faults repeat their appearance in subsequent steps. Its drawbacks are it requires a long latency to discriminate, and infrastructure to detect and accumulate the respective faults. Other recent studies which also employ SAN with thresholds [20] applied to real systems only consider intermittent errors captured in state variables, which last more than one clock cycle.

In the case of fugacious faults, we take into account events of a quickly 'evanescent' nature where the capture and diagnosis procedure must have intrinsically low latency. It must be able to process two or more faults per cycle in order to discriminate an intermittent activation from a transient activation, avoiding a new constraint in the frequency requirements.

3 Solutions for Detection and Diagnosis

Our effort has been focused in two directions: determining an appropriate structure to detect and diagnose the set of faults we have previously presented and defining a procedure to apply such structure to the standard design flow.

3.1 Architecture of a Faults Detection and Discrimination System

In every VLSI circuit we can find combinational stages separated by registers. Since the pursued goal is to have accurate and flawless computations, we will require 2 conditions:

- *To produce correct results.*
- *To sense any deviations in the datapath which may be out of reach by just checking registered values.*

The steps to take in order to reach these goals start by considering hardware replication and comparison. The large number of commercial systems utilizing such technique tells about the effectiveness of the approach at the expense of important amounts of hardware. The foremost advantage is quick on-line mitigation (when a voter is included), and usually there is no need to include voters in every stage but just in critical ones. Nevertheless, for detection and diagnosis a lighter, cheaper technique would enable the possibility to deploy detection to a larger number of partitions spread around the system. The use of codification may well fill the gap and combine with replication in a wise manner.

An interesting feature of codification is that systematic codes do not require to alter the original bits, thus alleviating the decoding of outputs in the datapath. In order to minimise speed penalty applied to outputs, this makes a great advantage [21]. Berger codes and parity groups are the most popular systematic codes and have been profoundly studied. In [22] conditions for fault secureness in parity predictors are derived. Furthermore [23] presents a generic optimisation technique for parity prediction functions, to achieve quick and small circuits.

The envisioned topology using codification would follow that in Figure 2. In it, a *Detection Block* would receive inputs directly from partition input registers, and also from outputs prior to registering. A properly selected codification could reduce block area and optimise the speed. This block would include thus a set of Commercial Off-The-Shelf (COTS) encoder and decoder which can be a single bit parity prediction/decoding pair in its simplest form.

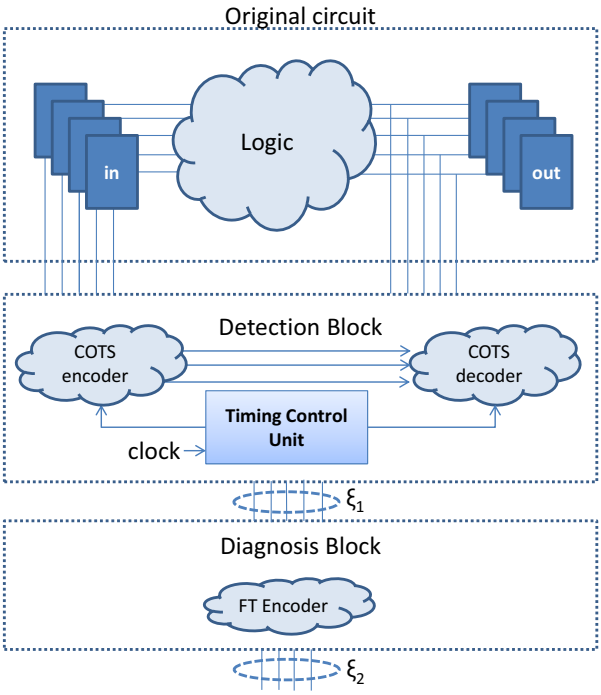


Fig. 2. Global scheme of the faults detection and diagnosis infrastructure. Timing Control Unit handles temporisation of Detection decoder.

Is that enough to handle every type of fault? The answer is 'no'. Coding functions are effective techniques to detect permanent errors, or transient errors which are not time filtered. For effective detection of transient faults of a limited activation time (smaller than a clock period in general), additional elements are required. An example using triplication was presented in [24], where intermittent

faults were not considered at all, and the sensing time was rather reduced. On-line detection of intermittent faults has been previously devised in different ways. One proposed idea was to inject a carrier signal in the line under study and monitor the correct behaviour of it [25]. Again, the cost is rather high: an injector and receiver for the lines under analysis, plus extra wear-out due to increased switching of the lines. A cheaper detection can be achieved by monitoring those coded lines devoted to detection.

To avoid those shortcomings, an additional element included is the *Timing Control Unit* (TCU). Its function is to adjust the timings of detection elements with one goal in mind, i.e. to increase the *observation window*. The term refers to the percentage of the clock period when the lines under study are monitored for any potential faults. If we reduce the switching interval as opposed to the stability interval of the signals, we will have increased the observation window and thus the effectiveness of the detection (see Figure 3). The reason for preferring this method to the observation of a reduced period of time assuming equiprobable distribution of faults is clear, i. e. to gain in speed of detection.

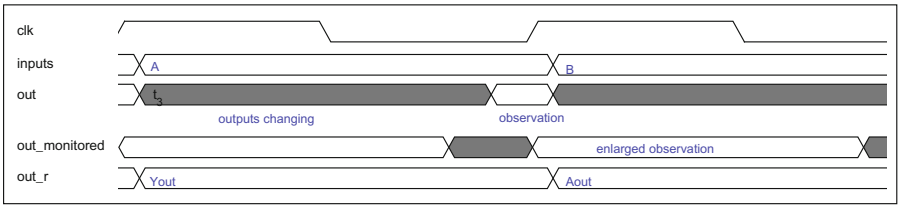


Fig. 3. Observation window enlarged by means of reducing period of signal switching

Finally once gathered, the detection information could be codified against faults using a code (ξ_1) and passed to a *Diagnosis Block*, where the same or a different code (ξ_2) can be used to notify the diagnosed output to a fault controller.

Inside the *Diagnosis Block*, inputs must be analysed and discriminated to offer 5 different output possibilities:

- *Transient fault.*
- *Permanent fault.*
- *Intermittent fault.*
- *No fault.*
- *Error in diagnosis infrastructure.*

To achieve the goal, the Diagnosis Block will be built using a fault-tolerant (FT) encoder designed to minimise resources taken. By providing all these different outputs and doing so in a fault tolerant codification, the most adequate decision will be enabled to be taken at the fault controller. Hence, smart reactions can be applied well in advance to an eventual collapse of fault tolerance infrastructure.

3.2 Workflow to Apply in the Proposed Technique

In order to automate the process of deploying a detection and diagnosis infrastructure to a generic design block, a suggested procedure is shown in Figure 4. What is depicted is a typical semi-custom design flow for VLSI products, where the standard steps are on the left hand side. *Technology files* can represent a silicon foundry design kit or an FPGA manufacturer primitives library. Likewise, *Physical* element can be a layout file or a programming bitstream for an FPGA. On the right we find detail of 2 interventions in the flow.

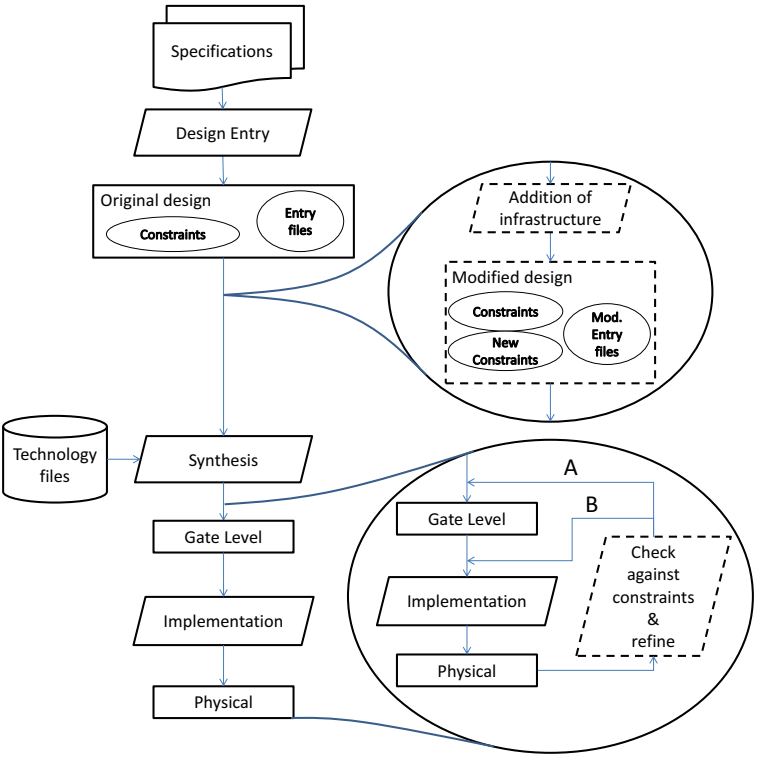


Fig. 4. Tools interaction

A first intervention comes before the *Synthesis* and after *Design Entry*. This step comprises an addition of required infrastructure in the Detection Block, i.e. the COTS components and Timing Control Unit. Entry files are modified as required and new timing constraints are generated for the TCU, to drive the remainder of the design flow.

A second intervention happens in a loop between Gate-level and Physical stages of the design. The purpose is to check timings against new constraints, mainly affecting the TCU, and refine the implementation in a loop by tweaking

in one of the 2 re-entry points A or B. If path B is selected, a faster process will be obtained as a result, but deep knowledge of the underlying technology will be required and we will find a side effect of loss of portability. With path A, a more general solution will be obtained at the cost of speed of implementation.

The challenge in the integration of processes is derived from the difficulty to achieve the optimum observation window for the whole range of process variability. Other difficulties can come from the capabilities and restricted information offered by technology suppliers.

4 Ongoing Work

An initial implementation is currently under development, where an FPGA-based design flow has been chosen to support initial testing. Following the presented ideas, we have been able to develop first modification point working models. To reach optimal performance, we need first is to maximise the detection capabilities of the structure, both in area and time. This means achieving a high degree of observability at the check lines.

As for the second modification our efforts are devoted to achieve low performance penalty results and at the same time maximising the period in which lines are under surveillance. We need the least possibly intrusive system in order not to give in too much in exchange for detection. This is vital when applied to extreme performance demanding systems.

Last but not least, keeping the additional area small can be complex in certain circuits, if a powerful logic optimisation is not wisely applied. The upper limit will be that imposed by pure replication but this should be perfectly reducible without loosing much of the observability. An associated parameter to area increase is the power drain due to new infrastructure. As usual in engineering, specifications and market constraints drive the balance between detection and diagnosis capability and power/area/performance penalty.

Acknowledgements. This work has been funded by Spanish Ministry of Economy ARENES project (TIN2012-38308-C02-01).

References

1. Narayanan, V., Xie, Y.: Reliability concerns in embedded systems design. IEEE Computer 1(39), 118–120 (2006)
2. Hannius, O., Karlsson, J.: Impact of soft errors in a jet engine controller. In: Ortmeier, F., Daniel, P. (eds.) SAFECOMP 2012. LNCS, vol. 7612, pp. 223–234. Springer, Heidelberg (2012)
3. Borkar, S.: Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. IEEE Micro 25(6), 10–16 (2005)
4. JEDEC: Measurement and reporting of alpha particle and terrestrial cosmic ray-induced soft errors in semiconductor devices. JEDEC Standard JESD89A. JEDEC (2006)

5. Gracia-Moran, J., Gil-Tomas, D., Saiz-Adalid, L.J., Baraza, J.C., Gil-Vicente, P.J.: Experimental validation of a fault tolerant microcomputer system against intermittent faults. In: DSN, pp. 413–418 (2010)
6. Constantinescu, C.: Intermittent faults and effects on reliability of integrated circuits. In: Proceedings of the 2008 Annual Reliability and Maintainability Symposium, pp. 370–374. IEEE Computer Society, Washington, DC (2008)
7. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secur. Comput. 1, 11–33 (2004)
8. Johnson, C., Holloway, C.: The dangers of failure masking in fault-tolerant software: Aspects of a recent in-flight upset event. In: 2007 2nd Institution of Engineering and Technology International Conference on System Safety, pp. 60–65 (October 2007)
9. Bolchini, C., Salice, F., Sciuto, D.: Fault analysis for networks with concurrent error detection. IEEE Des. Test 15(4), 66–74 (1998)
10. Goessel, M., Ocheretny, V., Sogomonyan, E., Marienfeld, D.: New Methods of Concurrent Checking (Frontiers in Electronic Testing), 1st edn. Springer Publishing Company, Incorporated (2008)
11. Iyer, R.K., Rossetti, D.J.: A statistical load dependency model for cpu errors at slac. In: Twenty-Fifth International Symposium on Fault-Tolerant Computing, 'Highlights from Twenty-Five Years', p. 373 (June 1995)
12. Dodd, P.E., Shaneyfelt, M.R., Felix, J.A., Schwank, J.R.: Production and propagation of single-event transients in high-speed digital logic ics. IEEE Transactions on Nuclear Science 51, 3278–3284 (2004)
13. Nightingale, E.B., Douceur, J.R., Orgovan, V.: Cycles, cells and platters: an empirical analysisof hardware failures on a million consumer pcs. In: Proceedings of the Sixth Conference on Computer Systems, EuroSys 2011, pp. 343–356. ACM, New York (2011)
14. Kimseng, K., Hoit, M., Tiwari, N., Pecht, M.: Physics-of-failure assessment of a cruise control module. Microelectronics Reliability 39(10), 1423–1444 (1999)
15. Savir, J.: Detection of single intermittent faults in sequential circuits. IEEE Trans. Comput. 29(7), 673–678 (1980)
16. Correcher, A., Garcia, E., Morant, F., Quiles, E., Rodriguez, L.: Intermittent failure dynamics characterization. IEEE Transactions on Reliability 61(3), 649–658 (2012)
17. Sorensen, B., Kelly, G., Sajecki, A., Sorensen, P.: An analyzer for detecting intermittent faults in electronic devices. In: AUTOTESTCON 1994. IEEE Systems Readiness Technology Conference. 'Cost Effective Support Into the Next Century', Conference Proceedings, pp. 417–421 (September 1994)
18. Sosnowski, J.: Transient fault tolerance in digital systems. IEEE Micro 14(1), 24–35 (1994)
19. Bondavalli, A., Chiaradonna, S., Di Giandomenico, F., Grandoni, F.: Threshold-based mechanisms to discriminate transient from intermittent faults. IEEE Trans. Comput. 49(3), 230–245 (2000)
20. Rashid, L., Pattabiraman, K., Gopalakrishnan, S.: Intermittent hardware errors and recovery: modelling and evaluation. In: International Conference on Quantitative Evaluation of Systems, QEST (2012)
21. Touba, N.A., McCluskey, E.J.: Logic synthesis of multilevel circuits with concurrent error detection. IEEE Trans. CAD 16(7), 783–789 (1997)

22. Nicolaidis, M., Manich, S., Figueras, J.: Achieving fault secureness in parity prediction arithmetic operators: General conditions and implementations. In: Proceedings of the 1996 European Conference on Design and Test, EDTC 1996, pp. 186–193. IEEE Computer Society, Washington, DC (1996)
23. Ko, S.B., Lo, J.C.: Efficient realization of parity prediction functions in fpgas. J. Electron. Test. 20(5), 489–499 (2004)
24. D'Angelo, S., Sechi, G.R., Metra, C.: Transient and permanent fault diagnosis for fpga-based tmr systems. In: Proceedings of the 14th International Symposium on Defect and Fault-Tolerance in VLSI Systems, DFT 1999, pp. 330–338. IEEE Computer Society, Washington, DC (1999)
25. Kim, C.: Detection and location of intermittent faults by monitoring carrier signal channel behavior of electrical interconnection system. In: Electric Ship Technologies Symposium, ESTS 2009, pp. 449–455. IEEE (April 2009)

GraphSeq Revisited: More Efficient Search for Patterns in Mobility Traces

Pierre André[1,2], Nicolas Rivière[1,2], and Hélène Waeselynck[1,2]

[1] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[2] Univ de Toulouse, UPS, LAAS, F-31400 Toulouse, France
{pierre.andre,nicolas.riviere,helene.waeselynck}@laas.fr

Abstract. GraphSeq is a graph matching tool previously developed in the framework of a scenario-based test approach. It targets mobile computing systems, for which interaction scenarios must consider the evolution of the spatial configuration of nodes. GraphSeq allows the analysis of test traces to identify occurrences of the successive configurations of a scenario. This paper presents a recent improvement made to the tool, to allow for better performance in the average cases. It consists in rearranging the configuration patterns extracted from the scenario, so that the most discriminating nodes are matched first. The improvement is assessed using randomly generated graphs and a test trace from a case study in ad hoc networks.

Keywords: Graph matching, Performance, Testing, Mobile computing systems.

1 Introduction

Mobile computing systems involve devices (handset, PDA, laptop, intelligent car, ...) that move within some physical areas, while being connected to networks by means of wireless links. Compared to "traditional" distributed systems, such systems execute in an extremely dynamic context. The movement of devices yields an unstable topology of connection. Links with other mobile devices or with infrastructure nodes may be established or destroyed depending on the location. Moreover, mobile nodes may dynamically appear and disappear as devices are switched on and off, run out of power or go to standby.

Our work addresses a passive testing approach for such systems. Passive testing (see e.g., [1]) is the process of detecting errors by passively observing the execution trace of a running system. In our case, the properties to be checked are specified using graphical interaction scenarios. Figure 1 gives a schematic view of the approach. The system under test (SUT) is run in a simulated environment, using a synthetic workload. The SUT may involve both fixed nodes and mobile devices. The movement of the latter ones is managed according to some mobility model, a context simulator being in charge of producing location-based data. Execution traces are collected, including both communication messages and location-based data. The traces are then automatically analysed with respect to predefined scenarios, representing test requirements or test purposes.

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 88–95, 2013.

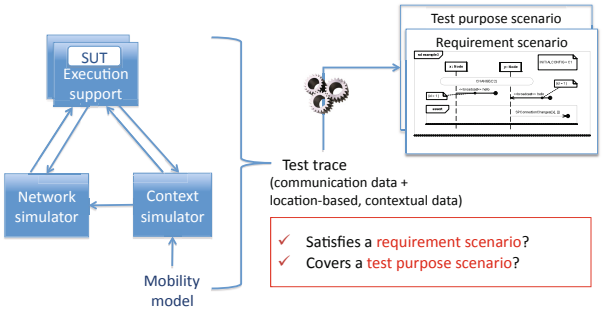


Fig. 1. Overview of the scenario-based approach

Test requirements specify mandatory (positive requirement) or forbidden (negative requirement) interactions. Any observed violation of a requirement must be reported. Test purposes specify interactions of interest, which we would like to observe at least once during testing. If the interaction appears in the trace, the test purpose is reported as covered.

Scenarios are described using a formal UML-based language called TERMOS (Test Requirement Language for Mobile Settings). TERMOS is a specialization of UML Sequence Diagrams [7]. Its genesis can be found in our earlier work [5,6,11]. We first noticed that the spatial configurations of nodes should be a first class concept [5]. As a result, a scenario should have both (i) a spatial view, depicting the dynamically changing topology of nodes as a sequence of graphs, and (ii) an event view representing the communications between nodes. To account for both views, the checking of test traces against scenarios should combine graph matching [6] and event order analysis [11]. In this paper, we focus on the graph matching part, which was implemented by a tool called GraphSeq.

The spatial configurations of a scenario provide a sequence of graphs (the patterns) and GraphSeq search for all occurrences of this sequence of patterns in mobility traces. The addressed graph matching problem is inherently costly, with a worst case complexity exponential in the size and number of the patterns. We present a functionality added to the original version of GraphSeq, in order to improve performance of the average cases. It consists in re-arranging the order of nodes in the patterns, so that GraphSeq tries to match the most discriminating nodes first. The improvement is measured using random graphs and a test trace from a case study in ad hoc networks.

Section II of this paper gives an overview of the TERMOS language. Section III describes the re-arrangement functionality we implemented. Section IV gives performance results. Section V discusses related work. Section VI concludes.

2 Overview of TERMOS

Figure 2 shows an exemplary TERMOS scenario, with its spatial and event views. Note that the shown syntax is not exactly the UML-based one presented in [11]. We adopt here a more compact representation that conveys the same

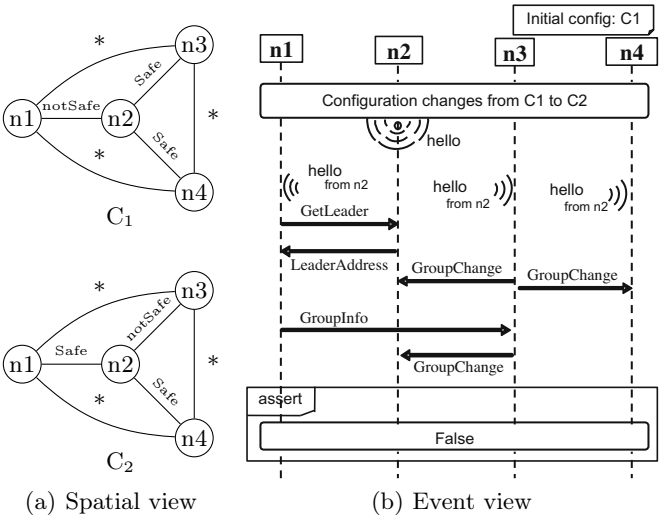
concepts. The scenario is extracted from a Group Membership Protocol (GMP) case study we performed [10]. In this GMP, groups split and merge according to the location of mobile nodes. The protocol uses the concept of safe distance to determine which nodes should form a group. Figure 2 presents a negative requirement (indicated by a false assertion). It describes a pathological interleaving of concurrent split and merge operations that should never occur.

The spatial view contains a set of spatial configurations for the nodes of the scenario. In [11], we depicted them using UML diagrams, but conceptually they consist of labelled graphs. The modeller chooses the labels that are meaningful for the target application. Edge labels characterize the connection of nodes, while node labels (not shown here) are used for contextual attributes of nodes. In Figure 2(a), nodes can have two types of connection, depending on their distance: Safe and NotSafe. Wildcards '*' indicate don't care connection types.

The event view shows the interaction of nodes. Lifelines are drawn for the nodes and the partial orders of their communications are shown. The successive spatial configurations underlying the communications are made explicit: the scenario has an initial configuration and configuration change events are represented (e.g., a change from C1 to C2 in Figure 2(b)).

We interpret TERMOS scenarios as generic patterns, instances of which are searched for in the execution trace. In Figure 2, the nodes ni are symbolic. Any subset of four SUT nodes can match them during execution, by exhibiting the proper spatial configurations and communication events. The search for scenario instances involves two steps:

1. Determine which physical nodes of the trace exhibit the sequence of configurations of the scenario, and when they do so.
2. Analyze the order of events in the identified SUT configurations.



(a) Spatial view (b) Event view

Fig. 2. A TERMOS scenario for groups of mobile nodes

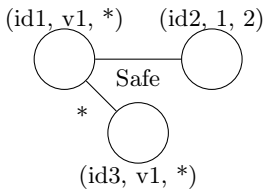


Fig. 3. A pattern graph with various forms of label

Step 1 amounts to a graph matching problem: we search for a sequence of graph patterns (coming from the scenario) to appear in a sequence of SUT configurations (retrieved from the location-based data in the trace). Step 2 requires an interpretation of the event view in terms of partial orders of events. The TERMOS semantics encodes the partial orders in a symbolic automaton to categorize trace fragments as valid, invalid or inconclusive.

In the processing of a scenario, the costliest part is Step 1. For example, we had a GMP execution with 15 nodes moving according to the Reference Point Group Mobility model, during 15 minutes. We checked the logged trace against the scenario of Figure 2. It took about one hour and fifty minutes for the graph matching, while less than five seconds for the event checking in all found configurations. Clearly, an improvement of the graph matching performance is the key for better efficiency.

3 Improvement: Re-arranging Patterns

GraphSeq takes as input two sequences of graphs: (1) a sequence of pattern graphs coming from a scenario description, (2) a sequence of concrete configuration graphs extracted from an execution trace. It compares the two sequences of graphs and returns all matches for the pattern sequence. A match identifies a subset of concrete nodes that exhibit the searched sequence of patterns.

The patterns may involve label variables and wildcards, as illustrated by Figure 3. Nodes have at most three labels. The first one is mandatory and has the form of a variable; it is a symbolic id to be matched by the concrete id of a physical node. The other two labels may be used to represent additional contextual attributes. They may have the form of constant values, variables or wildcards. In Figure 3, the pattern indicates that the concrete nodes playing the role of $id1$ and $id3$ have their first optional attribute at unknown, but identical, values. Each instance of the pattern determines a valuation for the variables. GraphSeq will explore all possibilities, with a sequential reasoning to account for successive patterns. The worst cases are exponential in the size and number of patterns. They occur when every pattern node can be mapped to every concrete node, and the choices made at some point of the sequence of graphs does not restrict the choices for the rest of the sequence.

Fortunately, such worst cases are unlikely to correspond to meaningful scenarios. Rather, the specified patterns should possess some specificities that make them of interest for the application. An idea is then to re-arrange the patterns so that the most discriminating nodes are matched first. To introduce the idea,

let us take the example of the C1 pattern in Figure 2(a). In the encoding supplied to GraphSeq, the first node of the pattern is n1. This node has don't care values for its optional labels (like have other nodes in this example) and also has don't care values for most of its connections to other nodes of the pattern. It is thus not discriminating: many concrete nodes are likely to match n1, and GraphSeq will explore all possibilities. If the pattern is re-arranged so that n2 appears first, GraphSeq will have fewer possibilities to explore, because n2 is more discriminating as regards its connections to other nodes.

We thus introduce a pre-processing step in GraphSeq. Given a pattern, a fitness score rewards the discriminative power of each node appearing in it. Then, the pattern is re-arranged so that the nodes are sorted in descending order of fitness, making the graph matching algorithm consider the fittest nodes first. Algorithm 1 shows the computation of the fitness score of a node. It rewards pre-determined attribute values and a high number of pre-determined connection types with other nodes.

```
Fitness = 0;

// Reward the node if optional labels are discriminating
for each optional label li
    if li is a constant value or a variable that appeared in a previous pattern then
        Fitness += 2;
    endif
endfor

// Reward the node if its connection types with other nodes are discriminating
for each other node nk of the pattern
    if connection to nk is not a don't care then
        Fitness += 1;
    endif
endfor
```

Algorithm 1. Fitness score for a node appearing in pattern P_i

4 Experimental Results

The functionality optimizing the order of nodes in patterns was integrated into GraphSeq, in such a way that it can be activated/deactivated by the user. This allows us to assess its effect on the efficiency of the search for matches. Given two sequences of graphs to be compared, we successively run the tool with and without activation of the functionality to compare the obtained durations.

All experiments were performed on the same platform, a computer with two 2.26GHz quad core and 48GB of ram. The current implementation of GraphSeq is not multi-threaded and uses only one core. Some runs required an amount of memory greater than the available RAM. In such cases, we decided to forbid the use of virtual memory, which would anyway considerably decrease performance. A run is stopped whenever it consumes more than 90% of memory or its duration exceeds three hours.

We first considered the GMP scenario of Figure 2. We ran again the analysis of the GMP mobility trace, using the new version of GraphSeq with the optimization deactivated. The trace involves 15 concrete nodes exhibiting a sequence of 850 concrete configurations. It took GraphSeq 6600.78 seconds to analyze them

and find 59 matches for the scenario. With the optimization activated, it took GraphSeq only 40.63 seconds to find the same matches, more than 160 times faster than previously.

To further assess the improvement, we used randomly generated sequences of graphs. A generation function was available from previous work: when we developed the GraphSeq algorithms, we also developed a test tool to verify the correctness of the matching. The tool generates pairs of pattern and concrete configuration sequences, and by construction there is at least one match for each pair. It can then be verified whether this match is correctly found by GraphSeq. The size and number of graphs can be tuned. The test tool proved very useful to debug GraphSeq and perform regression testing of its successive versions. Here, we use it for evaluation purposes.

Table 1 shows the results of running GraphSeq on randomly generated sequences of graphs. The first column indicates some generation parameters. For example, quadruplet (5, 5, 5..35, 700..2100) means that:

- the generated pattern graphs have 5 nodes;
- the lenght of the pattern sequences is of 5 successive graphs;
- the concrete configuration graphs have a number of nodes in the range of 5..35;
- the lenght of the concrete configuration sequence is in the range of 700...2100 graphs.

We generated 20 pairs of sequences for each experimental quadruplet, yielding 20 runs of GraphSeq without and with optimization. The table gives the number of aborted runs in each case, due to either excessive memory consumption or excessive time duration. It was never the case that a run successfully completes with the optimization deactivated while being aborted with optimization. For the longest pattern sequences (second and fourth row of the table), the optimization proved very effective to allow completion of runs that had to be stopped in the original version of GraphSeq.

Table 1. Runs with random sequences of graphs. Graph generation is characterized by a quadruplet (number of nodes per pattern, number of patterns, range for the number of nodes per concrete configurations, range for the number of concrete configurations). There are 20 runs in each experimental setting.

	♯ Aborted runs		Duration of completed runs in seconds: $\mu(\sigma)[median]$		p-Value
	w/o opt	opt	w/o opt	opt	
(5, 5, 5..35, 700..2100)	Mem: 2 Time: 0	Mem: 0 Time: 0	1110.29 (2335.47) [140.14]	382.55 (1369.86) [18.72]	$< 10^{-5}$
(5, 10, 10..40, 700..2100)	Mem: 7 Time: 3	Mem: 0 Time: 0	511.82 (635,93) [226.50]	213.18 (31.96) [207,76]	0.037
(10, 5, 5..35, 1200..3600)	Mem: 0 Time: 2	Mem: 0 Time: 1	909.22 (1786.19) [43.38]	259.93 (799.54) [39.68]	0.001
(10, 10, 10..40, 1200..3600)	Mem: 6 Time: 8	Mem: 0 Time: 0	281.92(396.61) [95.67]	47.07(6.16) [47.33]	0.031

The duration values of completed runs could be compared. The tables give the mean, median and standard deviation we observed. The high value of σ indicates that, for a given setting of the graph generation, the difficulty of the generated matching problems still largely varies. Moreover, the mean and median could be quite different indicating the values are not normally distributed. We observed lower mean and median when GraphSeq had optimization activated. To determine whether the improvement is statistically significant, we performed a paired difference test. We used the Wilcoxon T test since a normal distribution cannot be assumed. The p-values are reported in the tables.The null hypothesis can be rejected with a 95% confidence level for all experiments.

We conclude that the proposed pattern re-arrangement facility yields a significant improvement of GraphSeq.

5 Related Work

Subgraph isomorphism detection is a problem well studied in the literature [9,4]. In GraphSeq, the core functionality to check whether a pattern appears as a subgraph of a concrete configuration is reused from an existing graph tool developed by colleagues at LAAS [3].

The salient feature of GraphSeq is its algorithm to match *sequences* of graphs: the sequence of symbolic configurations of the scenario, and the sequence of concrete configurations traversed during SUT execution. While the problem of comparing two graphs has been extensively studied, there has been relatively little work on the comparison of sequences of graphs (see [2] for a survey on graph matching). The closest work we found is for the analysis of video images. In [8], the authors search for sequences of patterns (called pictorial queries) into a sequence of graphs extracted from video images. A difference with our work, however, is that a pattern node corresponds to at most one object in an image. In our case, several instances of a pattern may be found in a concrete configuration, with different possible valuations for the variables (including node ids). Hence, to the best of our knowledge, the sequential reasoning implemented by GraphSeq is original.

6 Conclusion

In this paper, we have presented an improvement made to our graph matching tool GraphSeq. Its principle is simple: reward pattern nodes according to their discriminating power, and re-arrange the encoding of the patterns so that the fittest nodes are matched first. While simple, the proposed optimization proved very effective to improve performance. A mobility trace issued from a case study, a group membership protocol in ad hoc networks, could be processed 160 times faster than previously. These promising results were confirmed by experiments on a sample of randomly generated sequences of graphs. The duration values were found significantly higher with than without optimization. Moreover, for the largest configurations we generated, the optimization made it possible to complete a significantly higher number of runs than the original version of GraphSeq. This of course does

not change the theoretical limitation of the tool due to the exponential complexity of the matching problem. But it improves our ability to handle the cases we are targetting in practice, that is, scenarios with small-size patterns that are unlikely to correspond to the worst cases (in the worst cases, no node would be discriminating, hence jeopardizing the node re-arrangement functionality).

GraphSeq is an important component of our scenario-based test platform for mobile computing systems. It addresses the processing of the spatial view of TERMOS scenarios, where the movement of nodes is abstracted by a sequence of labelled graphs. Another tool, integrated into the Papyrus UML environment, uses the outputs of GraphSeq to process the event view showing inter-node communication. The complete processing of a TERMOS scenario is dominated by the duration of the graph matching, which is the costliest part of test trace analysis. By significantly improving the performance of GraphSeq, we thus also significantly improve the overall approach.

References

1. Cavalli, A., Maag, S., de Oca, E.M.: A passive conformance testing approach for a MANET routing protocol. In: Proceedings of the 2009 ACM Symposium on Applied Computing, SAC 2009, pp. 207–211. ACM, New York (2009)
2. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in pattern recognition. IJPRAI 18(3), 265–298 (2004)
3. Guennoun, K., Drira, K.: Using graph grammars for interaction style description: applications for service-oriented architectures. Comput. Syst. Sci. Eng. 21(4) (2006)
4. Messmer, B.T., Bunke, H.: Efficient subgraph isomorphism detection: A decomposition approach. IEEE Trans. Knowl. Data Eng. 12(2), 307–323 (2000)
5. Nguyen, M.D., Waeselynck, H., Rivière, N.: Testing mobile computing applications: toward a scenario language and tools. In: Proceedings of the 2008 International Workshop on Dynamic Analysis: Held in Conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2008), pp. 29–35. ACM (2008)
6. Nguyen, M.D., Waeselynck, H., Rivière, N.: Graphseq: A graph matching tool for the extraction of mobility patterns. In: Proc. Third Int. Software Testing, Verification and Validation (ICST) Conf., pp. 195–204 (2010)
7. Omg. OMG Unified Modeling Language (OMG UML), Superstructure Specification (Version 2.4.1). Technical Report OMG Document Number: formal/2011-08-06, Object Management Group (August 2011)
8. Shearer, K., Venkatesh, S., Bunke, H.: Video sequence matching via decision tree path following. Pattern Recognition Letters 22(5), 479–492 (2001)
9. Ullmann, J.R.: An algorithm for subgraph isomorphism. J. ACM 23(1), 31–42 (1976)
10. Waeselynck, H., Micskei, Z., Nguyen, M.D., Riviere, N.: Mobile systems from a validation perspective: a case study. In: Proc. Sixth Int. Symp. Parallel and Distributed Computing, ISPDC 2007 (2007)
11. Waeselynck, H., Micskei, Z., Rivière, N., Hamvas, Á., Nitu, I.: TERMOS: A formal language for scenarios in mobile computing systems. In: Sénac, P., Ott, M., Seneviratne, A. (eds.) MobiQuitous 2010. LNICST, vol. 73, pp. 285–296. Springer, Heidelberg (2012)

Issues and Ongoing Work on State-Driven Workload Generation for Distributed Systems

Roberto Natella[1] and Fabio Scippacercola[2]

[1] Università degli Studi di Napoli Federico II
roberto.natella@unina.it
[2] Consorzio Interuniversitario Nazionale per l'Informatica
fabio.scippacercola@consorzio-cini.it

Abstract. The dependability of a complex distributed system needs to be assured against the several conditions, namely *states*, in which it can operate. Generating a workload able to cover a desired target state of a distributed system is still a difficult task, since the relationship between the workload and states is nontrivial due to system complexity and non-deterministic factors. This work discusses our ongoing work on a state-driven workload generation approach for distributed systems, based on an evolutionary algorithm, and its preliminary implementation for testing a fault-tolerant distributed system for flight data processing.

Keywords: Distributed Systems, State-based Testing, Fault Tolerance, Fault Injection, Workload, Genetic Algorithms, Off-line synchronization.

1 Introduction

In order to assess and to improve the dependability of a complex distributed system, verification techniques have to inspect those operating conditions, namely *states*, that can expose the system to failures.

The relationship between the application states and dependable behavior was shown in several past studies [1,2,3]. This is particularly important in the case of *fault injection*, as revealed by several studies on the assessment through fault injection of distributed filesystems [4,5], DBMSs [6], and multicast and group membership protocols [7,8,5]. These studies emphasized that the success of recovery is influenced by the state of the distributed system. For instance, if we consider a DBMS that has to guarantee the ACID properties to distributed transactions, its recovery mechanisms (e.g., the rollback to a previous state) can be affected by several factors, such as the presence of several transactions that access to the same resources or that are nested. It is thus evident how a *state-driven workload*, i.e., a workload that brings the system in target states during the analysis, is important to assure the significance and the efficiency of experiments, by *covering the states where the system has to be tested*.

It is well-known that generating a state-driven workload for distributed systems is a difficult and time-consuming task, since the relationship between the workload and states is nontrivial, due to system complexity and non-deterministic factors,

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 96–110, 2013.

such as concurrency and network delays. Past studies proposed the generation of synthetic randomly-generated workloads [9,10], or relied on realistic workloads derived from performance benchmarks [6,11,12]. Nevertheless, these approaches are not meant to cover hard-to-reach. Other approaches generate a workload from stochastic or non-deterministic models of the system, but do not scale well for complex systems [13,14]. Therefore, the automatic generation of state-driven workloads is a relevant but still open issue in distributed systems.

This paper discusses our ongoing work on the automatic generation of state-driven workloads for distributed systems. We propose the use of an agent, which iteratively explores the space of workloads using an evolutionary algorithm. At each iteration, the agent modifies the workload and evaluates its goodness, until the system converges to the target states. To this aim, the approach allows to specify the desired mean probability of reaching the target state for a specified amount of time, thus enabling the injection of faults in the target state. The approach is fully automated and does not rely on a detailed characterization of the relationship between workloads and states, and can therefore be adopted for testing the *actual implementation* of complex distributed systems. We also discuss a preliminary implementation of the approach for testing a fault-tolerant distributed system for flight data processing.

The paper is organized as follows. Section 2 discusses previous studies on state-based and fault injection testing of distributed systems. Section 3 provides basic concepts and assumptions, and Section 4 describes the proposed approach. Section 5 describes how the approach has been implemented in a real-world distributed system, and provides preliminary results. Section 6 concludes the paper and describes future developments of the work.

2 Related Work

While the problem of testing stateful non-distributed systems has been studied in depth [15], the state-based testing of distributed system poses additional and still unsolved challenges, in particular in testing the *actual implementation* of a distributed system. Studies on the verification of distributed systems can be classified into two classes: the analytical-simulation studies and the experimental ones.

Analytical and simulation studies are based exclusively on *analytical* or *behavioral* descriptions of the system, such as Finite State Machines (FSM), Petri Nets (PN), and Computational Tree Logic (CTL). They assess properties or conditions of the system through mathematical proofs, simulations or model checking methods on models [16]. These approaches require abstract models of the system, which have to be hand-written by the tester, or extracted from the system [17]. They are suitable for the verification of the high-level design of the system (e.g., testing protocols or distributed algorithms), but need to be complemented with experimental approaches in order to test low-level design and implementation aspects of the system.

Experimental studies, in which our work is included, exercise and asses the actual implementation of a system, by allowing to analyze a system during its execution. They include, for instance, fault injection methods, which assess fault tolerance mechanisms and algorithms through the deliberate injection of faults in the actual system or in a prototype [7]. In these studies, the problem of the state and of workload generation has been approached in several ways.

In some studies, a model of the system is adopted for automatically generating test cases for the system. For instance, conformance testing approaches generate test cases aimed at covering the states of the model and at assuring that the system evolves as described in the model. These approaches are based on a detailed model of the target system, which describes the expected behavior[18,19,20]. Since in most cases distributed systems are non-deterministic (i.e., the system may evolve in more than one way given the same inputs, due to random factors), the model is also non-deterministic. Some approaches, such as those described in [13,14], generate sequences of inputs able to drive the system state in spite of random factors, but their application in complex systems is limited by scalability issues due to the space explosion problem, and by restrictive assumptions they implicitly make about the behavior of the system (for instance, they only consider "stable" states, in which the system waits for inputs or events [18]).

Other studies, including ones on fault injection, do not rely on a system model to generate a workload, but they assess its performance or dependability by adopting a workload representative of the real system workload that will be experienced during operation [6,11,12], in a similar way to performance benchmarks of non-distributed systems [21]. In other cases, synthetic workloads are randomly generated, in which the tester provides a probability distribution over the input space of the system [9,22,23], or provides a high-level description of the synthetic workload, e.g., using the *Synthetic Workload Specification Language* [24]. In such studies, the system states that are tested are only those ones exercised by the considered workload, and they do not consider the problem of tuning the workload in order to bring the system in "hard-to-reach" states. In particular, many fault injection studies randomly inject faults during an experiment, repeating this process several times and performing a very high number of experiments [25,26,8], which can be ineffective at uncovering vulnerable states of the system. More sophisticated fault injection approaches trigger the injection when a specific state of the system occurs [27,28,5]. For instance, Loki [5] considers the global state of a distributed system for triggering fault injection: in order to assure that a fault has been injected in a desired state, it performs an off-line analysis of execution traces and repeats the experiment if the injection has been triggered in a wrong state. However, these approaches still rely on a workload provided by the tester, either hand-written or using a representative workload, which does not assure that all important states are covered during testing. Compared to these works, our approach actively tunes the workload in order to cover a specific state specified by the tester, thus complementing experimental assessment approaches such as Loki.

3 Basic Concepts and Assumptions

According to the traditional definition [29], a Distributed System (DS) is composed by *processes*, which execute concurrently on a set of nodes, and by a *network*, which is the only medium through which the processes interact. Each process has its own clock and encapsulates some local resources. Local resources cannot be read or updated by any other process without an explicit request. Therefore, processes cannot share memory and the interactions among them only occur through messages exchanged on the network.

In order to design an approach for generating state-driven workloads, we make some practical assumptions about the architecture of a distributed system. We consider distributed systems in which a set of *services* is exported by a *frontend* process, which masks the complexity of the system to its users (Fig. 1). A client sends requests to the frontend process by means of one or more messages, the frontend interacts with the other processes of the DS and, once the computation has finished, replies to the client. This view of DSs applies to several systems, including orchestrated web services and three-tier web applications [29].

More formally, the frontend exports a set of services $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$ by an *interface*: each service can be invoked by the clients and it triggers different functionalities and actions in the DS depending on the actual values of the parameters sent in its requests. Without loss of generality, we assume that the tester defines the sets M_{u_i} before the assessment of the target system, where $m_{i,j} \in M_{u_i}$ is the j-th combination of parameters for the service u_i that a client can invoke on the frontend. A *service request* for the service u_i is a message produced by the client with parameters $m_{i,j}$ at time $t \in \mathcal{T} = \{0, \ldots, t_{\max}\}$ of the experiment, and can be represented by a pair $r \in M_{u_i} \times \mathcal{T}$. A *workload* is a set of service requests generated during an execution, and it is a subset W of the set W^* representing the space of all possible requests that can be submitted to the system: $W \subseteq W^* = \bigcup_{u_i \in \mathcal{U}} \{(m_{i,j}, t), m_{i,j} \in M_{u_i}, t \in \mathcal{T}\}$. In other words, a workload is an element of the *powerset* (the set of all subsets) of W^*, that is, $W \in \mathcal{P}(W^*)$.

The aim of the Workload Generator (WG) is to select a $\overline{W} \in \mathcal{P}(W^*)$ such that the DS reaches a *target state* (or any state from a *set of target states*), specified by the tester, during the execution of $\overline{W}$. The state of an individual process in the DS is referred to as *local state* of the process, whereas the *global state* of the DS, denoted with $s \in \mathcal{S}$, is the union of all the local states. The state of the process and of the DS is determined by the tester according to some high-level specification of the system, which takes into account the state of local resources as well as the state of computations performed by each process. An example of local state of a process could be DOWN if the process is failed, or UP otherwise; or INITIALIZING and WAITING FOR ACK to distinguish between different states of a computation. The global state of the system is specified by the tester through a *system model*. For instance, if we want test the correctness of a deadlock detection mechanism in a distributed DBMS, the system model and the global state would reflect the contents of the lock table and the distributed

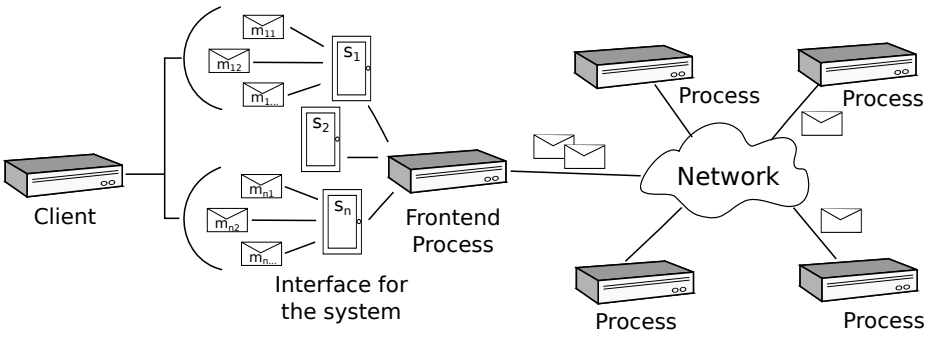


Fig. 1. The distributed system architecture considered in this work

transactions being performed. The system model could be represented using any formalism, such as Finite State Machines and Petri Nets.

A distributed system is intrinsically concurrent. The rate at which each process executes and the timing of the messages exchanged on the network are unknown. Moreover, the exchange of messages through the network is affected by several factors, including the delays for accessing to the network and for transmitting the contents of messages, and the delays introduced by the OS and by a middleware layer at both the ends of a communication. It follows that it is often impractical to predict the evolution of a DS, since it is difficult to precisely model all the factors that affect the system, especially when the system in the very complex and includes third-party and off-the-shelf hardware and software components. Moreover, the same sequence of client requests can produce different evolutions of the system due the randomness of these factors.

When an experiment is executed, the workload W causes the system to traverse one or more states, and to sojourn in each of them for a finite time. Let the *execution report* be the sequence $\{e_n\}_{n\in\mathbb{N}}$, where $e_n = (s_n, d_n)$ represents the state s_n traversed by the system during its n-th evolution of an execution, with a sojourn time d_n. Let $\mathcal{S}_G \subset \mathcal{S}$ be the subset of target states in which the tester aims to bring the distributed system: the minimum sojourn time τ is the time required by the tester to evaluate a property of the target system in $\mathcal{S}_G$, and represents a constraint for the WG. The *target hit ratio*, $p_{\mathcal{S}_G,\tau}(\mathcal{R}_W)$, applied on the set of execution reports $\mathcal{R}_W = \{r_1, \ldots, r_N\}$ obtained from one or more executions under workload W, estimates the probability that the workload W brings the system in the target state for a sojourn time greater than τ during an execution:

$$p_{\mathcal{S}_G,\tau}(\mathcal{R}_W) = \mathcal{P}\{\text{The DS reaches } \mathcal{S}_G \text{ for more than } \tau \text{ at least one time when executing } W\} = \frac{|\{r_i \in \mathcal{R}_W : \exists (s_k, d_k) \in r_i : (s_k \in \mathcal{S}_G) \wedge (d_k > \tau)\}|}{|\mathcal{R}_W|} \quad (1)$$

where the $|\cdot|$ operator represents the cardinality of a set. For instance, if the fault tolerance of the DS is being evaluated through fault injection, the DS has to sojourn in the target state long enough to allow a fault injection tool to detect the state and to inject a fault before the DS leaves the target state. In this scenario, $p_{\mathcal{S}_G,\tau}$ would represent the likelihood of a correct fault injection experiment, i.e., the fault is injected while the system is in the desired state.

The problem of generating a state-driven workload consists in searching for a workload $\overline{W}$ such that

$$p_{\mathcal{S}_G,\tau}(\mathcal{R}_W) > p_d \ , \tag{2}$$

that is, the likelihood to spend a period τ in the target state is high enough (i.e., greater that p_d) to allow an accurate and reproducible test. The search is conducted by the tester before performing the desired test. Eq. 2 provides a stop criterion for the search. Then, the tester supplies again $\overline{W}$ to the system, and performs the test when the system reaches a target state (e.g., it performs the actual fault injection experiment).

4 Proposed Approach

The proposed approach is based on a Workload Generator (WG) agent that interacts with the Distributed System Under Test (DS-UT) in a closed-loop configuration, as shown in Figure 2. The WG exercises the DS with a workload, analyzes its behavior, and modifies the workload until a specified target state is reached.

The WG follows a *system model* of the DS, which is adopted by the tester to specify the states of the system, and enables the WG to understand whether a target state has been reached (i.e., the system model is adopted for computing the execution reports from the raw events occurred and collected during the execution).

The WG works iteratively, by alternating at each iteration an *off-line* and an *on-line* phase. In the on-line phase, the WG executes the DS several times. At each execution, it first brings the DS in its initial state s_0 through a *reset* operation, it feeds the DS with a workload W, and observes its evolution for a fixed time period. Then, after the DS has stopped, there is an off-line phase in which the WG analyzes the behavior of the system through *logs* collected during execution, and evaluates whether the target state has been reached. If this was not the case, a new workload is computed and used in the next iteration. The distinction between the off-line and on-line phases allows to reduce the intrusiveness of the WG on the DS under test, since only minimal information is collected during the execution of the system, and most of the processing for analyzing the system evolution and computing the workload occurs in the off-line phase.

We divide the discussion of the proposed approach in three parts: *modeling* the system states, *monitoring* the execution of the DS, and *driving* the DS by tuning the workload.

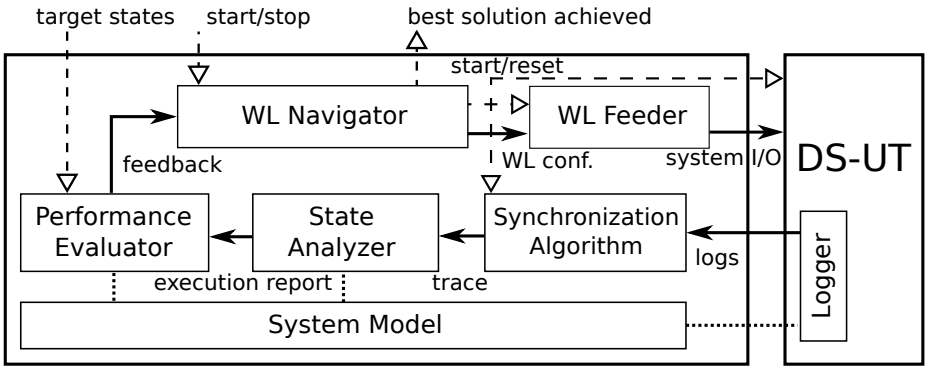


Fig. 2. The architecture of the proposed Workload Generator. Dotted lines represent a relation of dependence, while arrows represent dynamic interactions: striped arrows represent a control flow; continuous ones denote a data flow.

Modeling

Our approach is based on a *system model* of the DS under test, which allows to compute the execution report $\{(s_n, d_n)\}_{n \in \mathbb{N}}$, and to evaluate how close the workload brings the system to the target states.

Since the complexity of the system under test is typically very high, we assume that the system model provides an abstract and simplified view of the system. In particular, we do not require the system model to characterize the *timing of events* in the system, but only the *relationship* between events and states. As discussed in Section 3, timings in complex distributed systems, including communication and computation delays, can be unfeasible to characterize even in a probabilistic way, since they are tightly depending on each other, and involve third-party and off-the-shelf components, whose internals are unknown. Moreover, the timing of events also depends on the state of the system and on its workload (e.g., communication delays depend on the number of processes accessing the network at a given time): since the workload is iteratively modified by the WG to drive the DS in the target state, the characterization of delays would not hold when the workload is changed by the WG.

Petri Nets (PN) are the formalism that we adopt for modeling the DS under test, as they are a popular formalism that fits well for modeling concurrent systems. In the system model, transitions are triggered by local events occurring at the processes of the DS. Since the timings of events are not modeled, transitions are not timed, but only express the relationship between events and the state of the DS. The state is represented by the marking of the PN. In our approach, the system model is used in the off-line phase (*after* exercising the DS using a workload) to obtain, from raw event logs of an execution, the sequence of states that the system has followed during the execution.

Monitoring

The monitoring is realized by the Logger and the Synchronization Algorithm (Fig. 2). The process for generating an execution report is summarizing in Fig. 3. Each process of the DS logs the local events, timestamping the records with its own clock. When the experiment is over, the logs of local events are collected, and an offline synchronization algorithm, which is described later, performs the temporal sorting. Then, the State Analyzer obtains a sequence of markings of the system model, by mapping events in the logs to transitions in the model, and generates the execution report.

Regarding the synchronization, in order to obtain the evolution of the system from the analysis of events, we adopt an *off-line synchronization algorithm* to align the events of an execution on a single global timeline [5,30]. Off-line synchronization has been preferred over on-line synchronization approaches [29], such as NTP, since on-line synchronization protocols exchange packets during the execution of the system and can thus interfere with its evolution. Off-line synchronization is performed after execution, by correcting the timestamps of events recorded during the execution. The correction is performed using an estimate of the drift rate of the clocks, which is obtained by analyzing the round-trip time of a set of messages exchanged before and after the execution. Since the clock drift rate can only be estimated, the exact timing of an event is unknown; instead, the off-line synchronization algorithm provides, for each occurred event, a *lower* and an *upper bound* for the event on the global timeline, which represent the *uncertainty interval* in which the event has occurred. When the uncertainty intervals of two events are not overlapped, their ordering and the evolution of the system can be determined. When overlaps occur, the state of the DS is unknown in the overlapped region of the global timeline. More details about off-line synchronization algorithms can be found in [5,30].

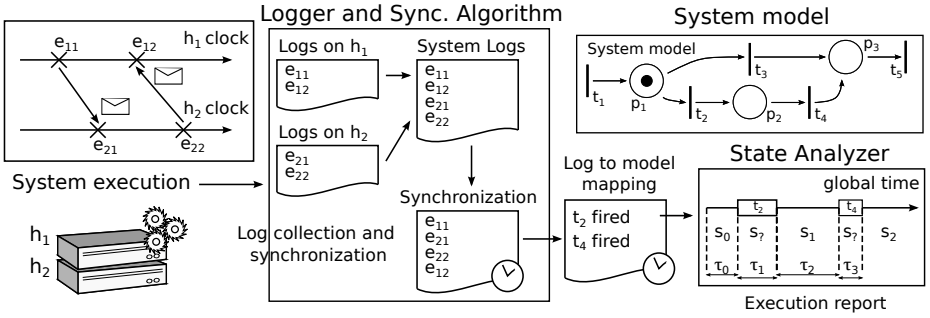


Fig. 3. The process for issuing execution reports. When the State Analyzer cannot determine the system state, due to the synchronization accuracy, it assigns the dummy state $\mathcal{S}_?$.

Driving

In our approach the WL Navigator and the WL Feeder drive the DS in the target states. The Feeder is the active part of the agent, which interacts with the DS generating the requests, i.e., the workload. The workload is synthesized by the Feeder on the basis of a *workload configuration* $w_c \in W_c$, which characterize the timing and the type of requests, either deterministically or statistically, through a set of parameters. The Navigator is the "smart" part of the agent, which searches for a proper workload configuration and analyzes the feedback from the system.

The distinction between Feeder and Navigator eases the search process, as the number of parameters in a workload configuration is a compact representation of a set of request $W \in \mathcal{P}(W^*)$ (actually, $|W_c| \ll |\mathcal{P}(W^*)|$), and frees the search algorithm from subtle details about individual interactions with the DS.

Herein, we consider a workload configuration $w_c \in W_c$ defined the following vector of parameters:

$$w_c = \left\langle T_{m_{1,1}}, T_{m_{1,2}}, \ldots, T_{m_{N,1}}, T_{m_{N,2}}, \ldots, D_{p_1}, \ldots, D_{p_M} \right\rangle \ . \tag{3}$$

These parameters are used by the Feeder to generate a set of requests $W \subseteq W^*$ to submit to the DS. In our case, the Feeder periodically invokes each service u_i using parameters $m_{i,j}$, with a period of $T_{m_{i,j}}$, $\forall m_{i,j}$. The Navigator explores several combinations of values for the $T_{m_{i,j}}$ parameters, in order to find a combination able to reach the target state. Moreover, we consider an additional set of parameters, D_{p_k}, which represent *delay factors* to introduce in one or more processes in the DS. Since the system may evolve very quickly, the target state could be reached only for short periods of time, leading to a low probability of hitting the target state for a sufficient time (Eq. 2). The introduction of small delays, by either slowing down a process (e.g., reducing its CPU quota by tuning its scheduling priority) or by forcing the process to sleep for short periods of time, increases the likelihood of sojourning in the target state for long enough.

The search for a proper workload proceeds through iterations. At each iteration, the Navigator changes the workload configuration on the basis of the feedback of the previous iterations, until a target state has been reached. To do so, two elements need to be defined, namely (i) a search algorithm to explore the space of workload configurations W_c, in order to find a workload $\overline{W}$ suitable for reaching the target state, and (ii) a criterion for assessing the "quality" of the current workload with respect to the target state.

We adopted in the Navigator a *genetic algorithm* (GA) [31]. A genetic algorithm evaluates a *population* of solutions (*individuals*) during the off-line phase of each iteration; then, a new population is generated from the previous one, by randomly *mutating* and *combining* previous individuals, on the basis of their quality (*fitness*).

Each individual of the population represents a workload configuration. An individual w_c is evaluated by executing the system under the workload W, generated by the Feeder using w_c, and by evaluating a *fitness function* on the

execution traces, which is computed by the *Performance Evaluator* component (Fig. 2). The definition of the fitness function is an important aspect of our approach, since it drives the Navigator in the search for the target state. The $p_{\mathcal{S}_G,\tau}(\mathcal{R}_W)$ (Eq. 2) cannot be used as fitness function, as it is not able to compare two different solutions that do not reach the target states (it would be 0 for both solutions). Instead, we propose a fitness function that evaluates the "distance" between the tentative solution and the target states, and the "continuity" of periods spent in the target state. Considering a set of execution reports obtained from one or more executions under workload W, $\mathcal{R}_W$, the fitness function is defined as

$$f_{\alpha,\varepsilon}(\mathcal{R}_W) = \frac{1}{|\mathcal{R}_W|} \sum_{r_w \in \mathcal{R}_W} \left(\sum_{(s,d) \in r_w} d^{\alpha} \cdot 10^{-\varepsilon \cdot \mathrm{dist}(s)} \right) \tag{4}$$

where the α rewards the continuity of the sojourn times (*permanence bonus*), while the ε penalizes the distance from the target states (*distance bonus*). When two solutions have a different distance, the closest solution is privileged (the *distance bonus* predominates); when two solutions have the same distance, the most continuous solution is privileged (the *permanence bonus* predominates). The times are normalized, i.e., $\sum_{(s,d) \in r_w} d = 1 \quad \forall r_w \in \mathcal{R}_W$ and $d \geq 0 \quad \forall (s,d) \in r_w$.

The function *dist* is a *distance measure* between any state and the target states, which is introduced to reward the workloads that are closer to the target states. We propose two different measures for the Petri net model we have adopted:

1. The minimum difference of tokens between the marking of the actual state and any target state: this measure is coarse and imprecise, however, is fast to calculate and it is easy to understand. Given a vector marking M, and let G be the set of target markings of a PN with m places, we have:

$$\mathrm{dist}(M, G) = \arg \min_{M' \in G} \left(\sum_{0 \leq i < m} |M_i - M'_i| \right) \tag{5}$$

2. The difference in the breadths on the reachability graph between the state and any target state, i.e., the minimum number of transitions (events) that have to happen to reach the closer target state. To compute this distance, we need to do a breadth-first search in the PN. Since we might not reach any target states, and because we do not need a precise value in the applications if the distance is greater than a fixed threshold, then we can limit the expansion of the PN at a depth R. Let BFS be an R-bounded breadth first visit, we have the follow:

$$\mathrm{dist}(M, G, R) = \arg \min_{M' \in G} \left(\mathrm{BFS}\left(M, M', R\right) \right) \tag{6}$$

In particular, we could save in memory all the states that are within range R from all the target states, avoiding to search in a graph each time. Whenever

we would need to compute the distance of a node, we would just need to check if M is in memory. If there is, then we would know the distance, else $dist(M, G, K) > R$.

To select the α and ε parameters of the fitness function, the following heuristics can be adopted. Considering any two states $s_l, s_{l+1} \in \mathcal{S}$ such that $\text{dist}(s_{l+1}) = \text{dist}(s_l) + 1$, it is possible select the parameters α, ε, according to the following system of inequalities:

$$\begin{cases} f_{\alpha,\varepsilon}\left(\{r_w^1 = \{(s_l, \theta \cdot \tau)_1\}\}\right) \geq f_{\alpha,\varepsilon}\left(\{r_w^2 = \{(s_l, \tau)_1, \ldots, (s_l, \tau)_k\}\}\right) \\ f_{\alpha,\varepsilon}\left(\{r_w^3 = \{(s_l, \eta)_1\}\}\right) \geq f_{\alpha,\varepsilon}\left(\{r_w^4 = \{(s_{l+1}, 1)_1\}\}\right) \\ \theta > 1;\ 0 < \eta < 1 \end{cases} \tag{7}$$

The first inequality expresses that it is better to sojourn in a state s_l for a $\theta \cdot \tau$ time, rather than visiting the same state up to k times for the same τ time. In this way, the tester can control the amount of reward to provide through the permanence bonus. The second inequality expresses that it is better to remain at least a η time in a state s_l, rather than to stay full-time in a state s_{l+1}, one level farther from the target states; this inequality can be adopted to control the level bonus. Developing the system we obtain the following relations:

$$\begin{cases} \frac{(\theta\tau)^\alpha}{10^{\varepsilon l}} \geq \frac{k\tau^\alpha}{10^{\varepsilon l}} \Rightarrow \alpha \geq \frac{\log k}{\log \theta} \\ \frac{\eta^\alpha}{10^{\varepsilon l}} \geq \frac{1}{10^{\varepsilon(l+1)}} \Rightarrow \varepsilon \geq -\alpha \log \eta \end{cases} \tag{8}$$

For instance, if we prefer staying in a state for a 25% longer time instead of visiting ten times a state for the same duration, and if we want that the sojourning for the 1% of total time in a "close" state s_l is better than staying for the 100% of total time in a "far" state s_{l+1}, then we have: $\alpha \geq \frac{\log 10}{\log 1.25} \geq 10.318$, $\varepsilon \geq -\alpha \log 0.01 = 2\alpha \geq 20.636$.

5 Preliminary Implementation

Herein, we present a preliminary implementation of our approach for testing a Flight Data Processing System (FDPS). FDPS is a distributed software developed in C++ which uses CARDAMOM, a fault-tolerant CORBA-compliant middleware. It is a part of an Air Traffic Control (ATC) system, in charge of managing Flight Data Plans (FDPs). An FDP is a data structure containing information about a flight; the goal of FDPS is to keep FDPs up-to-date. For example, FDPS has to analyze the actual position of aircrafts, retrieved from radar tracks, and update flight routes consequently, in order to efficiently allocate the flight space and to avoid flight collisions.

The architecture of FDPS (Fig. 4) is composed by a Façade component, which acts as the frontend of the system and is replicated by the CARDAMOM Fault-Tolerance (FT) Service, and by a set of three Processing Servers (PSs), managed by the Load-Balancing (LB) Service. Service requests are delivered to the Façade by the middleware: the Façade forwards requests to a specific PS according to

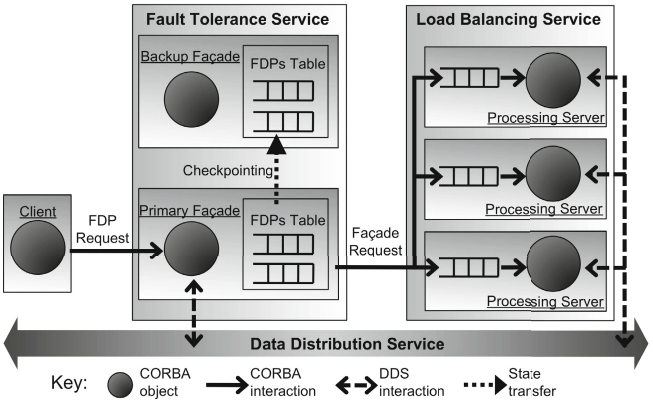


Fig. 4. A simplified view of the architecture of FDPS

a round robin scheduler; once the requests are completed, they are sent back to the Façade, which disseminates the updated FDP through a Data Distribution Service (DDS) and replies the clients.

The requests are relative to a specific flight track that is identified by means of an FDP-ID number: for each FDP-ID, the Façade dispatches no more than one request at time towards the PSs, and enqueues the others. The state of requests for each FDP is stored in a FDP Table of the Façade. Because the PSs are managed with a mono-threaded policy, the middleware in turn enqueues the requests forwarded to a PS if that PS is busy. The FT Service performs a *warm replication* of the Façade process: FDP Tables are checkpointed at each update and transmitted to backup replicas, which are activated in the case of failure of the primary replica.

Since our aim is to test the fault-tolerance and load-balancing mechanisms, in this case study we include in the system model (and thus in the definition of the state) the number and type of requests in the FDP Tables, and the number of requests enqueued at each PS. The system model was not included in this paper due to space constraints; it is described in [32].

In our preliminary implementation, we considered only one service u_1 of the DS, the UPDATE interface, which is invoked by specifying the FDP-ID. For each FDP-ID, we composed the workload configuration with two parameters, $T_{m_{1,i}}$ and D_i: the first one specifies the period between two requests for the i-th FDP-ID; the second one imposes on PSs the time to spend in processing the respective UPDATE invocations. The Feeder interacts with FDPS through the middleware (Fig. 5). We fixed six FDP-IDs and set the domains for the parameters of the workload configuration ranging from 500ms to 5s, with a step of 500ms. In our tests, the application has been deployed on 100Mbps Ethernet LAN, using 3 Processing Servers, one active Façade and one backup Façade replica.

We conducted a preliminary experiment in order to evaluate the feasibility of the proposed approach. The Navigator was configured to drive the system in a state fulfilling the following conditions: (i) one PS is idle, the other two PSs are

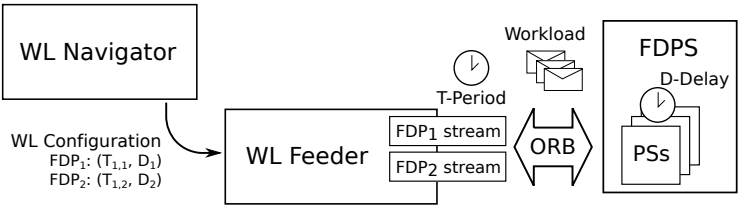


Fig. 5. The interaction among the WL Navigator, the WL Feeder and FDPS. The Feeder generates a *request stream* for each FDP-ID. Each request stream is configured through two parameters: a period between requests (T-Period), and a delay in processing of the requests associated with that stream (P-Delay).

busy, (ii) there are requests enqueued by the middleware for the two busy PSs, and (iii) there are between 1 and 5 enqueued requests for each FDP Table. This scenario is interesting because it represents an hard-to-reach condition, as also discussed in [33]: there should be an idle Processing Server, while the other two PSs should be busy and have requests enqueued by the middleware for them (i.e., the enqueued requests should not be forwarded to the idle PS). This condition is actually possible since the round robin scheduler selects the PS for a request regardless of whether it is busy.

In the experiment, we aimed at reaching this target state with a high probability $p_{G,\tau}(\mathcal{R}_W)$ for at least 0.25s. The WG found a solution with $p_{\mathcal{S}_G,\tau}(\mathcal{R}_W) \backsimeq 66\%$ in the first population, after 30 minutes. At the fourth population of solutions and 2 hours, the best solution found by the WG was able to reach the target state with $p_{\mathcal{S}_G,\tau}(\mathcal{R}_W) = 100\%$.

6 Conclusion and Future Work

In this paper, we discussed an approach for state-driven workload generation in complex distributed systems. Our approach, based on a genetic algorithm, iteratively tunes the workload until a desired target state is reached. Our preliminary implementation on a real-world case study (a Flight Data Processing System) confirmed the feasibility of the approach and provided encouraging results. In future work, we will perform a more throughout evaluation of the approach, by evaluating fault tolerance mechanisms in the FDPS through fault injection. Moreover, we aim to further develop our implementation of the approach, in order to make it portable to other systems and to freely distribute it.

Acknowledgments. This work has been supported by the projects 'Embedded Systems in Critical Domains" (CUP B25B09000100007) within the framework of "POR Campania FSE 2007-2013", and by the TENACE PRIN Project (n. 20103P34XC) funded by the Italian Ministry of Education, University and Research.

References

1. Chillarege, R., Iyer, R.: An experimental study of memory fault latency. IEEE Trans. Computers 38(6), 869–874 (1989)
2. Czeck, E., Siewiorek, D.: Observations on the Effects of Fault Manifestation as a Function of Workload. IEEE Trans. Computers 41(5), 559–566 (1992)
3. Meyer, J., Wei, L.: Analysis of workload influence on dependability. In: Proc. Intl. Fault-Tolerant Computing Symp., pp. 84–89 (1988)
4. Lefever, R., Cukier, M., Sanders, W.: An experimental evaluation of correlated network partitions in the Coda distributed file system. In: Proc. Intl. Symp. Reliable Distributed Systems, pp. 273–282 (2003)
5. Chandra, R., Lefever, R., Joshi, K., Cukier, M., Sanders, W.: A Global-State-Triggered Fault Injector for Distributed System Evaluation. IEEE Trans. Parallel and Distributed Sys. 15(7), 593–605 (2004)
6. Vieira, M., Madeira, H.: A dependability benchmark for OLTP application environments. In: Proc. 29th Intl. Conf. on Very Large Data Bases, pp. 742–753 (2003)
7. Arlat, J., Aguera, M., Amat, L., Crouzet, Y., Fabre, J., Laprie, J., Martins, E., Powell, D.: Fault injection for dependability validation: A methodology and some applications. IEEE Trans. Software Eng. 16(2), 166–182 (1990)
8. Meling, H., Montresor, A., Helvik, B., Babaoglu, O.: Jgroup/ARM: a distributed object group platform with autonomous replication management. Software: Practice and Experience 38(9), 885–923 (2008)
9. Tsai, T., Hsueh, M., Zhao, H., Kalbarczyk, Z., Iyer, R.: Stress-Based and Path-Based Fault Injection. IEEE Trans. Computers 48(11), 1183–1201 (1999)
10. Kiskis, D., Shin, K.: Swsl: A synthetic workload specification language for real-time systems. IEEE Transactions on Software Engineering 20(10), 798–811 (1994)
11. Duraes, J., Madeira, H.: Generic faultloads based on software faults for dependability benchmarking. In: 2004 International Conference on Dependable Systems and Networks, pp. 285–294. IEEE (2004)
12. Kalakech, A., Kanoun, K., Crouzet, Y., Arlat, J.: Benchmarking the Dependability of Windows NT4, 2000 and XP. In: 2004 International Conference on Dependable Systems and Networks, pp. 681–686. IEEE (2004)
13. Zhang, F., Cheung, T.Y.: Optimal transfer trees and distinguishing trees for testing observable nondeterministic finite-state machines. IEEE Transactions on Software Engineering 29(1), 1–14 (2003)
14. Nachmanson, L., Veanes, M., Schulte, W., Tillmann, N., Grieskamp, W.: Optimal strategies for testing nondeterministic systems. ACM SIGSOFT Software Engineering Notes 29(4), 55–64 (2004)
15. McMinn, P.: Search-based software test data generation: a survey. Software Testing, Verification and Reliability 14(2), 105–156 (2004)
16. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press (2000)
17. Holzmann, G.J., Smith, M.H.: An automated verification method for distributed systems software based on model extraction. IEEE Transactions on Software Engineering 28(4), 364–377 (2002)
18. Kerbrat, A., Jéron, T., Groz, R.: Automated test generation from sdl specifications. In: The Next Millennium–Proceedings of the 9th SDL Forum, pp. 135–152 (1999)
19. Fernandez, J.-C., Mounier, L., Pachon, C.: A model-based approach for robustness testing. In: Khendek, F., Dssouli, R. (eds.) TestCom 2005. LNCS, vol. 3502, pp. 333–348. Springer, Heidelberg (2005)

20. Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing approaches. Software Testing, Verification and Reliability (2012)
21. Jain, R.: The art of computer systems performance analysis. John Wiley & Sons New York (1991)
22. Avritzer, A., Weyuker, E.J.: Deriving workloads for performance testing. Software: Practice and Experience 26(6), 613–633 (1996)
23. Weyuker, E.J., Vokolos, F.I.: Experience with performance testing of software systems: issues, an approach, and case study. IEEE Transactions on Software Engineering 26(12), 1147–1156 (2000)
24. Kiskis, D.L., Shin, K.G.: A synthetic workload for a distributed real-time system. Real-Time Systems 11(1), 5–18 (1996)
25. Arlat, J., Aguera, M., Crouzet, Y., Fabre, J., Martins, E., Powell, D.: Experimental evaluation of the fault tolerance of an atomic multicast system. IEEE Transactions on Reliability 39(4), 455–467 (1990)
26. Basile, C., Wang, L., Kalbarczyk, Z., Iyer, R.: Group communication protocols under errors. In: Proceedings of the 22nd International Symposium on Reliable Distributed Systems, pp. 35–44. IEEE (2003)
27. Dawson, S., Jahanian, F., Mitton, T., Tung, T.: Testing of fault-tolerant and real-time distributed systems via protocol fault injection. In: Proc. Fault Tolerant Computing Symp., pp. 404–414 (1996)
28. Hoarau, W., Tixeuil, S.: A language-driven tool for fault injection in distributed systems. In: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing, pp. 194–201 (2005)
29. Coulouris, G., Dollimore, J., Kindberg, T.: Distributed Systems: Concepts and Design. Addison-Wesley (2001)
30. Poirier, B., Roy, R., Dagenais, M.: Accurate offline synchronization of distributed traces using kernel-level events. ACM SIGOPS Operating Systems Review 44(3), 75–87 (2010)
31. De Jong, K.A.: Evolutionary computation: a unified approach. MIT Press (2006)
32. Natella, R., Scippacercola, F.: State-Driven Workload Generation in Distributed Systems: System Model of an FDPS. Technical report (2013), `http://wpage.unina.it/roberto.natella/reports/`
33. Natella, R., Cotroneo, D.: Emulation of transient software faults for dependability assessment: A case study. In: Proc. European Dependable Computing Conf., pp. 23–32 (2010)

Towards Benchmarking of Functional Safety in the Automotive Industry

Mafijul Md. Islam[1], Behrooz Sangchoolie[2], Fatemeh Ayatolahi[2], Daniel Skarin[3], Jonny Vinter[3], Fredrik Törner[4], Andreas Käck[5], Mattias Nyberg[6], Emilia Villani[2], Johan Haraldsson[1], Patrik Isaksson[1], and Johan Karlsson[2]

[1] Advanced Technology and Research, Volvo Group Trucks Technology, Volvo AB
{mafijul.islam,johan.haraldsson,patrik.k.isaksson}@volvo.com
[2] Department of Computer Science & Engineering, Chalmers University of Technology
{behrooz.sangchoolie,fataya,emiliav,johan}@chalmers.se
[3] SP Technical Research Institute of Sweden
{daniel.skarin,jonny.vinter}@sp.se
[4] Volvo Cars Corporation
ftorner@volvocars.com
[5] QRTECH
andreas.kack@qrtech.se
[6] Scania AB
mattias.nyberg@scania.com

Abstract. Functional safety is becoming increasingly important in the automotive industry to deal with the growing reliance on the electrical and/or electronic (E/E) systems and the associated complexities. The introduction of ISO 26262, a new standard for functional safety in road vehicles, has made it even more important to adopt a systematic approach of evaluating functional safety. However, standard assessment methods of benchmarking functional safety of automotive systems are not available as of today. This is where the BeSafe (Benchmarking of Functional Safety) project comes into the picture. BeSafe project aims to lay the foundation for benchmarking functional safety of automotive E/E systems. In this paper, we present a brief overview of the project along with the benchmark targets that we have identified as relevant for the automotive industry, assuming three abstraction layers (model, software, hardware). We then define and discuss a set of benchmark measures. Next, we propose a benchmark framework encompassing fault/error models, methods and the required tool support. This paper primarily focuses on functional safety benchmarking from the Safety Element out of Context (SEooC) viewpoint. Finally, we present some preliminary results and highlight potential future works.

Keywords: Functional Safety, Fault Tolerance, Fault Injection, Robustness, Benchmarking, Safety Element out of Context (SEooC).

1 Introduction

Safety has always been an important property in the automotive industry. The safety provided can loosely be divided into *passive safety*, aiming at mitigating the effects of

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 111–125, 2013.

a crash, and *active safety*, aiming at preventing a crash altogether. An aspect, which is gaining in importance increasingly in the automotive industry, is that of *functional safety*. This is due to the fact that electronics have invaded virtually all vehicle functions and about 90% of all vehicle innovations are centered around software and hardware [1]. As opposed to passive and active safety provided by dedicated systems and functions, functional safety is an inherent attribute in systems indicating their ability to remain safe under various conditions, with and without faults. ISO 26262 [2], a new standard for functional safety in road vehicles, defines functional safety as absence of unreasonable risk due to hazards caused by malfunctioning behavior of E/E systems.

The bases of functional safety are the avoidance of faults (e.g. systematic software faults) or else the detection and handling of faults (e.g. random hardware faults) in order to mitigate their effects and thus prevent the violation of a safety goal by the embedded system [3]. To this end, ISO 26262 [2] provides requirements on an automotive safety lifecycle of electrical and/or electronic (E/E) systems within road vehicles. Furthermore, AUTOSAR (AUTomotive Open System ARchitecture) is a key enabling technology to manage the growing E/E complexity and provides mechanisms as well as systematic design approach to facilitate achieving functional safety of software-based systems [3]. However, standard assessment methods of evaluating functional safety of automotive systems are not available as of today. This is where the BeSafe (Benchmarking of Functional Safety) project [4] comes into the picture.

BeSafe project aims to lay the foundation for benchmarking functional safety of automotive E/E systems. A common way of evaluating functional safety will improve the industry's ability to provide safer vehicles. Benchmarking is also a way of evaluating to what extent the expected requirements of a system have been fulfilled. Consequently, benchmarking functional safety will be a valuable help in evaluating the fulfillment of safety goals and safety requirements, and will thus be a stepping stone in fulfilling the requirements stemming from the standards such as ISO 26262 [2] and IEC 61508 [5].

In the project, we identify benchmark targets in automotive electronic systems, assuming three different abstraction layers - model, software and hardware. We define benchmark measures, considering Safety Element out of Context (SEooC) as defined in Part 10 of ISO 26262 and with respect to a context. We then propose methods both experimental, for example, fault injection, and analytical, for example, probabilistic analysis of performing benchmarking along with required tool support. Finally, we are aiming for a functional safety benchmark framework that describes processes, methods and tools, defines how to use benchmark results, and establishes link to standards such as ISO 26262.

The rest of the paper is organized as follows: Section 2 summarizes works related to benchmarking of safety and dependability, Section 3 provides an overview of the BeSafe project, Section 4 introduces benchmark targets. Section 5 presents the proposed benchmark measures whereas Section 6 highlights methods and tools that are required for performing benchmarking of functional safety. Finally, Section 7 discusses preliminary results and Section 8 provides concluding remarks.

2 Related Work

The use of benchmarks has led to accelerated progress with respect to the measured capabilities in many areas. For example, performance benchmarking is now a well-established area that is led by organizations such as TPC (Transaction Processing Performance Council) and SPEC (Standard Performance Evaluation Corporation), and supported by major companies in the computer industry [6]. SPEC benchmark suites have driven the tremendous performance development of microprocessors.

Prior studies proposed a plethora of techniques to assess various dependability aspects of computer systems. However, only few reported works focus on dependability and robustness benchmarking. For example, the Dependability Benchmarking (DBench) project [6] presents a framework for defining dependability benchmark for computer systems, with particular emphasis on off-the-shelf (OTS) and OTS-based systems and proposes a benchmark validation approach. Furthermore, several earlier studies address benchmarking of software robustness [7] [8] [9]. Only recently, researchers have proposed a guidance framework for dependability assessment of AUTOSAR-based systems [10]. However, such approaches are not suitable for benchmarking functional safety in the automotive industry if we consider standard benchmarks that are currently in use, for example, for performance benchmarking.

Since 1997, benchmarking of vehicles with respect to safety has been performed in the *Euro NCAP* [11], assessing the ability of new cars to protect drivers, passengers, and pedestrians. This has led to safer cars as well as an increase in public awareness concerning vehicle safety. Euro NCAP results have also become a force in sales and marketing of new vehicles. The benchmarking of active safety systems is not yet standardized in the same way as for passive safety, but efforts in this direction are underway. For example, the *eValue* project [12] aims at defining a range of scenarios for standardized assessment of active safety systems. For functional safety, however, there are almost no standardized ways of assessment.

3 Overview of BeSafe

BeSafe project aims to lay the foundation for functional safety benchmarks for automotive electronic systems. We define a number of *Benchmark Targets* (BTs). A BT can in principle be any system or sub-system which has clear boundaries, and is equivalent to the word *element* used in ISO 26262 [2]. For each BT, we define a set of measures relevant for providing a useful benchmark along with methods for assessing those measures, and then evaluate those measures on the selected BTs.

Alongside the work on measures for the selected benchmark targets, the BeSafe project will define a general benchmarking framework in which the benchmarks will operate. This framework will include methodology and process, and tool support – both in terms of tools for performing the actual benchmarks and in terms of

supporting the benchmark measures in development tools. The contents of the benchmark result will be made up of multiple measures as defined by the project. Both quantitative and qualitative measures will be included, and the generation of the measurements considers analytical measurements, the process by which the element is developed, and empirical measurements, based on the realizations of the element (e.g., fault injection or robustness testing).

Each included measure will have a clear relation to the functional safety properties of the benchmark target. However, a single measure is typically not sufficient for the benchmark results to be useful. Instead the whole vector of measures will be considered for any particular use of a benchmark result. We focus on four generic uses which are of particular interest to benchmarking of functional safety: (a) *comparison*. Compare suitability of an element with respect to functional safety, during system development/integration; (b) *profiling*. Profile an element for identifying and highlighting strengths and weaknesses with respect to safety; (c) *requirements*. Safety-related requirements on the system, or its elements, can be communicated using benchmark details for a common understanding; and (d) *properties*. In a compositional way, safety benchmarks can aid in assessing system safety properties given safety profiles of its elements. We consider a generic "V" process model as our reference process to enable straight-forward mapping from BeSafe to ISO 26262. Moreover, we define a use case space in order to identify the realm in which the use cases for the BeSafe project reside. This use case space is a generic description of roles/actors (e.g., functional safety assessor, software developer, software supplier, E/E architect), activities (e.g., specification, evaluation, verification, assessment), process steps (e.g., concept phase, product development, production and operation), artifacts (e.g., E/E architecture, Function, ECU design, Software element) and so on.

The BeSafe is a 3-year (April 2011 – March 2014) research project funded by Vinnova (Swedish Governmental Agency for Innovation Systems) within Vehicle Development Program (FFI - Fordonsutveckling). The consortium consists of six partners that include Swedish automotive industries, university and research institute: Volvo AB, Volvo Cars Corporation, Scania AB, QRTECH, Chalmers University of Technology and SP Technical Research Institute of Sweden.

4 Benchmark Targets

This area deals with benchmarking of individual components and subsystem which are integrated in in-vehicle software. Some examples of such components could be individual modules in AUTOSAR or the entire AUTOSAR Basic Software (BSW)/ Run-time Environment (RTE). Approaches for assessment of application software components (SW-Cs) are also of interest and are considered. AUTOSAR BSWs such as CAN Transport Layer (CanTp), CAN Interface (CanIf) and CAN State Manager (CanSM) are considered as benchmark targets to demonstrate the effectiveness of proposed benchmark measures. Benchmark targets also include the error handling mechanisms on application level [13] as specified by AUTOSAR to evaluate the

effectiveness of those mechanisms. Furthermore, Simulink models that are used to generate code of AUTOSAR SW-Cs are potential benchmark targets to perform benchmarking at the model level.

The use case of a four-wheel brake-by-wire (BBW) consists of five ECUs (Electronic Control Unit) connected to a bus, see Figure 1. Each wheel has one corresponding ECU, and the central brake controller is located on the brake pedal ECU. The overall functionalities of the BBW are as follows:

- The brake pedal, connected to the central ECU, provides driver input for the requested brake torque. The brake torque calculator computes the driver requested torque and sends the value to the vehicle brake controller function.
- The vehicle brake controller then decides the required torque on each wheel. Each of the required brake torque values for the individual wheels is sent, together with the current measured vehicle speed, to the respective wheel ECUs.
- Based on the torque request received, current vehicle speed and wheel angular speed, the local brake-function decides appropriate braking force to apply to the wheel.

SW-Cs are based on AUTOSAR platform and developed as SEooC. The use case focuses on fault injection and interface testing of AUTOSAR SW-Cs. Test cases are generated based on operational conditions and functional description, including assumed safety requirements associated with the BBW at the SW-C level.

Benchmark targets also include software applications that can potentially be used in automotive domains. These applications can be found in embedded benchmark suites. MiBench [14] is one of these benchmark suites that contains more than twelve software components that are used in automotive domains. Fault injection can then help us benchmark the error sensitivity of these automotive applications for further in context assessment of the safety in a particular vehicle subsystem.

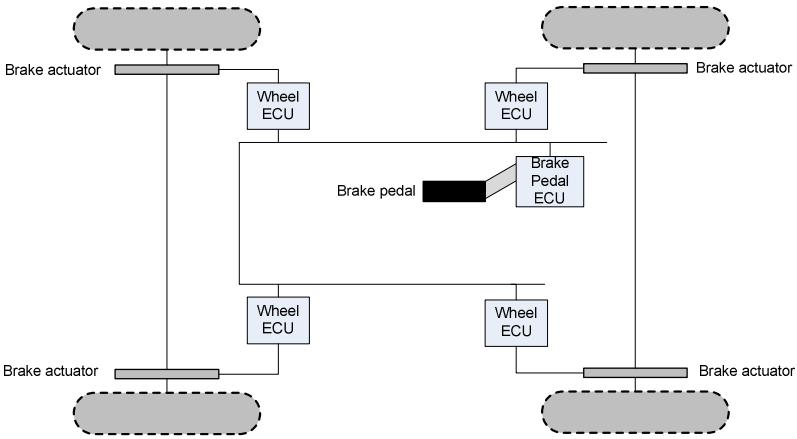


Fig. 1. Overview of the BBW

5 Benchmark Measures

In this section, we present and discuss a set of measures that can potentially be used for benchmarking of functional safety. The project considers measures for both experimental and analytical approaches. However, in this paper, we primarily discuss measures that are related to experimental approach of benchmarking.

5.1 Safety Element Out of Context (SEooC) and in a Context

Benchmark measures for a particular benchmark target (BT) can be developed either out of context or with respect to a context. The notion of "*out of context*" implies benchmarking a BT without prior knowledge of how the BT will actually be integrated and used in a complete system such as a vehicle as well as the impact of any failure of the BT on the system level. According to Part 10 of ISO 26262, a Safety Element out of Context (SEooC) is a safety-related element which is not developed for a specific "item". This means it is not developed in the context of a particular vehicle and is developed, based on assumptions, in accordance with ISO 26262. For example, a particular safety-critical software component can be developed and tested without exactly knowing the safety requirements of the system in which the component will actually be integrated. As a result, the software component required to be benchmarked "out of context" - independent of the target system.

In case of out of context measures, *failure mode distribution* that is derived through experiments such as fault injections and robustness testing are considered as one of the benchmark metrics. Fault injections can be performed to carry out robustness evaluation of a particular BT, considering a suitable fault or error model and results can be categorized as follows: (i) correct completion, (ii) incorrect completion, (iii) hang or silent, and (iv) abort or crash. From the BT point of view, correct completion in presence of faults or errors may include providing correct functionality, i.e., no impact of the fault/error, correct error code return or exception raised and handled, safe shutdown of functionality or service, fail signal and graceful degradation. Incorrect completion may include incorrect error code return, value failure (benign – value close to nominal, critical – value far from nominal) and timing failure. The hang or silent category may include outcomes such as lack of reporting an error or exception when one should be reported, not detecting the presence of erroneous input, appearance of looping indefinitely and not producing any new data or output. Finally, abort or crash may consider abnormal early program exit, restart of a task that failed so as to recover and system-killer or catastrophic. Benchmark measure is thus a vector of four elements to represent the failure mode distribution.

In case of measures related to a context, *failure severity distribution* that takes into account the impact of any failure of a particular BT on the system level is of particular interest. For example, the failure severity may be categorized as follows in decreasing order of severity (Severity 1 is the most severe): (a) Severity 1: Not acceptable, (b) Severity 2: Acceptable but non-optimal, and (c) Severity 3: Correct or expected. Severity level 1 may include failures that leave the failure of a software component undetected at the system level and may cause unpredictable system behavior.

Severity level 2 may include failure mode that does not provide the specified functionality at the system level in an optimal way and at the same time, does not compromise safety at the system level. Finally, Severity level 3 includes failure mode that behaves as specified or expected at the system level in an optimal way. Note that failure severity distribution proposed in this paper as a benchmark measure is different from the classes of severity that are defined in Part 3 of ISO 26262 and used for ASIL determination through hazard analysis and risk assessment.

Once the various severity levels are defined, it is required to perform mapping of the failure modes that are derived from the out of context benchmarking to failure severities with respect to a context. Benchmark measure is then a vector of three elements (Severity 1, Severity 2 and Severity 3). This mapping process requires exact knowledge of the functional safety requirements. For example, assuming the Engine Control System Model presented in the DBench project [6], one possible mapping between failure modes and failure severities is shown in Table 1. It is notable that depending on the context, the same failure mode of an "out of context" software component may be mapped to different failure severity levels with respect to a context. Indeed, in this paper, we focus on benchmarking functional safety assuming SEooC.

Table 1. Illustration of mapping between failure modes and failure severities

Failure mode / Failure severity	Severity 1	Severity 2	Severity 3
i. Correct completion			X
ii. Incorrect completion (value failure – produced value is close to the nominal value, benign)		X	
iii. Incorrect completion (value failure – produced value is far from the nominal value, critical)	X		

5.2 Measures

In this section, we describe some examples of safety and dependability benchmarks measures that are being investigated in the BeSafe project. These measures directed towards three specific benchmark targets, namely, AUTOSAR basic software modules, generic software components and Simulink models.

5.3 Measures for AUTOSAR

AUTOSAR specifies certain error handling mechanisms for certain types of errors for a number of basic software modules (BSW) [3]. For example, the error list of CAN stack includes CAN Bus Off, CAN Controller Hardware Timeout, CAN Transmission Buffer Full, etc. Furthermore, AUTOSAR specifies the way the architecture reacts to these errors according to the FDIR process (Fault Detection, Isolation and recovery) [13]. AUTOSAR details the specific items regarding error handling for each module as follows: (a) *Detection*: how the module detects or is notified of an error, (b) *Reaction*: the internal reaction of the module (e.g. internal stage change), (c) *Report*: how

the error is notified to other modules in the stack or to the AUTOSAR infrastructure, and (d) *Recovery*: how if the error is recovered or mitigated by the module. The goal is to provide an overview of dysfunctional behavior of the BSW and clarify error handling mechanisms to guarantee the same behavior for any BSW implementation as well as to permit a safer exchange of module [3]. Since the error list and the corresponding error handling mechanisms are standardized in AUTOSAR, it is possible to develop benchmark measures for BSWs.

Benchmark measures may simply consist of a checklist of each fault/error and corresponding handling mechanism (detection, reaction, report and recovery) for a particular implementation of a particular AUTOSAR BSW. Accordingly, a *failure mode distribution* that consists of the following categories can be derived for a particular AUTOSAR BSW: (i) No Impact, (ii) No Detection + (No Reaction/Report + No Recovery), (iii) Detection + No Reaction/Report + (No Recovery), (iv) Detection + Reaction/Report + (No Recovery), and (v) Detection + Reaction/Report + Recovery. This failure mode distribution can then be used as benchmark measures. Again, the failure mode distribution can be derived out of context and the results may then be mapped to failure severity with respect to a context. Benchmarking of functional safety of AUTOSAR BSWs thus consists of measures to conform whether a particular implementation of a particular BSW has fulfilled the requirements. Benchmark measures of BSWs may include answers in the form of yes/no or Euro NCAP-like rating. This can also be used for comparison across different implementations of the same AUTOSAR BSW and for profiling an arbitrary implementation of an AUTOSAR BSW.

Finally, AUTOSAR specifies mechanisms such as plausibility check, substitute values, voting, and program flow monitoring for error handling on application level [13]. Based on this, benchmark measures can be defined to evaluate the efficiency of such error management mechanisms. The benchmark measures can then be used for profiling and requirements to select a suitable subset of available error handling mechanisms for implementing fault-tolerant embedded applications.

5.4 Hardware Error Sensitivity Measures for Generic Software Components

In BeSafe, we use the term *generic software components* for small pieces of software, e.g., library routines, which provide generic functions such as sorting, checksum calculation and basic math. A generic software component is a building block that can be a part of an AUTOSAR Basic Software Module, or an AUTOSAR Software Component (SW-C). We use injection of bit flip faults in main memory and CPU registers as a way to measure the hardware error sensitivity of generic software components. We use this technique to compare the effectiveness of different software-implemented hardware fault tolerance (SIHFT) techniques. One possible way to measure[1] the

[1] All measures are conditional probabilities. For example, given that an error has occurred, Detected by Hardware corresponds to the conditional probability that the error is detected by hardware exceptions.

hardware error sensitivity is by means of an estimated probability distribution over the following experiment outcomes:

- *No Impact*: Program terminates normally and the error does not affect its output.
- *Time Out*: The program fails to terminate within a predefined time which is set to be approximately 10 times larger than the execution time of the workload.
- *Detected by Hardware*: Processor detects an error by raising a hardware exception.
- *Detected by Software*: Errors that are detected by software detection techniques.
- *Corrected by Software*: Errors that are corrected by software correction techniques.
- *Value Failure*: The program terminates normally, but the output is erroneous and there is no indication of failure (silent data corruption).

We consider an injected bit-flip to be covered if the outcome of the experiment is No Impact, Detected by Hardware, Detected by Software, Corrected by Software, or Time Out. Bit-flips resulting in Value Failure are considered to be non-covered. We thus define *error coverage* as the probability that a fault does not cause Value Failure.

5.5 Benchmark Measures for Simulink Models

Matlab/Simulink is a graphical block diagram language that is widely used for model-based development of systems. In the BeSafe project, we investigate benchmark measures for software described using Simulink blocks. Example of measures suitable for Simulink models are:

- *Error detection coverage*: Given that an error has occurred, this is the conditional probability that the error is detected.
- *Error recovery/masking coverage*: Given that an error has occurred, this is the conditional probability that the error is detected and correctly recovered/masked.
- *Error detection latency*: Number of cycles/iterations between an error has occurred and when the error is detected.
- *Failure mode distribution*: Vector of elements with the distribution of workload outcomes. Possible workload outcomes include: no impact, incorrect output, hang, detected by error detection mechanism, etc.

Some of the benchmark measures listed above are also valid for software and one project outcome is a comparison of results from fault injection experiments performed on both models and software.

5.6 Classification of Benchmark Measures

Benchmark measures must consider multi-dimensional aspects of functional safety and the requirements of relevant standards and specifications such as AUTOSAR and ISO 26262. Benchmark targets need to be analyzed and evaluated from different dimensions to formulate a complete set of benchmark measures as shown in Table 2. For example, ISO 26262-6 specifies that software design and implementation shall enforce low complexity for all ASIL (ASIL A – ASIL D). In this case, for example,

software complexity metrics such as cyclomatic complexity with a preset threshold value can be used as a benchmark measure to verify whether a particular implementation of a SW-C (developed as SEooC) conforms to the complexity requirements. This can be done either by software developers during unit implementation (coding) and testing or by integrators during integration and testing.

Table 2. Dimensions and classifications of benchmark measures

Measurement dimension	Measurement classification
Extent	Generic; Application specific
Abstraction layer	Model; Software; Hardware
Context	Safety Element out of Context (SEooC); Safety Element in a Context
Method	Experimental; Analytical; Formal analysis; Software metrics
Goal	Robustness; Performance; Conformance; Coverage; Latency
Software Development Lifecycle	Requirements/specifications; Architectural design; Unit design, implementation and testing; Integration and testing; Verification of safety requirements
Standards and specifications	AUTOSAR; ISO 26262; IEC 61508

6 Benchmark Framework

In this section, we present a benchmark framework that consists of fault/error models, methods and tools to facilitate performing benchmarking of functional safety.

6.1 Fault/Error Models

One aspect of functional safety benchmarking is *robustness* evaluation. Evaluating robustness of software implies analyzing software behavior by employing either invalid inputs or stressful environmental conditions [8]. Fault/Error model can potentially consist of set of corrupted parameters and this can be done via selective substitutions of parameter values or can be caused by bit-flips (fault) in registers or memory of the underlying ECU. In the project, three *fault/error* models are considered: (a) *data type*, (b) *fuzzing*, and (c) *bit-flip*. ISO 26262-6 also (strongly) recommends fault injection that includes injection of arbitrary values (e.g., by corrupting values of variables, by introduing code mutations, or by corrupting values of CPU registers).

Orthogonal Defect Classification (ODC) is a measurement technology that is consistently applied to a large number of IBM projects [15]. *In ODC, the fault type represents the defect in the source code, i.e. the cause of an error*. ODC employs six fault types (Assignment, Checking, Algorithm, Timing/Serialization, Interface, Function) related to code. Data type error model belongs to the Assignment class of ODC, though it could also be a Checking defect, resulting from a failing or missing check of a data value [16]. Fuzzing belongs to either *Assignment* or *Checking* classes in ODC [16]. Bit-flip belongs to the *Assignment* class of ODC defects [16].

Sensor fault models such as stuck-out of range, stuck-in range, oscillations and offsets will also be evaluated. Sensor failures are typically of permanent nature and are assumed to stress the benchmark target differently compared to experiment conducted with e.g. transient bit-flip faults. In addition, we will use bit-flips for hardware fault sensitivity benchmarking of software components. Note that sensor fault models and bit-flips are also presented in Annex D of Part 5 of ISO 26262.

6.2 Methods and Tools

This paper focuses on experimental benchmarking of functional safety although the project considers both experimental and analytical approaches. Furthermore, this paper emphasizes experimental benchmarking based on fault injection. In this regard, it is notable that ISO 26262 strongly recommends adopting fault injection and interface tests, for example, during software unit testing and integration testing.

6.2.1 Software-Implemented Fault Injection for AUTOSAR Based Systems

One approach is to intercept call to/from SW-C interface of an AUTOSAR-based system. This facilitates implementing data type and fuzzing error models to evaluate robustness of the SW-Cs. This detects data errors as specified in AUTOSAR [13]. Furthermore, error handling mechanisms as suggested by AUTOSAR [13] can successively be incorporated into a SW-C to evaluate the effectiveness of those mechanisms. This will eventually provide a set of error handling mechanisms that are most effective in error handling for a particular SW-C and facilitate benchmarking of those mechanisms. Approaches adopted by Ballista [8] and Fuzzing [7] can be applied in the context of AUTOSAR for evaluating robustness of BSWs. The robustness metrics based on CRASH [8] can be adopted as well. It is of particular interest to see how the results of fault injections based on bit-flips correspond to the results that are obtained by applying Ballista and fuzzing approaches. The software-implemented fault injection tool B-FEAT (BeSafe Fault Injection and Analysis Tool) will successively incorporate various fault/error models to evaluate software robustness with a view of benchmarking functional safety.

6.2.2 Benchmarking of Software Components with Respect to Hardware Faults

Here, we focus on out of context benchmarking of software components. Our fault injection tool, Goofi-2, [17] helps us evaluate the error sensitivity of software components with respect to transient hardware faults that manifest as bit flips.

The error sensitivity of a software component depends on several sources of variation, such as the inputs processed by the program, the way the source code of a program is implemented, the compiler optimization level, the microprocessor architecture, and the fault models used in the experiments.

Programs under test are executed on a PowerPC based microcontroller from Freescale. Goofi-2 uses a debugger with a NEXUS [18] interface to inject faults into the microprocessor instruction set architecture registers and main memory. We define

fault injection *experiment* as injecting one fault, single bit-flips or multiple bit-flips according to the fault model, and monitoring its impact on the workload. A fault injection *campaign* is a set of fault injection experiments with the same fault model on a given workload.

The fault injection tool defines faults as time-location pairs according to a fault-free execution of a workload. Where locations are randomly selected bits to be flipped from the memory words or the instruction set architecture registers, and time is a point in the execution trace. The analysis in [19] is used to exclude unreachable locations of the workloads from the fault space. In this way, the fault injection takes place on a register or memory location, just before it is read by executing instruction.

6.2.3 Benchmarking Simulink Models

We use a tool called MODIFI [20] to perform benchmark experiments on Simulink models. The purpose of the tool is to carry out early dependability evaluation of Simulink models, exercise and evaluate added error detection and recovery mechanisms in the model, as well as to create test cases for fault injection on the real system.

Mechanisms to inject faults are implemented using model blocks, which are inserted and activated during the execution of the model. Input to the model is provided by a stimulus file that include, e.g., sensor values and user inputs. The model is executed by the tool, which also creates an output file with values from the model such as actuator outputs. The output file is then compared by the tool to a fault-free execution of the model to find potential violations of safety requirements. The tool includes fault models for sensor faults and bit-flip faults, and can be easily extended with additional fault models.

Model-based development can be used in the context of ISO 26262 and ISO 61508, e.g., see [21] and [22]. As models can capture a system's specification and design, model-implemented fault injection can be used for verification activities that are performed in early phases of the development. One example in the context of ISO26262 is verification of the functional safety concept, which is performed during the concept phase. The functional safety concept includes, among others, fault tolerance mechanisms which later should be implemented in hardware or software parts. During the design phase, fault injection in models can provide evidence that the fault tolerance mechanisms are capable of preventing faults from violating top-level safety requirements. In later phases of the development, model-implemented fault injection can be used to, e.g., assess the effectiveness of fault tolerance mechanisms which have been implemented in models.

In the BeSafe project, we use a brake-by-wire (BBW) system, which has been developed by Volvo AB for research purposes, to evaluate benchmarks for Simulink models. Using the BBW system, we will benchmark different implementations of AUTOSAR fault handling mechanisms, e.g., several implementations of a plausibility check. Results from fault injection at the model level will be compared with fault injection experiments carried out at the software level. Table 3 summarizes methods for benchmarking of functional safety.

Table 3. Summary of methods for benchmarking of functional safety

Method	Abstraction Layer	Benchmark Target	Fault/Error Model	Tool support	Use case
Software implemented fault injection	Software	AUTOSAR SW-Cs and BSWs	Data type and fuzzing	B-FEAT	Profiling, Comparison, Requirements
Software implemented fault injection	Software	Software components, e.g. MiBench suite	Bit-flips	Goofi-2	Profiling, Comparison, Requirements
Model implemented fault injection	Model	Simulink models	Bit-flip and sensor faults	MODIFI	Profiling, Comparison, Requirements

7 Preliminary Results and Analysis

In this section, a sample of fault injection outcomes is presented. These results are obtained using Goofi-2 tool by injecting faults in instruction set architecture registers and memory words. We conducted 12,000 experiments for each workload.

As mentioned, the error sensitivity of a software component depends on several sources of variation, such as the inputs processed by the program, the way the source code of a program is implemented, the compiler optimization level, the microprocessor architecture, and the fault models. To this end, we have addressed variations in the inputs processed by a number of programs in [23]. For example, Figure 2 shows how different inputs contributed to the error coverage of the software implementation of the CRC 32-bit check. The inputs to the CRC program are strings of 0 to 99 characters, with CRC-1 corresponding to the smallest input and CRC-5 to the longest one.

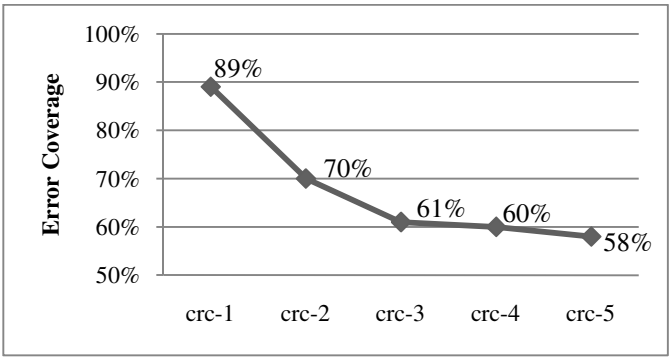


Fig. 2. Error coverage variation with respect to different inputs

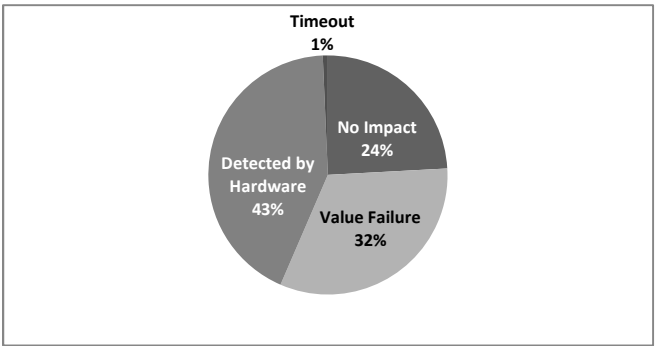


Fig. 3. Average failure mode distribution for the CRC program

Figure 3 also shows the average distribution of failure modes over all inputs of the CRC program. It is noticeable that 43% of the injected faults were detected by hardware exceptions, while still around 30% of the faults resulted in value failures. Therefore, in [23], we equipped software components like CRC with software-implemented hardware fault tolerance (SIHFT) techniques to increase the error coverage.

In addition, we plan to investigate several other sources of variation to improve our understanding on how to benchmark the error sensitivity of software components.

8 Conclusions

In this paper, we introduce the BeSafe (Benchmarking of Functional Safety) project that aims to establish the foundation of benchmarking functional safety of the automotive E/E systems with the goal of bridging the present-day gap in the standard assessment methods of evaluating functional safety. To achieve the goal, we identify the relevant benchmark targets and fault/error models as well as define benchmark measures. Based on those, we propose a preliminary benchmark framework that consists of methods and the required tool support to carry out benchmarking activities and present preliminary results. In future, we plan to enhance our focus on analytical benchmarking along with experimental benchmarking. Furthermore, we intend to incorporate the processes to perform benchmarking and how to use the results into the framework. Finally, we plan to demonstrate the applicability of the benchmark framework by using both generic applications and AUTOSAR-based systems, and extend the framework to establish links to relevant standards such as ISO 26262.

Acknowledgement. Authors would like to thank Dr. Martin Hiller who initiated and coordinated the BeSafe project while he was working at Volvo AB. The project is funded (50% of the total project budget) by Vinnova (Swedish Governmental Agency for Innovation Systems) within the Vehicle Development Program (Diary number: 2010-02114).

References

1. Lemke, K., Paar, C., Wolf, M.: Embedded Security in Cars. Springer, Berlin (2006)
2. ISO Standard, http://www.iso.org/iso/catalogue_detail?csnumber=43464
3. Technical Safety Concept Status Report, http://www.autosar.org/download/R4.0/AUTOSAR_TR_SafetyConceptStatusReport.pdf
4. BeSafe Project, http://www.vinnova.se/sv/Resultat/Projekt/Effekta/BeSafe/
5. IEC 61508 Functional safety of electrical/electronic/programmable electronic safety-related systems, http://www.iec.ch/zone/fsafety
6. Kanoun, K., et al.: DBench Dependability Benchmarks. Final Project Report, Dependability Benchmarking Project (IST-2000-25425) (May 2004)
7. Miller, B.P., Fredriksen, L., So, B.: An empirical study of the reliability of UNIX utilities. Communications of the ACM 33(12), 32–44 (1990)
8. Koopman, P., Devale, K., Devale, J.: Interface Robustness Testing: Experience and Lessons Learned from the Ballista Project. In: Kanoun, K., Spainhower, L. (eds.) Dependability Benchmarking for Computer Systems, pp. 201–226. John Wiley & Sons (2008)
9. Mukherjee, A., Siewiorek, D.P.: Measuring software dependability by robustness benchmarking. IEEE Trans. on Software Engineering 23(6), 366–378 (1997)
10. Piper, T., Winter, S., Manns, P., Suri, N.: Instrumenting AUTOSAR for dependability assessment: A guidance framework. In: Proc. of the 42nd DSN 2012 (2012)
11. Euro NCAP, http://www.euroncap.com/home.aspx
12. eValue (Testing and Evaluation Methods for ICT-based Safety Systems), project ICT-2007-215607 in EU FP7, http://www.evalue-project.eu/
13. Explanation of Error Handling on Application Level, http://www.autosar.org/download/R4.0/AUTOSAR_EXP_ApplicationLevelError.pdf
14. MiBench Version 1.0, http://www.eecs.umich.edu/mibench/
15. Christmansson, J., Chillarege, R.: Generation of an Error Set that Emulates Software Faults – Based on Field Data. In: Proc. of the 26th Annual Int. Symposium on Fault-Tolerant Computing, FTCS 1996 (1996)
16. Johansson, A., Suri, N., Murphy, B.: On the Selection of Error Model(s) for OS Robustness Evaluation. In: Proc. of the 37th DSN 2007 (2007)
17. Skarin, D., Barbosa, R., Karlsson, J.: GOOFI-2: A tool for experimental dependability assessment. In: Proc. of the 40th DSN 2010 (2010)
18. Nexus 5001™ Forum, IEEE-ISTO (1999), http://www.nexus5001.org/
19. Barbosa, R., Vinter, J., Folkesson, P., Karlsson, J.M.: Assembly-level pre-injection analysis for improving fault injection efficiency. In: Dal Cin, M., Kaâniche, M., Pataricza, A. (eds.) EDCC 2005. LNCS, vol. 3463, pp. 246–262. Springer, Heidelberg (2005)
20. Svenningsson, R., Vinter, J., Eriksson, H., Törngren, M.: MODIFI: A MODel-Implemented Fault Injection Tool. In: Schoitsch, E. (ed.) SAFECOMP 2010. LNCS, vol. 6351, pp. 210–222. Springer, Heidelberg (2010)
21. Conrad, M.: Testing-based translation validation of generated code in the context of IEC 61508. Formal Methods in System Design 35(3), 389–401 (2009)
22. Conrad, M.: Verification and Validation According to ISO 26262: A Workflow to Facilitate the Development of High-Integrity Software, http://www.mathworks.com/tagteam/71300_1D-4.pdf
23. Di Leo, D., Ayatolahi, F., Sangchoolie, B., Karlsson, J., Johansson, R.: On the Impact of Hardware Faults - An Investigation of the Relationship between Workload Inputs and Failure Mode Distributions. In: Ortmeier, F., Daniel, P. (eds.) SAFECOMP 2012. LNCS, vol. 7612, pp. 198–209. Springer, Heidelberg (2012)

Fault Injection in the Automotive Standard ISO 26262: An Initial Approach

Ludovic Pintard[1,4], Jean-Charles Fabre[1,2], Karama Kanoun[1,3], Michel Leeman[4], and Matthieu Roy[1,3]

[1] CNRS, LAAS, 7 avenue du colonel Roche, BP 54200, F-31031 Toulouse, France
firstName.lastName@laas.fr
[2] Univ de Toulouse, INPT, LAAS, F-31400 Toulouse, France
[3] Univ de Toulouse, LAAS, F-31400 Toulouse, France
[4] VALEO, 2 rue André Boulle, 94046 Créteil cedex, France
firstName.lastName@valeo.com

Abstract. Complexity and criticality of automotive electronic embedded systems is steadily increasing today. A new standard —ISO 26262— recommends methods and techniques, such as fault injection, to improve safety. A first goal is to use fault injection earlier at the design stage, particularly on models providing an appropriate level of abstraction, to identify errors in the handling of safety requirements. A second objective is to use the results of these model-based analyzes to efficiently identify targets and check their implementation by fault injection. Hence, a verification approach, based on fault injection, has to be defined to complement conventional testing methods and analyzes traditionally used in automotive development process. The paper discusses the various steps of this approach, the link between abstraction and implementation, and gives a brief illustration on a real automotive application.

Keywords: fault injection, automotive systems, ISO 26262, development process.

Introduction

As safety is a non-negotiable requirement for automotive critical embedded systems, the development process is evolving to assure that they should not lead to severe hazards. To respond to this trend a new standard have been proposed. ISO 26262, published in November 2011, defines the safety aspects of the development of electric and electronic automotive systems. A significant aspect of ISO 26262 is that it recommends fault injection to verify if systems are safe. To this end, this paper explores the integration of fault injection techniques throughout the development process, to perform efficient fault removal activities.

To illustrate our approach described in this paper, we use an electronic automotive component, the *Electronic Steering Column Lock* (ESCL), that has strong

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 126–133, 2013.

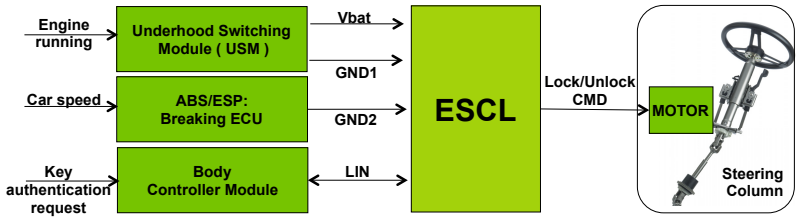


Fig. 1. ESCL Component and its Environment at System Level

safety requirements. Indeed, according to a conventional automotive scale of criticality, the highest Automotive Safety Integrity Level (ASIL D) is allocated to this component.

The ESCL, as shown in Fig. 1, is a component intended to manage the locking/unlocking of the steering column of a vehicle, so that if a thief tries to steal the vehicle, he cannot turn the vehicle wheels. However, this component may endanger the safety of the driver, since a spurious blocking of the steering column when the vehicle is at high speed could obviously threaten passengers safety.

The article is structured as follows. Section 1 describes the problem statement. In Section 2, we discuss the motivation and the benefits of using fault injection at the design stage, particularly on models. We briefly describe conventional fault injection on implementation in Section 3, and conclude on the lessons learnt.

1 Problem Statement

The new ISO 26262 standard highly recommends the use of fault injection techniques throughout the development process to verify safety requirements and safety mechanisms. Requirements of ISO 26262 highlight several targets for fault injection into the V-Cycle represented in Fig. 2.

The possible targets can be classified in two categories, namely (1) during the design steps down to the implementation, and (2) up to the verification and the validation of the integrated system. In the left side of the cycle, targets are *models* or representations of the system before the implementation. The right side of the cycle corresponds to an implementation of the system *components*, their integration and their validation.

Models: The standard recommends fault injection on models of system level and hardware level. There are two goals: *i*) to check that specifications related to the behavior in the presence of faults do not contain any omission or error, and *ii*) to ensure if the system implements appropriate mechanisms to prevent the violation of safety properties.

Components: These are effective targets linked to the verification of a system implementation, from the unit tests, through the integration phase, to the

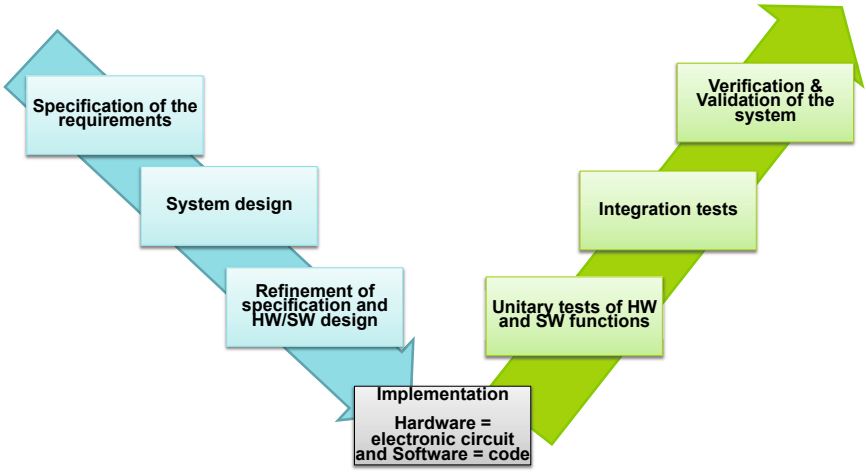


Fig. 2. ISO 26262 V-cycle Development Process

verification and validation of the complete system. At this stage, we seek to characterize the effectiveness of fault tolerance mechanisms (detection and recovery of errors) that have been implemented to increase safety as well as reliability and availability.

A first challenge is to define fault injection methods throughout the development process of an electronic/electrical system and particularly explore the possible contribution of fault injection at each stage of the process to improve the quality of the design. In a certain sense, we introduce the notion of multi-level fault injection and aim at analyzing possible links between targets, objectives and results at various development stages. This paper reports on our initial approach to tackle this problem.

Concerning integration, our second objective is to analyze how fault injection experiments related to a system can benefit from the results obtained to its components by fault injection. However, the composition of fault injection experiments in a hierarchical way is out of the scope of this paper, but will be highly explored during the project.

Fault injection [1] is a mature technology that has been successfully applied using several techniques on different targets [3–5, 7, 8, 14], that are usually components implementation. However, to the best of our knowledge, fault injection has not been studied throughout a development process at various development or abstraction levels in cooperation with usual testing methods.

2 Fault Injection During System Design

2.1 Fault Injection before Implementation

The high-level specifications of the system are progressively refined to provide detailed specifications of the hardware and of the software before implementation. During this refinement process, the form in which the specifications are

written may evolve. These specifications can be expressed in natural language, or in the form of a well-structured and formalized model (e.g., in an enterprise proprietary language), or in the form of a formal model, based on a standard formal language. Two kinds of requirements are usually distinguished: functional requirements and safety requirements.

At the initial step, the requirements are related to very high-level functions of the system, without addressing the system structure. A functional model may exist or can be built. Faults can be defined only at the same level of abstraction. Fault injection consists in assuming (or simulating) a failure of a basic function, and analyzing the impact of this failure on the other system functions and on the overall system functions. In this case, fault injection constitutes a way *i*) to check the impact of the failure of each basic function on the other function(s), and *ii*) to ensure that the safety requirements are satisfied. Hence, fault injection can be seen as a method very similar to the well-known and widely used approach, usually referred to as Failure Mode and Effect Analysis, FMEA [2], or Failure Mode and Effect and Criticality Analysis, FMECA (depending on whether the criticality is analyzed or not).

The primary benefit of fault injection is the same as for FMECA: the early identification of all critical system failure modes so they can be eliminated or minimized through design modification at the earliest phase in the development process. Another benefit is to identify the parts of the system and functions that require error detection and/or fault-tolerance mechanisms.

As for FMECA, the results of the fault injection analysis become more precise when more details about system functions are available (i.e., when the abstraction level of the system functional description becomes lower).

From the system requirements, a functional model is created. A fault injection "experiment" consists in selecting a failure mode of a function (or a component) and analysing its resultant effects on system operation, taking into account the overall safety requirements. The effects are usually defined with respect to the impact of the failures on the safety properties; they are referred to as "system failure modes". Each fault injection experiment corresponds to a single failure mode of one basic function (or of one component). A function or a component, with several potential failure modes, requires one experiment per failure mode. Indeed, each fault injection experiment corresponds to one line of a FMECA worksheet (or spreadsheet). Examples of information items that can be included in one line (or provided after a fault injection experiment) are: identification of the basic function (or component) analyzed, its potential failure mode addressed, the potential causes of this failure mode, the local effect of the failure mode, the next assembly layer effect, system level effect, the risk level, detection mechanisms to put in place, actions for further investigation.

Even though, the analyses can be performed manually for high-level preliminary design, the support of a modelling formalism and a tool becomes mandatory as soon as detailed information becomes available.

From a modelling point of view, at a high-level, the analyses can be performed based on data flow diagrams and/or state charts, to help the analyst to propagate

errors between the components of the system. When more details are available, languages such as UML (Unified Modelling Language) or AADL (Architecture Analysis Design Language) can be used. Their main advantage is that they have been extended to perform quantitative dependability assessment of critical systems (see e.g., [9, 10]).

Finally, several simulation modelling languages have been used for fault injection. At these levels, the model can be very similar to the real implementation of the system (see e.g., VHDL [6], SystemC [11], Matlab/Simulink [12] or SCADE [13]).

2.2 Illustration on a Case Study

The first objective of our method is to determine a high-level abstraction of our system. Fig. 1 presents the relation between ESCL and its environment. Indeed, a component has dependencies with other components. The dependencies are related to the inputs and outputs, because they link the components. Hence, fault injection at this level consists in propagating component failures and errors through the relationships between blocks.

Then, we have to define the safety requirements this system has to verify. There are two safety requirements, called safety goals according to ISO 26262, that must be ensured:

- SG1 = The ESCL must not lock the steering column when the vehicle speed is greater than 10 km/h. (ASIL D)
- SG2 = If the steering column is locked, the ESCL must prevent to start the engine of the vehicle. (ASIL A)

For example, SG2 will be specified at this level as follow: the ESCL should not send erroneous messages, via the LIN bus, stating that the steering column is unlocked while it is locked.

All safety requirements, at each level, are important because they define a set of invariants that must be satisfied by the system. Would an invariant be violated, a hazard may occur. To satisfy these properties, the design should explicitly exhibit each safety mechanism that deals with a property. Here, the criticality of a mechanism can be defined according to the ASIL level of the safety requirement, and so, the targeted safety mechanisms must be evaluated by fault injection techniques.

Then, the error model has to be defined. At this level, we can identify the hardware architecture of our case study with five components, and the associated five links. There are four electrical links, with four failures modes: *i*) there is no power when it is required, *ii*) there is power while it is not required, *iii*) oversupply, *iv*) and under-supply. The failure modes of the fifth link, a bidirectional LIN bus, are the following: no message transmitted, erroneous message transmitted, corrupted message.

Following the approach described in Section 2.1, the fault injection applied to the functional description of the ESCL enables the identification of critical blocks and their effect on the system. Considering the link between them,

we can identify whether a safety property may be violated. A critical path is the propagation of an error through the link that violates the safety properties.

Let's take the example of a corrupted message from the Body Controller Module ordering to lock the column when the vehicle is at high-speed. This could violate SG1, and the study of the architecture allows to check whether a safety mechanism exists — here, the ABS/ESP switches off the ESCL when the speed is larger than a threshold and hence the ESCL will not lock the column. The system level architecture shown on figure Fig. 1 can be considered as the first modeling level. However, this description is at a too high level to verify the real hardware architecture or the software architecture of the ESCL.

Considering the hardware, a more detailed model should be used to describe the architecture in terms of subcomponents: sensors, micro-controllers, memories, power supply units. This detailed architecture is represented on Fig. 3.

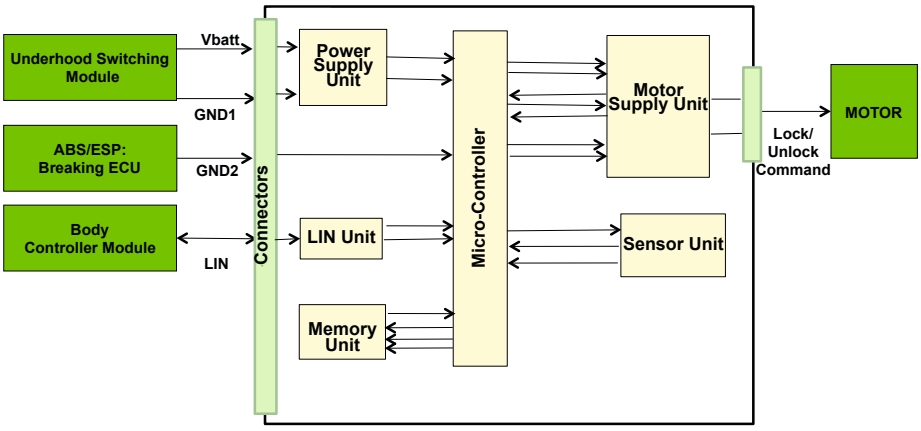


Fig. 3. Architectural Model of ESCL Component

Considering the software, modules and their dependencies can be modeled through function and procedure calls on a static view with a communication diagram, and then with sequence diagrams for the dynamic representation of the interactions. The robustness of implemented components running on the hardware must be evaluated according to a fault model:

- Hardware errors: errors on the inputs of the micro-controller, errors at the interfaces of software module due to a corruption of the memory, or error in sequences and with timing constraints.
- Software errors: coding faults should be represented by malfunctions on the interface of each software module.

3 Fault Injection during Verification

On the right-hand side of the V-cycle, an implementation of the system is available, in the form of a hardware support system, a set of software components,

and later on in the form of a global system in which the software components are integrated on the hardware system. All conventional fault injection methods are applicable.

Our recommendation is to use the results of the analyses carried out during the design phase to guide fault injection experiments in the verification phase. For example, fault injection campaigns will focus on critical components identified earlier. In our case study, we concentrate our analysis on the internal architecture of an ESCL component. The goal could be to activate error detection and error recovery mechanisms according to a fault model and evaluate their robustness. The aim of the experiments will be for example to:

- Check the correct implementation of the system together with the associated error detection and fault-tolerance mechanisms.
- Assess the error detection coverage and error recovery coverage of safety mechanisms.

In addition to the conventional aims of fault injection experiments targeting concrete components, the objective is also to find complementarities with usual testing methods applied in the automotive industrial domain in order to optimize the validation process.

Conclusion

The use of fault injection in the development of safety critical embedded automotive systems is explicitly mentioned in the ISO 26262 development standard. One can easily understand that similar types of techniques are nowadays used during the testing phase of embedded automotive systems. Such techniques include in particular FMECA analyses. However, advocating fault injection at various levels of the development of the system poses several challenges, not only with respect to the ISO 26262 application in the automotive domain, but raises more general scientific challenges. In particular, how to handle the complementarities between FMECA and fault injection at various stages of the development process.

We observed that FMECA is similar to fault injection when targeting models. Indeed, these concepts should mutually enrich each other because they share the same objectives. Yet, in practice, FMECA is often applied to coarse grain models. The short-term objective of this work is to show that the concept of FMECA can be applied to more fine grain structural and behavioral models, reaching finally the implementation. Conventional fault injection is the major technique on implemented components, but one can understand that FMECA and Fault Injection are overlapping concepts as models become more and more detailed. Conversely, fault injection automatized on a detailed model can produce similar results as those expected with FMECA.

In this paper, we have shown how model-based fault injection could be of interest to identify drawbacks in the handling of safety requirements but also to guide lower layers fault injection experiments, for instance with the identification of key targets for conventional verification by SWIFI.

References

1. Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 1(1), 11–33 (2004)
2. Bouti, A., Kadi, D.A.: A state-of-the-art review of fmea/fmeca. Int. Journal of Reliability, Quality and Safety Engineering 1(04), 515–543 (1994)
3. Carreira, J., Madeira, H., Silva, J.: Xception: A technique for the experimental evaluation of dependability in modern computers. IEEE Transactions on Software Engineering 24(2), 125–136 (1998)
4. Cotroneo, D., Lanzaro, A., Natella, R., Barbosa, R.: Experimental analysis of binary-level software fault injection in complex software. In: 2012 Ninth European Dependable Computing Conference, pp. 162–172. IEEE (2012)
5. Hsueh, M., Tsai, T., Iyer, R.: Fault injection techniques and tools. Computer 30(4), 75–82 (1997)
6. Jenn, E., Arlat, J., Rimen, M., Ohlsson, J., Karlsson, J.: Fault injection into vhdl models: the mefisto tool. In: Digest of Papers Twenty-Fourth International Symposium on Fault-Tolerant Computing, FTCS-24, pp. 66–75. IEEE (1994)
7. Karlsson, J., Liden, P., Dahlgren, P., Johansson, R., Gunneflo, U.: Using heavy-ion radiation to validate fault-handling mechanisms. IEEE Micro 14(1), 8–23 (1994)
8. Koopman, P., Sung, J., Dingman, C., Siewiorek, D., Marz, T.: Comparing operating systems using robustness benchmarks. In: Proceedings of The Sixteenth Symposium on Reliable Distributed Systems, pp. 72–79. IEEE (1997)
9. Rugina, A.-E., Kanoun, K., Kaaniche, M.: The adapt tool: From aadl architectural models to stochastic petri nets through model transformation. In: Seventh European Dependable Computing Conference, EDCC 2008, pp. 85–90 (2008)
10. Rugina, A.-E., Kanoun, K., Kaaniche, M.: Software dependability modeling using aadl (architecture analysis and design language). International Journal of Performability Engineering 7(4), 313 (2011)
11. Shafik, R.A., Rosinger, P., Al-Hashimi, B.M.: Systemc-based minimum intrusive fault injection technique with improved fault representation. In: 14th IEEE Intl. On-Line Testing Symposium, IOLTS 2008, pp. 99–104. IEEE (2008)
12. Svenningsson, R.: Model-implemented fault injection for robustness assessment. QC 20111205 (2011)
13. Vinter, J., Bromander, L., Raistrick, P., Edler, H.: Fiscade-a fault injection tool for scade models. In: 2007 3rd Institution of Engineering and Technology Conference on Automotive Electronics, pp. 1–9. IET (2007)
14. Winter, S., Sârbu, C., Murphy, B., Suri, N.: The impact of fault models on software robustness evaluations. In: 2011 33rd International Conference on Software Engineering, ICSE, pp. 51–60. IEEE (2011)

A GPS Spoofing Resilient WAMS for Smart Grid

Alessia Garofalo, Cesario Di Sarno, Luigi Coppolino, and Salvatore D'Antonio

Department of Technology, University of Naples Parthenope, Naples, Italy
{alessia.garofalo,cesario.disarno,
luigi.coppolino,salvatore.dantonio}@uniparthenope.it
http://www.dit.uniparthenope.it/FITNESS/

Abstract. Smart grids provide efficiency in energy distribution, easy identification of disturbance sources, and fault prediction. To achieve these benefits a continuous monitoring of voltage and current phasors must be performed. Phasor Measurement Units (PMUs) allow measurements of the phasors. A Wide Area Measurement System uses PMUs placed in different locations to assess the status of the power grid. To correctly analyze the phasors provided by PMUs, phasors must refer to the same time. For this reason each PMU uses the clock provided by a GPS receiver. GPS receiver is vulnerable to spoofing attack and it is a single point of failure. In this context we examined Network Time Protocol (NTP) as an alternative time source when the GPS receiver is compromised. In this paper a resilient architecture is proposed that is able to detect and react to the GPS spoofing attack. Experimental tests have shown the effectiveness of our solution.

Keywords: Smart Grid, Wide Area Monitoring System, GPS Spoofing Attack, Phasor Measurement Unit.

1 Introduction

Power grids were designed in order to meet requirements that were defined in the 20th century when the goal was "to keep lights turned on". Today, the requirements expected to be fulfilled by power grids have changed. The increasing load and consumption demands increase electricity issues, such as blackouts, and overloads. In July, 2012 for two days, India experienced blackouts that involved a large portion of the country's power grid. Specifically, a 9% gap was estimated between the effective energy requirements and the available energy amount [1] [2]. In the afternoon of September 8, 2011, an 11 minutes-long system disturbance occurred in the Pacific Southwest, leading to cascading outages and leaving approximately 2.7 million customers without power. The failure of the power grid was due to the bad redistribution of the power flow caused by the failure of a transmission line. Other examples of power grid blackouts due to different types of failure are reported in [3] [4] [5] [6] whereas a security analysis of the technologies which enable data collection in power grid and in other critical infrastructures is provided in [7] [8] [9].

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 134–147, 2013.

Smart Grid systems represent the natural evolution of the power grid. The term smart grid defines a self healing network equipped with dynamic optimization techniques that use real time measurements to minimize network losses, maintain voltage levels, and increase reliability. Operational data collected by the smart grid are analyzed and they allow system operators to rapidly identify the best strategy to secure against attacks, vulnerabilities, faults and so on, caused by various contingencies [10]. In order to monitor the status of the smart grid Wide Area Monitoring Systems (WAMSs) are used. WAMSs make use of devices distributed throughout the power grid that measure the key parameters to detect anomalous conditions.

Today Phasor Measurement Units (PMUs) are the most commonly used devices in WAMS. In particular, PMUs are devices that perform measurements of real-time phasors of voltages and currents to provide information about power grid status. The time synchronization between different PMUs is required to understand the global status of the power grid at the same time. This is because events occurring in one part of the grid affect operations elsewhere, and they also extend to other systems beyond the grid that rely on stable power. Time synchronized measurements produced by PMUs are called synchrophasors. In order to obtain simultaneous measurements of phasors detected from different PMUs installed across a wide area of the power system, it is necessary to synchronize these times, so that all phasor measurements belonging to the same time are truly simultaneous. Each PMU uses a Global Positioning System (GPS) receiver [11] to take a unique timestamp within the global system. One of the main problems affecting smart grid monitoring is the spoofing of the GPS signal provided to the GPS receiver [12]. The GPS signal can be forged in order to mislead the GPS receiver that uses it. This type of attack is called "GPS spoofing" [13] and more details are provided in Section 4. If an attacker forges the timestamps provided by GPS to a PMU, it could cause variations in measured phase angles. The difference in the phase angle between two PMUs indicates that the power between the regions measured by each PMU has changed. These variations could compromise the stability of the system in such a way that grid operators or automatic response systems would make incorrect decisions as powering up or shutting down generators. Incorrect decisions can cause blackouts or damages.

Many techniques are available in order to detect the GPS spoofing attack. These techniques are based on different approaches as: monitoring the absolute GPS signal strength; monitoring the relative GPS signal strength; monitoring satellite identification codes and the number of satellite signals received [14]. While different techniques are available to detect GPS spoofing attack no remediation technique was proposed.

In this paper we propose an architecture resilient to GPS spoofing attack. In particular our architecture provides capabilities of detection and remediation for the GPS spoofing attack. To design this architecture we analyzed requirements in terms of maximum time delay required by PMUs to avoid loss of synchronization. Also we analyzed the time accuracy provided by the GPS receiver to the PMUs. Thus we identified a particular implementation of the Network Time

Protocol (NTP) that offers the same accuracy as GPS receiver and that satisfies the PMU time requirements to avoid synchronization losses. So we developed a new component called "Spoofing Detector" placed between PMU device and the two time sources. The main time source is provided by the GPS receiver while backup time source is provided by NTP. Spoofing Detector detects the GPS spoofing attack and activates the remediation i.e. it switches from main to backup time source to provide PMU device with correct timestamp even under attack. Our architecture provides intrusion tolerance capabilities using both detection techniques of GPS spoofing attack and two external time sources.

The paper is organized as follows. Section 2 provides an overview about WAMS with reference to power grid. Section 3 describes the PMU devices and the way how they perform measurements of the synchrophasors. Section 4 presents the GPS spoofing attack that affects each PMU that uses a GPS receiver. In this section several techniques are discussed to detect this attack. Section 5 presents the synchronization protocol NTP as a candidate backup time source to use when the GPS receiver is compromised. Section 6 describes the resilient architecture proposed to detect and react to the GPS spoofing attack. Section 7 provides details about the implementation of the proposed architecture. Section 8 describes an attack model on the architecture proposed and presents the experimental results obtained.

2 Background

2.1 Wide Area Monitoring System

The power grid is composed of three main components: power plant, transmission substation and distribution grid. The power plant produces simultaneously three different phases of AC power with 120 degrees offset from each other. The three-phase power feeds a transmission substation. This substation uses large transformers to increase the generator's voltage up to extremely high voltages to reduce transmission line losses on long-distance. Distribution grid is the final stage of energy conversion before the electricity is supplied to end users.

The simplified architecture adopted today to monitor the power grid is shown in Figure 1.

PMUs are devices that use GPS signals as a common time source and analyze the waveforms of different transmission lines at different locations across a wide-area system at the same moment. In particular they perform a sampling of the waveforms provided by transmission lines and generate the phasors. These phasors are timestamped using the same clock provided by the GPS receiver. These synchronized phasors are called synchrophasors. Such timestamps can be used to compare collected synchrophasors with microsecond precision. In fact, the Phasor Data Concentrator (PDC) gathers the data provided by different PMUs and it performs a comparison between the synchrophasors to assess the status of power grid. A PDC can exchange phasors with PDCs at other locations to perform wide are monitoring.

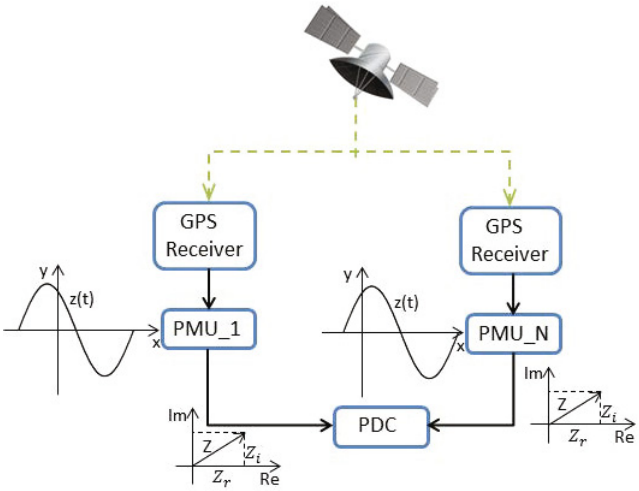


Fig. 1. Smart Grid Monitoring Architecture

The reference standard for PMU is IEEE Standard C37.118 [15]. It discusses about the definition of a synchronized phasor, time synchronization, method to verify compliance of measurements with the reference standard, and message formats for communication with a Phasor Measurement Unit (PMU).

3 Phasor Measurement Unit (PMU)

PMU devices are used in WAMSs in order to monitor power grids. In particular, PMUs analyze the 50/60 Hz AC waveforms provided by the power grid and they calculate the synchrophasors. The typical sinusoidal waveform analyzed by a PMU is:

$$z(t) = A_m * \cos(\omega t + \phi); \quad \omega = 2\pi f; \tag{1}$$

where f is the instantaneous frequency and A_m is the magnitude of the sinusoidal waveform. Waveform (1) can be represented as the phasor:

$$\bar{z} = Z_r + jZ_i = \frac{X_m}{\sqrt{2}} * e^{j\phi}; \tag{2}$$

where $\frac{X_m}{\sqrt{2}}$ represents the Root Mean Square (RMS) value of the waveform and ϕ is its phase angle relative to a cosine function at the frequency of the nominal system synchronized to Universal Time Coordinated (UTC). The time synchronization is provided by a GPS receiver. The advantage of referring phase angle to a global reference time is helpful in capturing the wide area snapshot of the power grid. The most common technique for determining the phasor representation of an input signal is to use data samples taken from the waveform,

and apply the Discrete Fourier Transform (DFT) to compute the phasor. Also, the obtained representation of the phasor is independent from the frequency of the signal $z(t)$.

So, the PMU calculates the voltage and current synchrophasors. Different PMUs are installed in different locations in order to obtain the global status of the power grid. In particular, the IEEE Standard C37.118 standard [15] defines the transmission rate of data generated by PMU. This rate changes if the system is 50 or 60 Hz. In Figure 2, the number of frames per second transmitted by PMU is shown for different types of systems.

System Frequency	50 Hz		60 Hz		
Frame per second	10	25	10	20	30

Fig. 2. PMU: phasor data transmission rate

Important topics covered by the mentioned standard concern the application of timestamps, the definition of the message format for communications between PMUs and the method to verify the measurement accuracy. The accuracy is expressed as the magnitude of the vector difference between the theoretical phasor and the phasor estimated by the measuring device, expressed as a fraction of the magnitude of the theoretical phasor. The magnitude of the vector difference is called Total Vector Error (TVE) and it is given by the following formula:

$$TVE = \sqrt{\frac{(Z_r(n) - Z_r)^2 + (Z_i(n) - Z_i)^2}{(Z_r^2 + Z_i^2)}} \tag{3}$$

where n represents the measurement time, $Z_r(n)$ and $Z_i(n)$ are the values measured by PMU, while Z_r and Z_i are the theoretical values of the input signal at the instant of time of measurement, determined by (2). The loss of synchronization occurs when the TVE value exceeds the value limit of 1%. There are three types of errors that can increase the TVE value: phase-angle measurement error; magnitude measurement error and time synchronization error. In this paper we analyze only the case of error in time synchronization. This is because when a GPS spoofing attack is performed successfully, the remediation architecture that we propose must satisfy specific time constraints to avoid losses in synchronization.

If a PMU is not accurately synchronized with UTC, then the measured phase will not match the true signal phase. In particular a phase error of 0.01 radians in (3) will cause 1% TVE. So we can calculate the maximum allowed time error before the synchronization loss using the following equation:

$$\Delta t = \frac{\phi}{2\pi f} \tag{4}$$

In (4), if we replace ϕ with the phase error that generates the maximum TVE value allowed and we consider power systems with nominal frequency of 50 Hz, then the maximum time error is $\Delta t = 31.8\mu s$. If the power system works with 60 Hz as nominal frequency then the maximum time error is $\Delta t = 26.5\mu s$.

Today, GPS systems represent the most commonly adopted method to provide time synchronization with the PMU devices. In particular, they provide a accuracy of about 100 ns that satisfies the requirements described above.

4 GPS Spoofing Attack and Detection Techniques

In section 3 we have argued about the importance of the PMUs synchronization. We showed how even a small time error can cause loss of synchronization. For this reason, PMU devices rely on the GPS receiver to obtain high accuracy. The GPS receiver is known to be vulnerable to GPS spoofing attacks. The goal of GPS spoofing attacks is to provide a forged version of the GPS signal to take control of a GPS receiver. So if an attacker successfully performs a GPS spoofing attack he/she can compromise the monitoring system of the power grid. This attack was discovered and highlighted in 2001 by U.S. Department of Transportation during a study performed on vulnerabilities of the transportation infrastructure that uses GPS signal [16].

The first step needed to perform a GPS spoofing attack is to acquire and to track the GPS signals to obtain a reference signal. Then a forged signal is generated and summed to the original GPS signal. The new signal is used to synchronize the spoofed signal with the authentic signal received. So the attacker produces a signal perfectly aligned with the authentic signals but with lower power. The generated spoofed signal is comparable to the noise of the target receiver in terms of power. Then the attacker increases the power of the forged signal until it overcomes the authentic signal. In this way, the forged signal shows higher Signal-to-Noise Ratio (SNR). So, the GPS receiver tracks the fake GPS signal (instead of the authentic signal) due to its higher SNR. After that, the attacker has successfully taken control of the GPS receiver. Then he/she slowly moves the spoofed signal from the authentic signal. The GPS signal received is considered to be completely captured when the spoofed signal is delayed by $2\mu s$ from the authentic signal as described in [17].

Thus the attacker could increase the time delay until it overcomes the 1% TVE as defined in the section 3.

Several techniques have been proposed in order to detect the GPS spoofing attack. These techniques are based on:

- monitoring the absolute GPS signal strength. This technique is based on comparisons between the observed signal strength and the expected signal strength. If their difference is greater than a fixed threshold, an alert is generated;
- monitoring the signal strength received from each satellite. The idea is to compare the observed signal strength with the expected signal strength for each satellite. The attacker will generate forged signal of equal strength for

each artificial satellite through the GPS satellite simulator. Instead, the signals provided by real satellites will change over time for each satellite. So an alert is generated if the signal characteristics are constant over time for each satellite;
- monitoring the relative GPS signal strength. This technique implies that the average signal strength is recorded and compared periodically. An alert is generated if a large change in relative signal strength is detected.

Further techniques that can be used to detect GPS spoofing attack are provided in [18] [19].

5 Network Time Protocol

Network Time Protocol (NTP) is widely used to synchronize system clocks among a set of distributed time servers and clients. NTP architecture is organized in layers, where synchronization flows from primary servers (higher layer) to the secondary servers and clients (lowest layer). The primary servers must be reliably synchronized to a GPS receiver and they must provide accurate and precise timestamps, even in case of a significant network jitter. Also the protocol must mitigate errors due to network disruptions or server failures. The synchronization process between a client and a server starts with a request from the client as shown in Figure 3. In particular the client sends current time T_1 to the

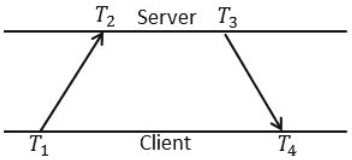


Fig. 3. NTP: client-server synchronization

server. The server saves this time T_1 together with the current time T_2. Then the server sends the client the current time T_3 together with the saved times T_1 and T_2. When the client receives the message, it reads its time T_4 and computes two values: offset of the clock α and round-trip delay β related to the server. The offset is computed as: $\alpha = \frac{1}{2}[(T_2 - T_1) + (T_3 - T_4)]$ whereas the round-trip delay is computed as: $\beta = (T_4 - T_1) - (T_3 - T_2)$. Both values α and β are minimized by a clock filter algorithm to obtain the synchronization of the client. More details about NTP are provided in [20] [21].

6 Anti GPS Spoofing Architecture

In this section we describe our architecture for remediation when a GPS spoofing attack is successfully performed. The proposed architecture is shown in Figure 4.

The idea is to increase the resilience to attacks using the concepts of redundancy and diversity. In fact, GPS spoofing attack can succeed because each PMU uses the timestamps provided by a unique time source i.e. GPS receiver. Also GPS receiver is a single point of failure for each PMU. Our approach is to use multiple time sources to provide timestamps to PMUs. To identify which technology can be used as a backup time source for PMUs, we analyzed PMUs requirements in terms of maximum time delay to avoid synchronization losses. Also we analyzed the accuracy provided by the GPS receiver so that the backup time source can provide coherent timestamps. We selected NTP as a technology that satisfies both requirements of maximum allowed delay by PMUs and expected accuracy by GPS receiver. The resulting NTP precision depends on the communication network behavior.

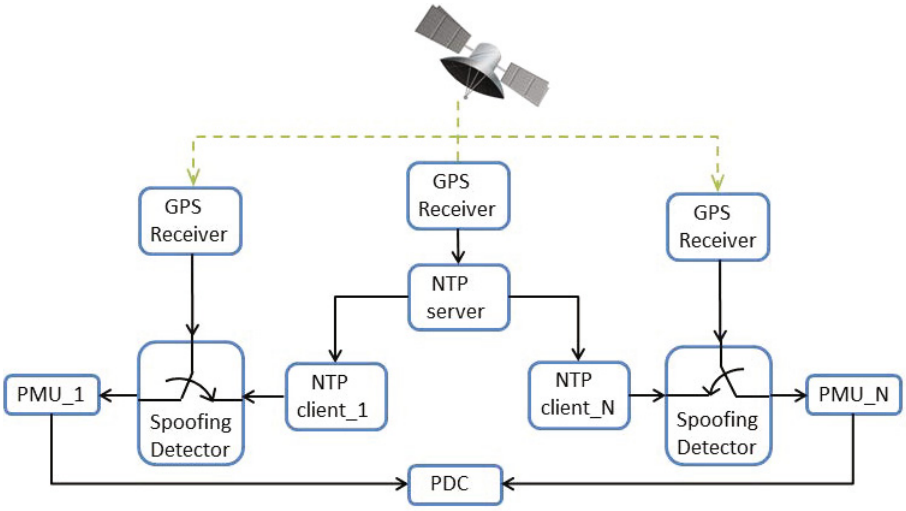


Fig. 4. GPS Spoofing Remediation Architecture

In the following, we assume that an NTP server is available and provides synchronization to many clients. The NTP server in Figure 4 is synchronized through a GPS receiver. NTP clients are synchronized with the correct time provided by NTP server. In our architecture we can see that PMU devices are not directly connected to the GPS receiver as in the standard monitoring model shown in Figure 1, but they are linked to the Spoofing Detector component. Of course the redundancy with two time sources is not sufficient to ensure intrusion tolerance. In fact to perform a voting with quorum at least three time sources are required. However we obtained the same results through only two time sources i.e. GPS Receiver and NTP client, because the Spoofing Detector uses a technique that allows to detect clock anomalous behaviours. In fact, Spoofing Detector component implements one of the techniques described in Section 4 to detect the GPS spoofing attack. The Spoofing Detector listens to the signal provided by the GPS receiver in order to recognize the characteristics

of the GPS spoofing attack. When an anomalous condition according to the chosen detection technique is found, then the remediation technique is activated. In particular, the Spoofing Detector component performs a switch of the timing source from GPS receiver to the NTP client. The NTP client replaces the compromised GPS receiver. In this way, PMUs always use a correct time source while measuring synchrophasors. NTP client is warm component i.e. it is always enabled and synchronized with NTP server.

The PDC component monitors the difference between the phases provided by different PMUs. Also the PDC uses information provided by Spoofing Detectors to avoid the generation of false alerts when an unexpected delay occurs during the synchronization of NTP clients.

In the proposed architecture, each PMU and NTP server uses a different GPS receiver. Also the GPS receiver of the NTP server is located very far from other GPS receivers. Then, when a spoofed GPS signal is propagated to compromise a GPS receiver belonging to a PMU device, we assume that the spoofed signal does not affect the GPS receiver of the NTP server.

7 Implementation Details

As we have shown in section 3, PMUs require strong time constraints to avoid synchronization losses. Our architecture works correctly when NTP protocol provides the same or better accuracy compared to GPS receiver. To obtain the maximum accuracy in NTP, modifications to operating system kernel are required. In particular the clock discipline algorithm in the synchronization daemon must be replaced with a module that offers the same functionality and operates in the kernel module. The clock discipline algorithm is the algorithm used to adjust the system clock in accordance with a final offset. While clock corrections are performed once per second in the classic synchronization daemon, they are performed every tick interrupt in the kernel. Using a specific implementation of NTP, it is possible to get an accuracy of the order of nanoseconds when an accurate reference clock is available [22] [23].

We use a dedicated network to reduce the network communication delay. Also all clients are linked to the primary NTP server to obtain a better accuracy.

Spoofing Detector component implements one of the techniques mentioned in section 4 to detect the GPS spoofing attack. The implemented technique is based on monitoring relative GPS signal strength. In particular, initially the component records a valid signal for a defined time interval. Then, it calculates the average value of the recorded signal and stores it. Finally, the alert threshold must be chosen. The choice of the threshold is very important because if it is set too low the component will perform many wrong switches of the chosen reference source; at the other side, if the threshold is too high the attack detection will be slow. Since the maximum time error allowed for PMUs to avoid the synchronization loss is about $20\mu s$, then a slow detection can compromise the synchronization of PMUs.

The component was developed in C++ in order to obtain good performances.

8 Attack Model and Experimental Results

In the first experiment we used two PMUs with a power grid that works at 60 Hz. The GPS receiver of a PMU is compromised by a spoofing attack. Instead, the GPS receiver of the other PMU works correctly and we use it as reference. In Figure 5 we show the attack model. In Figure 6 we show the difference

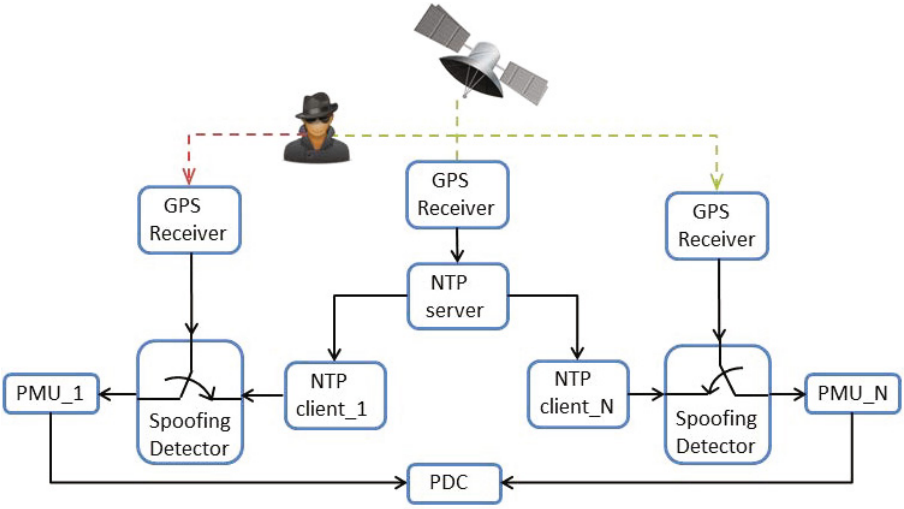


Fig. 5. Attack model used to compromise a GPS receiver of the PMU

between the phase angle measured by the compromised PMU and the phase angle measured by the reference PMU over time. Before the attack occurs, the synchrophasors measured by the two PMUs are overlapping vectors. This means that the phase angles of the two PMUs are aligned, so their difference is equal to zero.

At time $t = 300$ seconds the attacker tries to compromise the GPS receiver of a PMU through a spoofing attack. We have done many experiments in order to find the time required to perform a spoofing attack. In particular, in our case the time required is 2 minutes and 20 seconds. This time is in agreement also with another study [24], where the authors claim that about 2 minutes are needed to perform a spoofing attack. So at time $t = 440$ seconds the GPS receiver is completely compromised and the error time introduced for perfect synchronization with the attacker is $2\mu s$. The relation between phase angle and time error is provided by (4).

From now on, the purpose of the attacker is to introduce a higher error in the phase angle of the synchrophasors generated by the compromised PMU to overcome the maximum time error allowed before losing the synchronization. To achieve this goal, the attacker provides a constant acceleration of $3\frac{m}{s^2}$. A higher acceleration could desynchronize the GPS receiver from the attacker. We can see

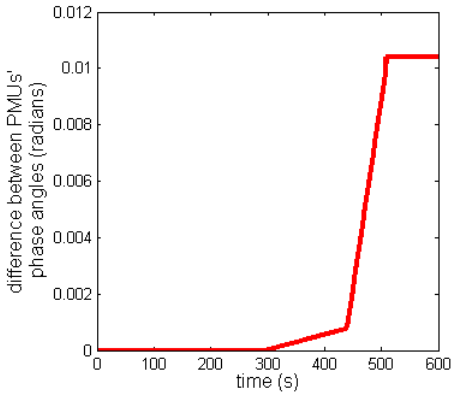


Fig. 6. Difference between PMUs' phase angles when one of them is compromised

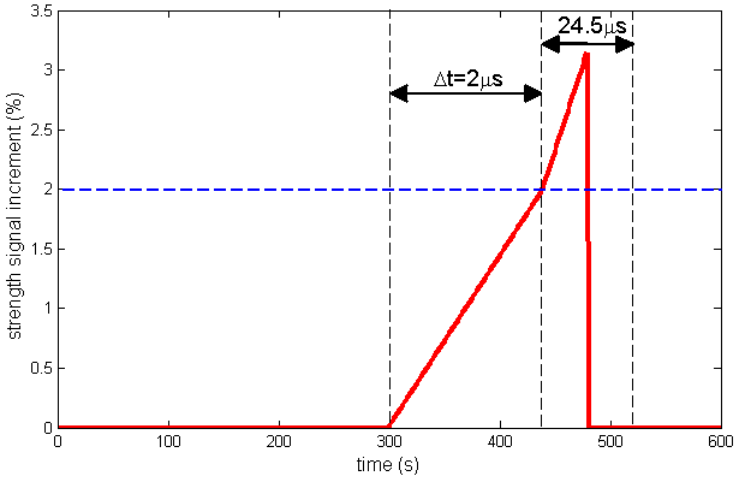


Fig. 7. Performance of the proposed resilient architecture

that at time $t = 470$ seconds the time error introduced is about $12.65\mu s$ and the difference of phase angles is about 0.0048 radiants. As described in section 3, the maximum tolerated error phase before the loss of synchronization is 0.01 radiants so, no loss of synchronization is occurring yet. At time $t = 510$ seconds, the difference between phase angles is greater than the maximum error phase allowed to obtain a TVE value under 1%. So the synchronization of the PMU attacked is lost. In Figure 7 we show the performance of our resilient architecture. In particular, we show the detection and remediation activities performed when spoofing attack occurs. At the time when the signal strength measured by Spoofing Detector overcomes 2% (blue line) of the average signal strength, our resilient architecture performs a change of time source. The threshold that allows the remediation to be activated was estimated through experimental tests. So at time $t = 440$ seconds the architecture has successfully detected the attack.

The detection delay causes a time error of $2\mu s$. At this time the attacker increases the acceleration of $3\frac{m}{s^2}$ to quickly reach the desynchronization from the correct time source. It is possible to use the available error time (i.e. $24.5\mu s$) to activate the remediation before the loss of synchronization. During the change of the timing source from GPS receiver to NTP client, the strength of the spoofed signal grows because the attacker tries to complete the spoofing attack. After the reference source time is changed, then the correct time is again provided to the PMU by the NTP client.

9 Conclusions

In this paper we discussed the usage of WAMS in smart grid. WAMSs use measurements of different PMUs to obtain information about power grid status. The comparison between measurements provided by different PMUs are useful if referred to the same time. So all PMU devices use a unique reference clock provided by GPS receivers. GPS receivers are vulnerable to GPS spoofing attacks. We reviewed several techniques to detect this type of attack. Also we presented a new resilient architecture that implements one of the proposed techniques to detect the spoofing attack. Also, the architecture implements a remediation technique when the attack is detected. The remediation technique is based on the use of the synchronization protocol NTP. When the attack is detected the time source switches from GPS receiver to NTP client. Experimental tests show the effectiveness of our solution.

In the future we plan to improve the detection time of spoofing attack. In fact, we think that times provided by NTP client and GPS receiver can be used together, to reduce detection latency. However the Spoofing Detector component of the proposed architecture could become more complex. We will perform other experimental tests in order to evaluate the impact of a greater complexity of the Spoofing Detector component on the reaction time of the architecture.

Acknowledgments. This work has been partially supported by the TENACE PRIN Project (n. 20103P34XC) funded by the Italian Ministry of Education, University and Research.

The research leading to these results has received funding from the European Commission within the context of the Seventh Framework Programme (FP7/2007-2013) under Grant Agreement No. 257644 (MAnagement of Security information and events in Service Infrastructures, MASSIF Project).

References

1. Romero, J.J.: Blackouts illuminate india's power problems. IEEE Spectrum 49(10), 11–12 (2012), doi:10.1109/MSPEC.2012.6309237
2. Singh, A., Aasma, S.: Grid failure in Northern, Eastern and North-Eastern grid in 2012: Cause & its effect on economy of India An Review. SAMRIDDHI-A Journal of Physical Sciences, Engineering and Technology (S-JPSET 2012) 3(2) (2012) ISSN: 2229-7111

3. FERC/NERC Staff Report on the September 8, 2011 Blackout. Arizona-Sourthem California Outages on September 8, 2011 (April 2012)
4. Final report on the august 14, 2003 blackout in the united states and canada: Causes and recommendations. Technical report, U.S.-Canada Power System Outage Task Force (April 2004)
5. Larsson, S., Danell, A.: The black-out in southern Sweden and eastern Denmark, September 23, 2003. In: 2006 IEEE PES Power Systems Conference and Exposition, PSCE 2006, October 29-November 1, pp. 309–313 (2003), doi:10.1109/PSCE.2006.296324
6. Oral, B., Donmez, F.: The Impacts of Natural Disasters on Power Systems: Anatomy of the Marmara Earthquake Blackout. Acta Polytechnica Hungarica (2010)
7. D'Antonio, S., Coppolino, L., Elia, I.A., Formicola, V.: Security issues of a phasor data concentrator for smart grid infrastructure. In: Proceedings of the 13th European Workshop on Dependable Computing, EWDC 2011, pp. 3–8. ACM, New York (2011), doi:10.1145/1978582.1978584
8. Coppolino, L., D'Antonio, S., Elia, I.A., Romano, L.: Security analysis of smart grid data collection technologies. In: Flammini, F., Bologna, S., Vittorini, V. (eds.) SAFECOMP 2011. LNCS, vol. 6894, pp. 143–156. Springer, Heidelberg (2011)
9. Coppolino, L., D'Antonio, S., Esposito, M., Romano, L.: Exploiting diversity and correlation to improve the performance of intrusion detection systems. In: International Conference on Network and Service Security, N2S 2009, June 24-26, pp. 1–5 (2009)
10. Momoh, J.: Smart Grid: Fundamentals of Design and Analysis. IEEE Press Series on Power Engineering, vol. 63. Wiley (2012)
11. Doberstein, D.: Fundamentals of GPS Receivers. Springer ISBN 978-1-4614-0409-5
12. Hadley, M.D., McBride, J.B., Edgar, T.W., O'Neil, L.R., Johnson, J.D.: Securing Wide Area Measurement Systems. U.S. Department of Energy, Office of Electricity Delivery and Energy Reliability (June 2007)
13. Humphreys, T.: Statement on the vulnerability of civil unmanned aerial vehicles and other systems to civil GPS spoofing, submitted to the Subcommittee on Oversight, Investigations and Management of the House Committee on Homeland Security, U.S. House of Representatives, Washington, DC (July18, 2012), `http://homeland.house.gov/sites/homeland.house.gov/files/Testimony-Humphreys.pdf`
14. Warner, J.S.: GPS Spoofing Countermeasures. Appeared in Homeland Security Journal (December 12, 2003)
15. IEEE Standard C37.118-2005: IEEE Standard for Synchrophasors for Power Systems (2006)
16. Volpe National Transportation Systems Center, Vulnerability Assessment of the Transportation Infrastructure Relying on the Global Positioning System, U.S. Department of Transportation Research and Innovative Technology Administration, Cambridge, Massachusetts (2001), `http://www.navcen.uscg.gov/pdf/vulnerability_assess_2001.pdf`
17. Shepard, D.P., Humphreys, T.E., Fansler, A.A.: Evaluation of the vulnerability of phasor measurement units to GPS spoofing attacks. International Journal of Critical Infrastructure Protection 5(3-4), 146–153 (2012)
18. Warner, J.S., Johnston, R.G.: GPS Spoofing Countermeasures. Vulnerability Assessment Team Los Alamos National Laboratory Los Alamos, New Mexico, 87545
19. Humphreys, T.: Statement on the vunerability of civil unmanned aerial vehicle and other systems to civil GPS spoofing. University of Texas at Austin (July 18, 2012)

20. Hinden, R., Deering, S.: Internet Protocol Version 6 (IPv6) Addressing Architecture, Network Working Group report RFC-3513. Nokia, 26 p. (April 2003)
21. Partridge, C., Mendez, T., Milliken, T.: Host Anycasting Service, Network Working Group report RFC-1536, Bolt Beranek Newman, 9 p. (November 1992)
22. Mills, D.L.: Unix kernel modifications for precision time synchronization, Electrical Engineering Department Report 94-10-1, University of Delaware, 24 p. (October 1994)
23. Mills, D.L.: Adaptive hybrid clock discipline algorithm for the Network Time Protocol. IEEE/ACM Trans. Networking 6(5), 505–514 (1998)
24. Warner, J.S., Johnston, R.G.: A simple demonstration that the global positioning system (GPS) is vulnerable to spoofing. Journal of Security Administration (2002)

A Dependable Alternative to the Spanning Tree Protocol

João Lopes[1], Susana Sargento[2,3], and André Zúquete[2,3,4]

[1] Vodafone Portugal, Lisboa, Portugal
joao.lopes@vodafone.com
[2] Dep. of Electronics, Telecommunications and Informatics, Univ. of Aveiro, Portugal
{susana.sargento,andre.zuquete}@ua.pt
[3] Instituto de Telecomunicações, Aveiro, Portugal
[4] Institute of Electronics and Telematics Engineering of Aveiro, Portugal

Abstract. The Spanning Tree Protocol (STP) is known to have stability problems and poor convergence intervals. Several protocols and variants exist targeting the replacement of STP variants, most of them proprietary and with limited scope of operation. The recent protocols, IETF TRILL and IEEE SPB, target mainly data center networks, are based on complex concepts, require great processing power from switches and huge investment in new gear.

In this paper we propose Self-Configurable Switches Protocol (SCS) as an alternative to all these protocols. It has the following advantages: it is configuration-free, thus less vulnerable to human mistakes; it enhances the network stability and performance when comparing with STP; and it is suitable to the range of equipment and networks that typically run STP variants, minimizing the need for potential large investments required by TRILL and SPB. This paper describes the main characteristics, processes and mechanisms of SCS, presents some lab and simulation experiments with STP and SCS, and provides demonstrations that SCS provides a more reliable service than STP variants, and a more cost effective alternative to TRILL and SPB network dependability.

Keywords: Layer 2 networks, dependability, Spanning Tree Protocol, TRILL, SPB, 802.1aq, Self-Configurable Switches.

1 Introduction

The Spanning Tree Protocol (STP) [14] is a protocol widely used in switched networks. Its goal is to define a loop-free path (a tree) for distributing traffic within a mesh of switches. For preventing loops, STP allows switched networks to be fully meshed, but effectively it only explores inter-switch links that are part of the current active spanning tree; the other links remain inactive until being included in the forwarding path of the spanning tree when necessary.

STP presents three issues regarding dependability. The first is the instability created while a spanning tree is being calculated, which can be endless due to critical race conditions. The second is the fact that proper configuration is critical

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 148–164, 2013.

to leverage STP with an admissible behavior (see example below). The third is that traffic paths along the switched network do not make use of the redundancy allowed by the meshed architecture, neither take the best paths towards the destinations; instead, they disable all the redundant links and follow a single traffic path between all switches. The consequences of this are threefold: (i) traffic is likely not to follow the shortest path from the source to the destination; (ii) hosts may experience congestion due to the overload of the spanning tree; and (iii) spanning tree reconvergence always induces service disruption.

Just to illustrate the risks posed by STP we give an example (extracted from [5]). In November 13th, 2002, the Beth Israel Deaconess Medical Centre (BIDMC) network, running STP, crashed repeatedly over four days, forcing the hospital to revert back processes thirty years, into 1970's paper systems. All redundancy in power supply, servers and data storage systems was useless, since the core network was not functional at all. After several dramatic days, it was found that the network problem was originated by the violation of STP hop count limit. In consequence, BIDMC spent near three million dollars to redesign and replace its entire network. This 2002 example sounds like a history from a long time ago; however, and unfortunately, it keeps pretty actual nowadays and all the major efforts done to compensate STP/RSTP/MSTP flaws aren't sufficient for current network stability, feasibility and availability requirements.

The fundamental motivation for our work was to create an alternative to control Ethernet networks enabling them to get self-configured, i.e. without the need for any human configuration, neither all the underlying complexity and investments required by TRILL [13] or SPB [1]. Additionally, we also add the motivation to enable the network's internal configuration to evolve seamlessly when switches or links are added or removed, thus breaking with the STP paradigm that can lead to network forwarding outages due to dramatic spanning tree reconfigurations. And, finally, a third motivation for our work was to improve the exploitation of meshed switches by distributing the traffic load among all possible links, instead of concentrating traffic in a single spanning tree.

Our contribution is a replacement for STP (and its variants), denoted by Self-Configurable Switches (**SCS**). SCS involves a very limited and risk-free human intervention and does its entire configuration autonomously. Unlike STP, where switches are labeled with critical priority values, and thus are not all equal (for creating hierarchies from which spanning trees are derived), in SCS the switches are all equal, in the sense that no switch has more responsibilities than others.

Each SCS switch builds its own view of the network topology, and uses it to manage how multicast and broadcast traffic is propagated throughout the network. SCS uses a gossip-based protocol to spread topology information along the network, but without the requirement of achieving a single, uniform set of information in all switches.

SCS prevents loops in traffic by using a controlled flooding strategy. Flooding traffic is unicasted to switches and then, locally, flooded to end-hosts (i.e. anything other than an SCS switch). For unicasting flooding traffic among switches we use Delegation Tables and a novel Ethernet packet (unicast flood packet,

UFP). Delegation Tables indicate whether a switch should act as a forwarder of an UFP from a switch towards other switches. The UFP differs from the standard Ethernet packet by carrying two extra fields: time to live (**TTL**) and unicast flooding source switch (**UFS**). TTL enables a last resort mechanism to prevent problems created by loops; it should be seldom used. UFS enables switches to reason about how to conduct the dissemination of a received UFP using the local Delegation Tables.

SCS was evaluated through simulation with NS-3 and complemented with STP experiments in a lab environment. We created a set of scenarios where STP is known to have problems, and we confirmed those problems in a laboratory infrastructure running several STP variants. We ran NS-3 simulations with those same scenarios to observe the behavior of SCS and to confirm its advantage over STP. In all cases, the SCS network evolved automatically, quickly and seamlessly to a stable configuration, while with STP variants we observed most of the expected problems.

This paper is organized as follows. Section 2 presents background information about STP, its variants and the recent IEEE and IETF proposals to replace them. The SCS protocol is described in Section 3 and Section 4 presents SCS implementation and evaluation. Section 5 exposes SCS dependability evaluation and finally Section 6 presents the conclusions and the ideas for future work.

2 Related Work

The objective of STP is to build a loop-free Layer 2 topology, sourced at a switch (Root Bridge) and spanning to all other network switches. The operation is quite simple, and STP only has to determine if the switch port should be forwarding or blocking frames towards it in order to prevent loops. Currently, STP entails many design and performance issues to be used in production networks.

Rapid Spanning Tree Protocol (RSTP) [2] supersedes STP mainly to optimize convergence times and increase stability. However, Myers *et al.* [12] showed that RSTP is not scalable and when critical nodes like Root Bridges fail, convergence times are far away from the ones advertised (if the network manages to converge at all). Generally, RSTP fast convergence in the order of milliseconds depends on several factors, like the type of ports, failure location, network topology and switch role. Furthermore, in some situations RSTP is permeable to race conditions and exhibits the classic *count-to-infinity* behavior [12]. Several authors addressed recently this issue – RSTP with Epochs [8,9], Reliable RSTP [3], Delay RSTP [4] – but none was adopted by standards.

Multiple Spanning Trees protocol (MSTP) allows groups of VLANs to be mapped into different spanning trees and the coexistence of multiple, independent, spanning tree instances to run simultaneously for those groups, and keeping the same issues as RSTP.

All STP variants concentrate unicast traffic in the current spanning tree, because the switches' forwarding tables are originally fed with flooding traffic. Thus, STP creates an unbalanced link utilization, wasting useful network

resources (links between switches) and reducing aggregate bandwidth by forcing all host-to-host paths onto one tree [15]. Complementary, hosts' traffic gets more vulnerable to flooding scenarios, caused by malicious or faulty applications/systems, than it would be if the network could maximize the exploitation of different paths along a Layer 2 network.

Being somehow complex and processor intensive, IETF's TRILL [13] radically changes the Layer 2 switching concept, "routerializing" switches. Using the Intermediate System-to-Intermediate System (IS-IS or ISIS) [7,6] link state routing protocol, it creates large groups of links that appear to IP devices like single links. Inheriting ISIS equal-cost multipath capability and optimal path calculations, TRILL supports load balancing and optimal path forwarding for unicast traffic. However, for multicast and broadcast traffic it uses a configurable number of distribution trees, which requires the problematic election and maintenance of multi-destination trees.

802.1aq Shortest Path Bridging (SPB) is the IEEE's proposal to replace STP, RSTP and MSTP, and a standard since March 2012 [1]. It also uses the ISIS to discover and advertise network topology and compute shortest path from all switches in the SPB network [10]. SPB is very similar to TRILL on many factors. Hence, their ISIS inheritance allows fast convergence, equal-cost load balancing and shortest paths calculations. However, they have completely different forwarding paradigms: TRILL acts like a Layer 3 protocol, rewriting the Layer 2 address hop-by-hop, whereas SPB behaves like a regular Layer 2 protocol, keeping frames unchanged until getting to their destination. Both are complex and processor intensive and require ISIS expertise to manage the network. Also, the size of forwarding databases will pull up switches' resource demands.

To overcome the issues of previous approaches, we propose SCS that is a lightweight protocol, with low requirements in terms of administration (reducing the impact of human errors), and providing resilience, load balancing and stability. The similarities between SCS and both TRILL and SPB are marginal. TRILL uses only TTL mechanisms for preventing flooding traffic to loop forever; SCS will only use TTL as a last resort mechanism for the same goal (in normal circumstances, it should not be necessary at all). Each SPB switch uses the knowledge of the complete network topology, and computes from it many multi-destination trees, to decide if a flooding packet is to be accepted or dropped (Reverse Path Forwarding Check, RPFC). SCS uses much less topological information (mainly the set of reachable switches) and some delegation rules per switch to take forwarding decisions about flooding traffic.

3 Contribution: Self-Configurable Switches

The SCS paradigm is considerably different from STP. STP aims to create a unique, loop-free topology within the network, breaking all redundant paths. By the contrary, SCS assumes the network as looped and works over it using as much as equal cost paths as possible in order to maximize the redundancy provided and the investment made. Furthermore, SCS empowers an Ethernet switch with

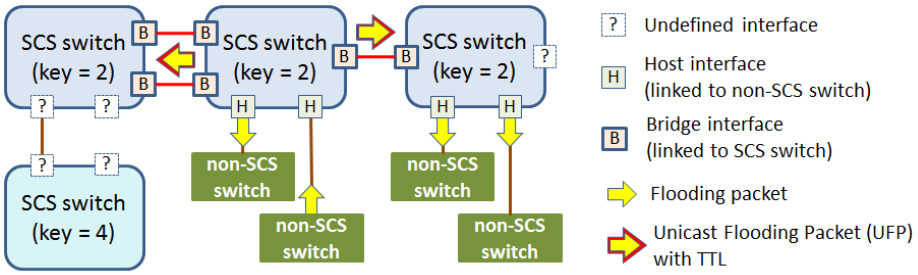


Fig. 1. Overview of SCS main characteristics. The 4 SCS switches form two different SCS domains: one with key 4, another with key 2. SCS switches handle interfaces as: undefined (disconnected or connected to an SCS switch with a different key), host (connected to anything but an SCS switch) and bridge (connected to an SCS switch with the same key). Flooding packets from hosts are flooded to all hosts in each SCS switch and unicasted encapsulated in UFPs between SCS switches. In case of redundancy, any of the links between a pair of switches can be used for transmitting a UFP, but only one each time.

self-configuring capabilities, mainly to deal with multicast, broadcast and unicast traffic that must be transferred throughout all switch ports, but also to handle topology changes that might occur in the network.

3.1 SCS Main Characteristics

This section briefly presents the main concepts, mechanisms and features that are assumed by SCS. Some of these concepts are graphically presented in Figure 1.

(Almost-)Zero configurations: Only a non-secret, 6-bit key is required to be configured on SCS switches – neighborship key. This key, shared by all switches running under a common administration, identifies the ones that are allowed to establish neighbor relationships between them, creating an SCS domain. If this key mismatches, the SCS switches will not establish a neighborship (see Figure 1). All other processes and mechanisms inherent to SCS are self-configurable and self-assessed by the switch itself.

Two interface types: SCS classifies the interfaces by the type of network device attached to it (see Figure1). Interfaces connecting SCS switches belonging to the same administration domain are automatically classified as bridge interfaces. Interfaces connecting SCS switches belonging to other administration domains remain virtually disconnected (thus undefined), for safety reasons. Any other type of network device that connects to an SCS switch uses host interfaces, even legacy STP switches. This interface classification is very important to deal with the traffic that needs to be flooded throughout the network, as SCS methods are quite different for host and bridge interfaces, as it will be explained later.

Switches mainly remain as switches: SCS does not change the way current switches transparently forward traffic to known destinations. However, the way

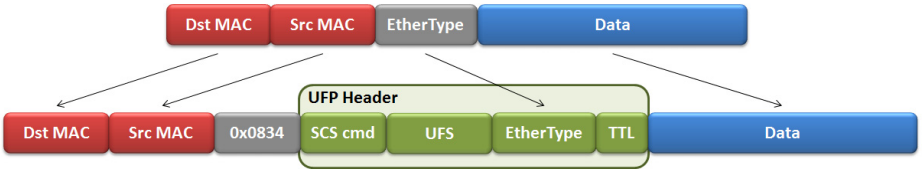


Fig. 2. Format of an UFP (below), a particular type of SCS frame, encapsulating an original Ethernet frame (top)

it deals with traffic to unknown destinations is radically different on bridge interfaces; for host interfaces, switches' behavior remains unchanged.

Regular switches act as follows regarding unicast traffic:

1. If targeting a known destination, then it is forwarded through a known port.
2. Otherwise, it is flooded throughout all ports, except the incoming one.

What SCS does differently is the way flooded traffic is forwarded, breaking up the second rule, which relates with broadcast, multicast and unknown destination unicast traffic. SCS changes this second rule to the two following ones:

2a. Traffic targeting an unknown destination is flooded throughout all local host interfaces, except the incoming one.
2b. This traffic is also encapsulated in unicast flooding packets (UFPs, see Figure 2) and unicasted towards all neighbor SCS switches (via bridge interfaces). Neighbors will then continue the flooding, following rule 2a and partially this one.

This SCS behavior removes all flooded traffic from inter-switch links, managing efficiently the traffic flooding within looped topologies: it unicasts flooding packets towards all neighbor switches and asks some of them to forward those packets onwards.

Flooding propagation protection: A loop in an Ethernet network, if not properly tackled, may cause flooded traffic to be indefinitely forwarded between switches and raise serious connectivity problems. SCS protects against such events by using two complementary mechanisms within UFPs (see Figure 2):

1. They contain an indication about the SCS switch responsible for its creation (unicast flooding source switch, **UFS**). UFS helps UFP receivers to take the appropriate decisions about its forwarding to other switches or if the UFP effectively looped.
2. They contain a TTL counter to limit an occasional traffic replication over a network loop. We anticipate that such loops can occur by accident during forwarding adaptations caused by topology changes. However, with a stable network configuration, TTL's are not required to prevent propagation loops.

SCS frames' format: SCS protocol messages are encapsulated in the payload of Ethernet frames, using a new EtherType: `0x0834` (see Figure 2).

Optimized frame forwarding: SCS optimizes traffic flow as it favors redundant paths, potentiating traffic load balancing. SCS load balancing is not a typical routing protocol load balancing scheme, where traffic can flow via equal cost redundant paths (some even over unequal cost). In turn, SCS allows load balance of traffic in terms of link occupation, i.e., as SCS does not block any link, they are all available to be used. That way, for certain destinations, some links will be used, while for other, different links will be selected. This brings extra advantages for network resource utilization, as traffic flow in a stable network is deterministic, like in STP, but now through all available links on the network. For parallel links, the load balance scheme is simplified, as the SCS switch will replicate all entries known through a link in its forwarding table to all other parallel links to the same neighbor, allowing traffic load balancing in a round-robin fashion over those parallel links without the need for any link aggregation protocol or technique.

Inverted flooding: Ethernet switches promiscuously listen the incoming traffic, remember the source address of that traffic, and forward traffic based on that learning method. If the destination is unknown, the switch floods the traffic throughout all ports, except the incoming one, even for unicast traffic: this is known as *unknown unicast flooding*. This flooding is fundamental to find the right path to forward packets along the Ethernet network, and in principle it should be re-executed upon a topology change. Instead, under the same circumstances SCS switches proactively inform the entire network of possible forwarding path changes by advertising the MAC address of all host interfaces, allowing all neighbor switches to quickly update their forwarding tables accordingly.

3.2 SCS Processes and Protocols

SCS Protocol comprises four distinct internal processes to control both *data* and *control* planes. For data plane, it uses the *Forwarding process* and the data plane component of the *Flooding process*. For control plane, SCS protocol uses *Neighborship process*, *Topology process* and the control plane component of the *Flooding process* – the flooding process has both data and control components.

Forwarding process: This process defines the way frames are forwarded within an SCS network. As previously stated, SCS does not change the overall transparent switching characteristic of a regular switch, so the switch forwards traffic as before. The major difference over today switches' forwarding table is the optimization that allows the presence of several entries to the same MAC Address, allowing traffic load balance over parallel links towards the same neighbor.

Neighborship process: This process is a control plane mechanism and it is the basis of the SCS Protocol. It classifies the neighborship into three states: *up*, *down* and *delayUp*. In the *up* state, the switch processes and forwards traffic from/to that particular neighbor. In the *down* state, the switch does not accept neither forward frames from/to that neighbor. The *delayUp* state is just a transition state between down and up states that protects SCS networks from events due to flapping links.

Three mechanisms control the entire course of the Neighborship process: *Neighbor Auto-discovery Mechanism*, *Relationship Mechanism* and *Purging Mechanism*. The *Neighbor Auto-discovery Mechanism* is responsible for discovery and establish of neighbor relationships between adjacent SCS switches, via the exchange of `SCS_Hello` messages. The *Relationship Mechanism* manages the neighbor relationship between adjacent SCS switches, defining the state maintenance or transition through the observation of periodic transmission of `SCS_Hello` messages, or the lack of it. The *Purging Mechanism*, for conditions intrinsically associated with physical connectivity losses, is accountable for neighbor relationship removals. All three mechanisms are also responsible for triggering events on other SCS processes, accordingly to the events/state transitions observed.

Topology process: The Topology process is another fundamental control plane engine. Basically, it provides an SCS switch with the knowledge of all other SCS switches running on campus, besides its own adjacent neighbors, and the cost to reach all them. The core function of the Topology Process relies on the *Topology Change Mechanism*, which uses dedicated messages (`SCS_Update` packets) to exchange information between neighbor switches. `SCS_Update` control frames are triggered whenever a switch needs to advertise a change towards all other switches in the network: install, clear or query a topology record or install or remove a delegation record. The two last ones will be described in the control plane component of the Flooding Process.

Flooding process: The Flooding process comprises both data and control plane components. The data plane component of the Flooding Process defines the way an SCS switch forwards traffic that needs to be flooded throughout the network, whereas the control plane component defines the flooding control mechanism to avoid unnecessary data replication and looped frames within the network.

The Flooding process data plane defines how to forward frames with unknown destination address, specifically broadcast, multicast and unicast frames for which the switch does not have an entry on its forwarding table and, consequently, does not know where to forward them. SCS protocol differentiates the flooding method by the type of the incoming port interface: host or bridge:

Host: Upon receiving a flooding frame, the switch will forward the frame throughout all other host ports; for all bridge interfaces, which connect adjacent SCS neighbors, the switch must create an UFP containing the unknown destination traffic and unicast them via the bridge interface.

Bridge: The switch is actually receiving an UFP from a neighbor. It will forward the original Ethernet frame (encapsulated in the UFP) throughout all host interfaces and unicast the UFP via the bridge interfaces for which it is delegated. The concept of delegation will be clarified below. UFPs contain a TTL field (just to circumscribe eventual loop occurrence) that is decremented on each switch hopping.

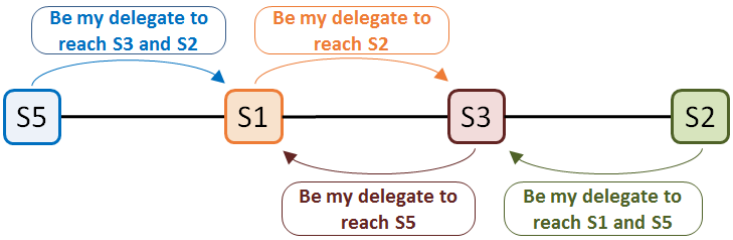


Fig. 3. SCS Delegation example. An UFP created by switch S5 will be transmitted to S1, in order to reach S1, S3 and S2. S1, being a delegate of S5 for S3 and S2, sends the UFP to S3, in order to reach S3 and S2. Finally, S3, being a delegate of S1 for S2, sends the UFP to S2. Along this path the UFP remains unchanged, apart from a TLL decrement.

> The UFP encapsulation is performed in a way that allows switches to update their forwarding tables with the source MAC address of the original Ethernet frame. For unicast traffic this enables a major performance improvement, as it reduces the occurrence of flooding caused by unknown destinations.

The Flooding process control plane defines all the procedures that allow switches to know how to process received flooded traffic, both locally via host interfaces and remotely via bridge interfaces. Among several loop avoidance and forwarding mechanisms, the control plane component of the Flooding Process relies mainly on a mechanism called *SCS Delegation* to deal with traffic that needs to be flooded.

The SCS Delegation mechanism concerns with the optimized path calculation to all known destinations behind non-adjacent switches. Delegation involves managing two tables, *Flood Table* and *Delegation Table*. For each frame flooded throughout the network, the original switch will use its Flood Table and all participating switches will use their own Delegation Table. Delegations can be progressive, meaning a switch S1 may be delegate of S5 towards some destination S2, but S1 can also delegate to another switch S3 the responsibility to reach S2. Figure 3 illustrates this delegation example.

3.3 Reconfiguration Actions

Network reconfigurations take place when a link is added or removed. In the next paragraphs we will briefly describe what happens when a single link between a pair of switches is added or removed.

When a new link is added, a new neighborship is created and the Neighborship table[1] of both switches is updated. Next, the Topology Change mechanism takes place where each switch imports the Topology Table of the new neighbor,

[1] This table is a special part of the Topology Table, containing only entries with metric 1 and link status information.

updates its Topology Table and flags those changes to the other directly attached neighbors. As a new topology exists, all other switches will compare the new paths with their own information and if any change is necessary they will update their Topology Tables and inform their own other neighbors, about that change. Also, if needed, their Flood Tables are also updated. In case where a Flood Table is updated, those changes are flagged to the corresponding neighbors in order for them to update their Delegation Tables accordingly.

This recursive, multi-branch update process terminates gradually when each switch receives a useless topology update, which is discarded. Hopefully, the network was reconfigured to a topology that provides the shortest path between switches (but not necessarily the most efficient one, because the load of the paths' links is not considered).

In the case of a link removal, a neighborship is removed. Therefore, each directly involved switch updates its Neighborship Table and starts the Topology Change mechanism: update the Topology Table (remove all direct or indirect entries using the interface of the removed link, inform neighbors about those changes, change Flood Tables, if needed, and, if so, flag their neighbors to change their Delegation Tables. In the event of the link removal causing a switch S_x to lose a path towards a particular switch S_y, S_x will query their directly attached neighbors for a path to that particular S_y switch.

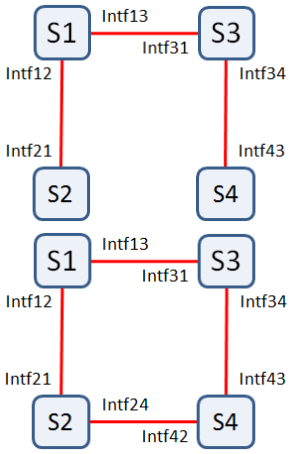
Recursively, the other switches, upon receiving topology updates and path queries, will start their own Topology Change mechanisms to update their tables with the most recent information received and then answer back with their best path to the queried destinations. At the end, the network was reconfigured to a topology that, once again, provides the shortest path between the switches.

3.4 Reconfiguration Example

In this section we will illustrate a network self-reconfiguration with an example. In this example we will consider a set of 4 switches where a link between two of them (S2 and S4) is created or destroyed (see Figure 4). As the network is symmetrical, what happens in S2 and S1 will happen as well in S3 and S4, respectively. Therefore, we will resume our explanation mainly to what happens in S1 and S2.

When the link is established, S2 receives the list of peers known by S4 and the metric towards them. Then, it updates its entry for S4 (Intf24, metric 1) and add a new path to S3 (via S4, metric 2). These updates are transmitted to the other neighbors (S1).

S1 already has a path with metric 2 to S4, via S3, therefore it adds a redundant entry to S4 via S2 (or Intf12). However, S1 already uses S3 as its delegate to reach S4, therefore it will not elect S2 as its delegate to reach S4 as well (otherwise, we would have more than one flooding message reaching S4, when starting in S1). Regarding the path to S3 via S2, S1 ignores it since it has a better path. Its redundant path to S4 via S2 is forwarded to all neighbors (S3), but S3 will ignore it for the same reason (already has a better path to S4). At this point, all updates in Topology Tables initiated by S2 have terminated.



	Topology Table			Flood Table	Delegation Table		
	Peer	Intf	Metric	Delegate to	Intf	Src	Flood to
S1	S2	12	1		12	S2	S3
	S3	13	1		**12**	**S2**	**S4**
	S4	13	2	S3	13	S3	S2
S2	S1	21	1				
	S3	21	2	S1			
	S4	**21**	**3**	**S1**			
S3	S1	31	1		31	S1	S4
	S2	31	2	S1	34	S4	S1
	S4	34	1		**34**	**S4**	**S2**
S1	S2	12	1		12	S2	S3
	S3	13	1				
	S4	13	2	S3	13	S3	S2
	S4	**12**	**2**				
S2	S1	21	1				
	S3	21	2	S1			
	S4	**24**	**1**				
	S3	**24**	**2**				
S3	S1	31	1		31	S1	S4
	S2	31	2	S1	34	S4	S1
	S4	34	1				
	S2	**34**	**2**				

Fig. 4. Reconfiguration of switches' tables (Topology, Flood and Delegation) when a link between S2 and S4 is added or removed. Entries that do not appear or are different in both scenarios are highlighted. For simplicity sake, the tables of S4 where omitted.

S2 also knows that it was using S1 as a delegate to forward flooding traffic to S3 and S4, but the later is no longer needed, because now it is neighbor of S4. Therefore, it removes the delegation to S4 from S1, keeping only the delegation for S3. This delegation is maintained, because the path from S2 to S3 via S4 is not shorter than the existing one (via S1).

When the link between S2 and S4 is removed, S2 removes from its topology entry all entries regarding interface Intf24. Consequently, it loses indirect connectivity with S4, keeping indirect connectivity with S3 through S1. S2 informs S1 about these changes (deleted path to S4, with metric 1, and path to S3, with metric 2). Finally, as S2 loses a path towards S4, it also asks its neighbors (S1) for a path to S4.

Regarding S2's topology update, S1 ignores updates regarding S3 (it has a better path) and uses the updates about S4 to remove its path to S4 through Inft12. It also queries its Flood Table and sees that no change is needed (none in this case, as its uses S3, and not S2, to flood traffic to S4). Finally, it forwards its topology update (removed path to S4 with metric 2) to all neighbors (S3). Since S3 has a direct link to S4, the update is discarded.

When S1 receives S2's request for a path to S4, it replies its best path towards S4 (via S3, metric 2). S2 updates its Topology and Flood Tables accordingly (Intf 21, metric 3, flooding via S1), and signals S1 that for now on it will be S2's delegate to flood towards S4. Accordingly, S1 creates a new entry in its Delegation Table in order to allow traffic sourced in S2 to reach S4.

This example shows that SCS does not have to keep, on each switch, the full network topology, as TRILL and SPB do (provided by ISIS). Furthermore,

in SCS we don't need to compute multi-destination distribution trees on each switch (TRILL computes at least one global tree, SPB computes one per each possible source switch for detecting loops with RPFC, see Section 2).

4 Implementation and Evaluation

The SCS feasibility to succeed STP variants was assessed using SCS simulated on NS-3[2] and the STP in real laboratory setups, for the same topologies. This section presents a brief explanation on how SCS was adapted to be simulated on NS-3 and some results of the assessment tests.

4.1 SCS Implementation in NS-3

SCS code was adapted to be simulated over NS-3 (version 3.13), in order to test the SCS algorithm and evaluate its potential and performance over different network topologies. Overall, NS-3 simulation models for wired networks are quite realistic; therefore, it is our conviction that it was a good choice to assess and evaluate SCS.

The SCS switch abstraction in NS-3 was accomplished by attaching the existing `BridgeNetDevice` to a *redesigned* `Node` enhanced with the SCS protocol, which works at both control and data plane. All the SCS intelligence, methods, tables, headers, frames and algorithms were developed from scratch within the `Node` class.

4.2 Assessment Tests

We focused our assessment tests in scenarios that are known to create problems to the original STP, or the successor RSTP or even to the improved MSTP. SCS handled all those scenarios without any problem, presenting a superior response in all of them [11]. To assess macroscopically SCS activities we simulated a host continuous initiating, with a 1 second pace, a request/reply dialog with another host, and we provoked some change in the topology between them that caused a network reconfiguration. Here is a summary of part of the results assessed[3], for a total of 8 network scenarios with no more than 15 switches each:

- Networks with a large[4] diameter, conservative STP timers. The observed STP convergence time varied between 30 to 50 seconds, while with SCS it varied between 0 to 3 seconds.
- Convergence black holes. Single link failure and restore, in STP, created black holes (total absence of communications) for 30 seconds. SCS was mainly unaffected, taken 3 seconds to converge in the worst-case scenario.

[2] `http://www.nsnam.org`

[3] Other results, such as the immunity to STP issues when an SCS network is used by STP networks on its edge, where omitted for simplicity sake.

[4] Close to the maximum recommended diameter of 7 for STP.

- Count-to-infinity. With RSTP or MSTP, stale Bridge Protocol Data Units for a faulty Root Bridge might persist in the network. SCS converged without problems.
- Topological limitations. STP variants are not suitable for all network topologies. In ring topologies, for example, RSTP is unable to converge in a timely manner, if it converges at all. The assessed SCS convergence is significantly better, always within the range 0-3 seconds.
- Unknown unicast flooding storms. All STP variants are vulnerable to this phenomenon when the topology changes and forwarding tables need to be flushed and rebuild. SCS avoids this problem by keeping forwarding tables and rebuilt them quickly with the inverted flooding process.
- Least cost path. All STP variants create a single forwarding path (the spanning tree), concentrating traffic on that single path. Therefore, they transform a well-designed, high-performance and redundant network into a single communication path between all network switches, reducing all investment in bandwidth and redundancy to simple dormant links. SCS, in turn, leaves an highly redundant network as is, uses all available links and always creates optimal paths (in terms of number of hops), thus leveraging all the network's available bandwidth and redundant paths.

Wrapping up, in the tests conducted SCS outperformed STP variants in all aspects: minimum and non-critical human configuration (SCS keys); topology independence; fast and deterministic convergence upon network changes; optimal paths; and link load balance. From all these factors, we can conclude that SCS is better suited than STP variants to provide a more dependable Layer 2 network.

Finally, it is worth mention that we did not tested SCS against TRILL and SPB because we did not had any equipment with these protocols for running the same tests we did with STP variants.

5 Dependability Evaluation

In this section we will informally demonstrate that SCS has several properties in terms of dependability, which make it a good replacement for STP variants.

Claim: *The reconfiguration of network paths, upon adding or removing links between SCS switches, evolves seamlessly.*

Demonstration: In both cases, link addition or removal, the network gets gradually reconfigured, without the need to reconfigure all the entries in all the tables. The reconfiguration is a sort of "*update wave*" that starts in the switches that detected the network change, and goes from switch to switch, recursively following the links of their neighbors, until reaching switches where nothing changes, and where a particular update branch terminates. At the end, the entire network is aware about the presence of a new switch or aware of the absence of an existing switch, or even about novel paths to reach a particular switch.

This initial update wave is followed by a distributed multitude of smaller scale update waves: the updates of Delegation Tables in neighbor switches. These do not propagate to other switches and in many situations may not happen at all.

In both update phases, no global agreement has to be reached, and the system does not need to stop in the meanwhile. The reconfiguration takes place seamlessly: when a link is added, the service is always assured and improved as the update goes; when a link is removed, some traffic may follow for a while a path to nowhere, but the network will eventually fix that situation in a short period of time.

***Claim**: SCS network reconfigurations converge to optimal states.*
Demonstration: Whenever there is a network topological change, SCS switches propagate it to adapt the tables they use to unicast and flooding traffic. Such adaptation happens only in one case: the information received is useful to provide a better service than before. If this is true, the adaptation happens and is propagated to neighbors; otherwise, it terminates. Therefore, assuming that network topological modifications do not happen continuously, network reconfigurations converge to an optimal state, in terms of minimum cost to flood traffic from each switch to all the other ones.

***Claim**: Network loops are expected.*
Demonstration: Being a loop preventive protocol, STP targets a loop-free network topology. So if STP fails to prevent a loop, how will the loop be open? The answer is: it will not. There are several enhancements to STP to do so, but most require human configuration. That way, not only STP networks are permeable to the occurrence of loops but also to human errors.

SCS, in turn, expects loops and consequently copes with them. SCS both prevents and detects looping frames, not topological loops. It prevents looping frames by controlling the flooded traffic via UFP encapsulations and delegations and by detecting abnormal situations using many built-in mechanisms; as a last resort, delegated traffic carry a TTL field. This topic will be addressed in the next claim.

***Claim**: SCS does not loop traffic forever.*
Demonstration: Inside an SCS network of switches, all flooding traffic travels encapsulated within UFPs. Therefore, we need to prove that an UFP cannot loop forever.

If the network loops an UFP, this means that an UFP will arrive to a switch that has previously handled it (déjà vu). If this is the originating switch (its MAC equals the UFS field of the UFP), then the UFP is discarded for breaking a potential loop. If it is not the originating switch, then two scenarios can be considered:

– The UFP is not arriving from the interface where the traffic from its UFS is supposed to arrive from. For instance, in the example of Figure 4, an UFP from S3 arrives at S1 from Intf12. In this case the packet is discarded for breaking a potential loop.

– The UFP arrives from the expected interface. Reasoning recursively, this is a strange situation that can only happen if that interface changed from the first time the packet has passed in the switch to this last one, possibly due to updates in the local Delegation Table. In this case, we cannot know if such update in fact happened, and therefore we forward the UFP as ordinarily, using the forwarding rules for its UFS. However, the UFP will not loop forever due to its TTL, which is decremented on each hop.

Concluding, we are able to detect and discard most UFPs that could create loops, and even in rare scenarios involving critical races between the update of Delegation Tables and the actual forwarding of UFPs, loops are prevented using UFPs TTL.

***Claim**: SCS networks are more protected against targeted menaces and much more resilient over external events.*
Demonstration: STP variants misuse network resources as a single distribution tree is calculated. That tree is easily identified and it is sourced on the STP Root Bridge. If the Root Bridge – or some of its links – is lost, the overall network stability is threatened due to the need of recalculating the entire tree. Moreover, the network availability is completely dependent on the location of switches in the tree, in case of inter-switch link failures.

SCS switches do not participate in any kind of Root Bridge election, neither build hierarchical topologies based on some switch being superior over others. SCS switches create neighborship relations with directly attached switches, one for each link between them, being then responsible for advertising to its neighbors all the paths it knows towards all other SCS switches in the network (that it knows about). With that information exchanged, they construct their own view of the network, creating a simple, local topology database.

In a well redundant network running SCS, link failures present low risk to the overall network service availability, as the probability of that link failure presenting a major topology change is extremely reduced. Moreover, switch failures have predictable effects on the network: they only affect the communication paths actually exploring the failing switches, unlike STP variants.

***Claim**: Unknown unicast storms are constrained and network convergence is enhanced.*
Demonstration: Unknown unicast flooding is an issue in STP networks, especially during topology changes. If an attacker manages to induce an STP network to be in topology change state, unicast traffic would start to be flooded and could be unduly sniffed. Conversely, under the same circumstances, SCS switches use the Inverted Flooding feature to proactively inform the entire network of possible forwarding path changes by advertising its attached host MAC addresses, allowing all neighbor switches to quickly update their forwarding tables accordingly. This way, network flooding is reduced significantly and overall network convergence is incredibly much faster.

6 Conclusions

The SCS protocol has a single objective: replace STP variants and, consequently, enhance the overall network dependability. SCS is well suited for this purpose, as it provides self-configuring capabilities, optimized forwarding paths for unicast and multi-destination traffic, high performance and traffic load sharing over redundant paths. During the SCS assessment tests, many experiments and simulations were performed to attest SCS behavior against STP, RSTP and MSTP, revisiting the problems those protocols face and observing how SCS overcomes all of them, one-by-one, providing at the same time all the aforementioned characteristics and proving its suitability to replace all STP variants. To better understand its limitations and challenges on real world networks, a physical prototype using OpenFlow[5] is currently under development.

As future work, and to further expand SCS functionality, there are four specific areas that could be explored: alternative routing metrics, traffic segregation, multicast and security. Currently SCS uses only a single metric for choosing broadcast traffic paths: the number of hops to target switches. In the future we want to improve its metrics in order to accommodate other factors, such as latency and bandwidth. Regarding traffic segregation, SCS might allow traffic distribution over redundant topologies creating multiple SCS domains, redundant of each other. VLAN topologies could be mapped into those SCS domains to create broadcast domain isolation and redundant topologies. Multicast traffic forwarding should be taken into consideration, in order to filter multicast traffic to only those switches and hosts which requested it. Security should be always taken into consideration and, currently, SCS does not target secure communications neither switches' protection.

References

1. IEEE Standard for Local and Metropolitan Area Networks: Bridges and Virtual Bridged Local Area Networks – Amendment 9: Shortest Path Bridging. P802.1aq/Draft D4.6 (C/LM) (March 2012)
2. IEEE Standard for Local and metropolitan area networks: Media Access Control (MAC) Bridges. IEEE Std 802.1D (June 2004)
3. Atif, S.M.: DRSTP: A Simple Technique for Preventing Count-to-Infinity in RSTP Controlled Switched Ethernet Networks. Int. Journal of Computer Networks 2(6), 278–296 (2011)
4. Atif, S.M.: RRSTP: A Spanning Tree Protocol for Obviating Count-to-Infinity from Switched Ethernet Networks. Int. Journal of Computer Networks 3(1), 17–36 (2011)
5. Barnard, A.: Got paper? Beth Israel Deaconess copes with a massive computer crash (November 26, 2002)
6. Callon, R.: Use of OSI IS-IS for routing in TCP/IP and dual environments. RFC 1195 (December 1990)

[5] `http://www.openflow.org`

7. Eastlake, D., Banerjee, A., Dutt, D., Perlman, R., Ghanwani, A.: Transparent Interconnection of Lots of Links (TRILL) Use of IS-IS. RFC 6326 (July 2011)
8. Elmeleegy, K., Cox, A.L., Ng, T.S.E.: On Count-to-Infinity Induced Forwarding Loops Ethernet Networks. In: Proc. of the 25th IEEE Int. Conf. on Computer Communications, INFOCOM 2006, pp. 1–13 (April 2006)
9. Elmeleegy, K., Cox, A.L., Ng, T.S.E.: Understanding and Mitigating the Effects of Count to Infinity in Ethernet Networks. IEEE/ACM Transactions on Networking 17(1), 186–199 (2009)
10. Fedyk, D., Ashwood-Smith, P., Allan, D., Bragg, A., Unbehagen, P.: IS-IS Extensions Supporting IEEE 802.1aq Shortest Path Bridging. RFC 6329 (April 2012)
11. Lopes, J.P.P.O.: Self-Configurable Switches. Master's thesis, University of Aveiro, Aveiro, Portugal (July 2012)
12. Myers, A., Ng, T.S.E., Zhang, H.: Rethinking the Service Model: Scaling Ethernet to a Million Nodes. In: Third Works. on Hot Topics in Networks (HotNets-III), San Diego, CA, USA (November 2004)
13. Perlman, R., Eastlake III, D., Dutt, D., Gai, S., Ghanwani, A.: Routing Bridges (RBridges): Base Protocol Specification. RFC 6325 (July 2011)
14. Perlman, R.: An algorithm for distributed computation of a spanning tree in an extended LAN. SIGCOMM Computer Communication Review 15(4), 44–53 (1985)
15. Touch, J., Perlman, R.: Transparent Interconnection of Lots of Links (TRILL): Problem and Applicability Statement. RFC 5556 (May 2009)

Understanding (Mis)Information Spreading for Improving Corporate Network Trustworthiness

Roberto Baldoni, Silvia Bonomi, Giuseppe Antonio Di Luna,
Luca Montanari, and Mara Sorella

Cyber Intelligence and Information Security Research Center and
Department of Computer, Control, and Management Engineering Antonio Ruberti
"Sapienza" University of Rome, Italy
{baldoni,bonomi,diluna,montanari,sorella}@dis.uniroma1.it
http://www.cis.sapienza.it

Abstract. The explosion of social networks is pervading every form of business. When used inside corporate networks, they can create potential vulnerabilities as employees at the lower levels in the organization chart may become influential thanks to social connections. This unexpected influence could be dangerous if the employee behaves maliciously reducing thus the trustworthiness of the overall organization. The paper is a first attempt in understanding this phenomenon by proposing a model for corporate networks that is able to measure the influence of each employee on the overall organizational chart, that is, to which extent an employee is able to spread (mis)information through the corporate network. The evaluation is done considering the Enron case.

Keywords: Corporate Social Network, Influential Entities, Trustworthiness, Misinformation Spreading, Complex Systems.

1 Introduction

Increasing the dependability and the security of a *system* has been one of the main challenges in the last three decades for a large community of researchers [1]; verification, testing, software injection have been some of the means used to achieve such an increment. Along the years, the meaning of *system* has radically changed. We passed from single to networked computing systems, from *Supervisory Control And Data Acquisition* systems (SCADA) to cloud systems etc. In some of such systems humans are part of it. For example a SCADA operator is responsible for applying pre-agreed rules and operations in response to some event appeared on a SCADA console. The paper considers a system being a group of employees belonging to the same company and structured according to an organization chart. Employees do not follow any specified rules when receiving information and they exchange information following two patterns: (i) formal communication going from a chief to his/her staff and (ii) informal communication moving among people without any role distinction (for example during a coffee-break). The latter form of communication has exploded in the recent

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 165–172, 2013.

years thanks to the diffusion of online social networks (starting from e-mails to the more recent Facebook, LinkedIn etc.). Their role is gaining even more importance as they may be used as tools for improving the efficiency of a large company, e.g., to find employees with required expertise for executing a certain task [2].

The paper presents a first attempt to model corporate networks encompassing hierarchical and social network relationships. Thanks to this model, we were able to identify unexpected influential employees and measure their effectiveness, i.e., their ability to reach (directly or indirectly) through their connections most of the people in the company even though, according to their ranking in the chart, they are not assumed to influence such large group of people. These persons are potential vulnerabilities for the company as they could be exploited as vectors (voluntary or not) by competitors for intelligence operations (with possible information leakage) and for misinformation campaigns [3]. All these facts turn out into a decrease of trustworthiness of the overall corporate network. Thus, understanding such vulnerabilities can improve the dependability of the overall organization seen as a single integrated system.

The paper then focuses on (mis)information spreading by identifying, based on a real data set from Enron scandal [4], which are such unexpected influential employees. We present results for two different instances of corporate networks: the first allows only top-down communications on the organization chart, the second one considers that a unit head can be influenced with some probability by its staff.

The rest of the paper is structured as follows: Section 2 presents the related work, Section 3 introduces a social corporate networking model. Then results on the (mis)information spreading are discussed in Section 4 and Section 5 concludes the paper.

2 Related Work

In the context of information propagation in social networks, two distinct issues have recently gained particular attention: *influence propagation* (and consequently influence maximization) and *misinformation spreading containment*.

Informally, the influence maximization problem can be defined as the problem of finding the set of the most "influential" individuals that, if selected to be the early adopters of an information, can maximize the number of users in the social network that will adopt the same information later on. The influence propagation problem has been studied for the first time by Domingos and Richardson in [5]. The authors formalized the problem as an optimization problem and they applied probabilistic data mining techniques to analyze the diffusion processes in the context of viral marketing.

Kempe et al. in [6] provided a different formalization in the form of a discrete algorithmic optimization problem, in which they add a constraint on the amount of resources that can be used to spread the information. In particular, they provide two models, namely *Independent Cascade Model* (ICM) and *Linear Threshold Model* (LTM), for the spreading of influence and proved that the

k-budgeted version of the influence maximization problem is NP-Hard but admits an approximation algorithm for both models. Bharathi et al. [7] extended ICM to address the possibility of concurrent competing campaigns (spreading good and bad information) and they modeled the problem as a two-player game. Budak et al. in [3] considered the strategy of using a good information dissemination campaign to fight against the propagation of misinformation started from a single adversary by individuating the Eventual Influence Limitation problem. They described two variants of ICM, proved it is NP-hard and provided a greedy algorithm suffering, of scalability problems for large networks. To address such point, several heuristics have been defined; examples are *degree centrality* [6,3], *Page Rank* [8] and *degree discount* [9]. The probabilistic diffusion of information could resemble the topic of information dissemination by means of gossip-based protocols in large scale systems [10], [11] gossiping algorithms are designed in settings where processors follow a protocol in order to maximize the diffusion of information minimizing the worst case performance in terms of messages exchanged or communication rounds.

3 Corporate Networks

We first introduce the model of corporate networks and recall the one of online social networks [3]. For each model, we describe the relationship graph and the corresponding diffusion model (i.e. node behavior). Then we merge them to create the corporate social network model.

Corporate Network model. Hierarchical relationships among entities in a company can be represented by a directed weighted graph $H(V, E_h, w_h)$ where V represents the set of employees, each edge $(u, v) \in E_h$ is a directed link from the node u to v and it may represent the relation *"u is chief of v"* or *"v is member of u's staff"*. Concerning the weight of edges, the following rules hold:

- $w_h(u, v) = 1$ for any edge $(u, v) \in E_h$ representing the relation *"u is chief of v"*. This means that a node that receives an information from a superior will surely assume that information.
- $w_h(u, v) = P < 1$ for any edge $(u, v) \in E_h$ representing the relation *"v is member of u's staff"*. This means that v that receives an information from u will assume that information with probability P.

As information diffusion mechanism we adopt the *Independent Cascade Model* (ICM). According to this model, each entity can be in *active* or *inactive* state. Initially a set $A_0 \subseteq V$ of vertices is active. The process unfolds in discrete steps during which every active vertex tries to activate every inactive neighbour. When a vertex u is firstly activated, it is given a single chance to activate each currently inactive neighbor v and it succeeds according to a given probability $p_{u,v}$ independently of the previous history. In case of success, v will become active in step $t + 1$; in any case, u cannot make any further attempts to activate v in subsequent rounds. The process runs until no more activations are possible.

When considering ICM applied to a corporate network graph, the edge weights correspond to the activation probabilities.

On-line Social Network Model. As in [3], we represent an On-line social Network (OSN) as an edge-weighted directed graph $S = (V, E_s, w_s)$ where the set of vertices is represented by the set of employees belonging to the social network and the set of edges is represented by the social relationships among employees (i.e. given two vertices u and $v \in V$ representing users in the social network, there exists an edge $(u, v) \in E_s$, going from u to v, if and only if user u has a social relation with v). For each edge $(u, v) \in E_s$, the weight $w_s(u, v)$ represents the influence that u has on v. Also in this case, the diffusion model considered is ICM.
In the specific case of companies, social networks can be represented by corporate social network tools, internal messaging systems (e.g. e-mails, internal chat service) etc...

Corporate Social Network Model. Merging the two previous graphs for the same organization, we obtain the corporate social network graph, i.e., $HS = (V, E_{hs}, w_{hs})$ where V is the set of employees and there exists an edge between two nodes u and v in E_{hs} only if there is that edge either in E_h or in E_s. Concerning weights, we associate to each edge the weight it had originally either in E_h or in E_s. If ad edge uv was present in both graphs H and S, then we associate with uv the minimum between the sum of the original weights and 1. Nodes follow ICM to spread information.
Figure 1 depicts the union of two graphs H and S.

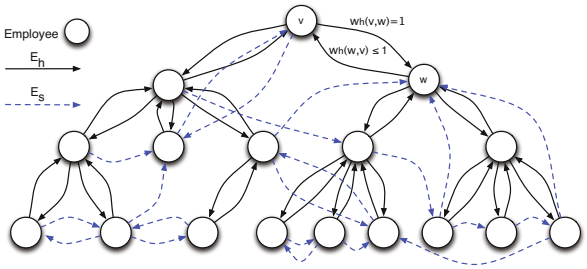


Fig. 1. Corporate social network graph highlighting the two graphs H and S

4 Identifying Influential Nodes

4.1 The Problem

In the context of corporate networks, employees with high impact in terms of information spreading over the company become of primary importance for an attacker (e.g., a competitor) because the latter may exploit such points of the

network to spread misinformation, malware, viruses or execute intelligence operations. Among those, there may exist a subset of entities particularly *weak*, in the sense that they may be bribed to inject bad information in the network (e.g. low-level employees).

As a consequence, it becomes extremely important for the administrators to identify such entities in order to monitor and avoid such kind of attack. We will formalize the *f-influencial Nodes Identification* problem as follows:
Given a direct weighted graph $G = (V, E, w)$ representing a network, identify the subset of nodes that, if injected with misinformation, will influence at least a fraction f of V.

4.2 Evaluation

For the evaluation of the *f-influencial Nodes Identification* problem, we consider the case of Enron corporation. Enron bankrupted in 2001 and thus the *Federal Energy Regulatory Commission*, during its investigation on the frauds occurred before the Enron crash, made available part of the email exchanges among 151 Enron employees containing 252,759 messages distributed in around 3000 user defined folders [4]. Thus from this dataset we derived the S graph with the rules described in Section 3.2 where the weight associated with the edge between two nodes u and v is determined by the number of e-mails sent by u to v normalized to represent the influence probability of u over v.

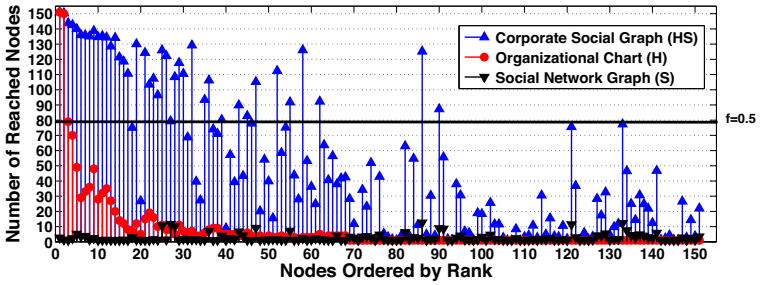
As far as the organizational chart of the Enron is concerned, no public full chart was ever disclosed, however we found two documents [12] and [13] that we used to reconstruct part of the original chart: one document contains a list of Enron employees with the positions covered inside the organization, the other contains a partial chart of the leading positions for the year 2000.

Based on this information, we derived an organizational chart for the Enron employees by building up a direct balanced tree resulting of height 8 and labeled trough a breadth-first-visit where approximately 60% of the nodes are leaves (from label 68 to 151).

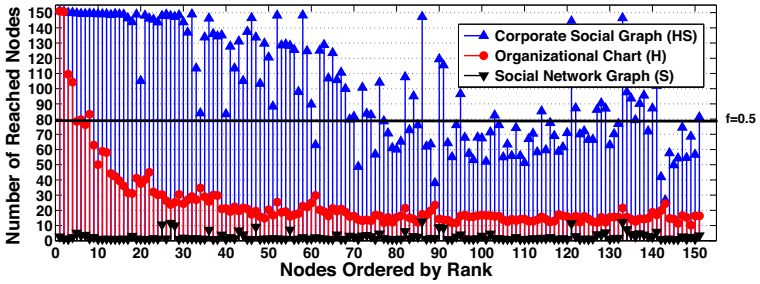
From the chart and applying the rules stated in Section 3.1, we derived the H graph. Finally, we obtained the HS graph using the rules of Section 3.3.

In order to discover the *f-influential weak nodes* we study the spread of information with 10000 Monte Carlo simulations from any single node in three different settings and the corresponding graphs H, S and HS. The same experiment is repeated by considering two different values of P: $P = 0$ and $P = 0.5$. Let us recall that P is the probability associated to edges (u, v) representing the *"u is member of v's staff"* relationship. P is the same for any of such edges to model the following cases:

- $P = 0$ corresponds to an organizational instance where supervisors don't listen at people in their staff.
- $P = 0.5$ models the situation in which a supervisor can either decide to accept or not an information coming from a person from his/her staff. The value of f in the experiments is set to 0.5 in order to discover which are the nodes that are able to contact the majority of the nodes.



(a) 0.45-influential Nodes evaluation for $P = 0$



(b) 0.45-influential Nodes evaluation for $P = 0.5$

Fig. 2. Number of nodes reached by each one of the 151 Enron employees (nodes ordered by rank) considering the influence given by the organizational chart (H); by the social network graph (S) and by the corporate social network graph (HS)

Figure 2(a) and Figure 2(b) show the results of the experiments. On the x axis we represent the employee positions in the organizational hierarchy in a non increasing order (on the left side there are the CEOs while, on the right, from id 68 on there are employees at the lowest-level of the tree); the y axis shows the amount of the spreading cascade achieved by the corresponding individual.

In both figures the results relative to the graph derived by the organization chart (H graph) show trivially that nodes that are located at the upper levels of the tree are the most influential, this influence tends to zero when the leaves of the tree are reached. The difference between results presented in Figure 2(a) and Figure 2(b) on H graphs depends on P. In Figure 2(b) each node in the organizational chart is able to influence its parent in the tree, thus allowing an overall greater influence.

In the social network graph (S graph) no sensible spreading cascades were observed, both because the probabilities over the edges are able to prevent the information from following long paths and because the average weighted out-degree of the nodes in the social network is relatively low.

What has been surprising from the study is that when considering the corporate social network graph (HS graph) shown in Figure 3, there are a number of

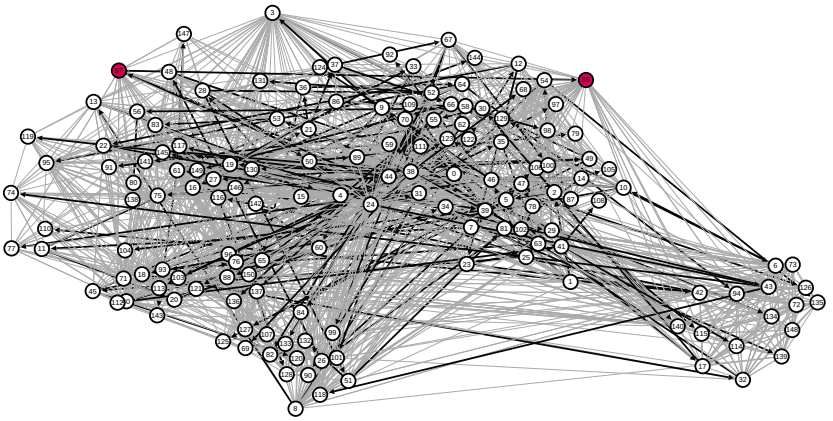


Fig. 3. The Enron HS graph. Black edges represent hierarchical relationships while gray edges represent social relationships.

unexpected nodes that are able to carry out an effective spread of information similar to the ones done by the nodes placed in the very upper layers of the organizational tree. This is due to the fact that such nodes exploit their (direct and transitive) social connections to inject information at higher levels of the hierarchy. As an example, Figure 2(a) reveals that node 57 and node 85 (red nodes in Figure 3) reach $f = 0.8$ i.e., the 80% of the total number of nodes and 5 employees cover more than half of nodes (i.e., $f \geq 0.5$). All these nodes belong to low levels of the organizational tree. Qualitatively, Figure 2(b) follows the same behavior, but the information spreading of each node is amplified thanks to the fact that $P > 0$ in the organization graph.

5 Concluding Remarks and Future Work

This paper considers the impact of the existence of social network-based communications inside a company and how they may impact the influence of employees.

In this paper, we introduced a model for corporate social networks that combines hierarchical and social connections. On this model we analyzed the practical problem of assessing the influence of each employee, by running some experiments working on the Enron dataset. From our preliminary analysis, we found that there exist several employees, at different points of the organization chart, that are able to reach a large percentage of people by exploiting both their social and hierarchal links. Such employees may represent a potential threat for the company as they could be exploited by a competitor to leak confidential information, to inject misinformation or to vector a virus. Thus the task of identifying and monitoring such "weak points" can be used by a chief security officer of the corporate network to increase the trustworthiness and the robustness of the organization with respect to insider threats.

As future works, we plan to address other relevant problems in the context of corporate networks. In particular, an interesting problem is to find clusters of employees in the corporate network (obtained by considering both the social and the hierarchical graphs), that are instead hidden when considering only the organization chart. Such clusters represent connections among employees that are not working together and may represent a vulnerability from the information confidentiality point of view; information may in fact be spread, trough the social links, to parts of the network that are not the supposed recipients.

Acknowledgements. This work has been partially supported by the Italian Ministry for Education, University, and Research (MIUR) in the framework of the Project of National Research Interest (PRIN) TENACE: Protecting National Critical Infrastructures from Cyber Threats.

References

1. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 1(1), 11–33 (2004)
2. IBM: Smallblue research tool, http://smallblue.research.ibm.com/
3. Budak, C., Agrawal, D., El Abbadi, A.: Limiting the spread of misinformation in social networks. In: Proceedings of the 20th International Conference on World Wide Web, WWW 2011, pp. 665–674 (2011)
4. http://www.isi.edu/~adibi/Enron/Enron.html (access: February 20, 2013)
5. Domingos, P., Richardson, M.: Mining the network value of customers. In: Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2001, pp. 57–66 (2001)
6. Kempe, D., Kleinberg, J., Tardos, E.: Maximizing the spread of influence through a social network. In: Proceedings of the 9th International Conference on Knowledge Discovery and Data Mining, KDD 2003, pp. 137–146 (2003)
7. Bharathi, S., Kempe, D., Salek, M.: Competitive influence maximization in social networks. In: Deng, X., Graham, F.C. (eds.) WINE 2007. LNCS, vol. 4858, pp. 306–311. Springer, Heidelberg (2007)
8. Luo, Z.-L., Cai, W.-D., Li, Y.-J., Peng, D.: A pageRank-based heuristic algorithm for influence maximization in the social network. In: Gaol, F.L. (ed.) Recent Progress in DEIT, Vol. 2. LNEE, vol. 157, pp. 485–490. Springer, Heidelberg (2012)
9. Chen, W., Wang, C., Wang, Y.: Scalable influence maximization for prevalent viral marketing in large-scale social networks. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2010, pp. 1029–1038. ACM, New York (2010)
10. Kermarrec, A.M., Van Steen, M.: Gossiping in distributed systems. ACM SIGOPS Operating Systems Review 41(5), 2–7 (2007)
11. Eugster, P.T., Guerraoui, R., Handurukande, S.B., Kouznetsov, P., Kermarrec, A.M.: Lightweight probabilistic broadcast. ACM TOCS 21(4), 341–374 (2003)
12. http://www.isi.edu/~adibi/Enron/Enron_Employee_Status.xls (access: February 20, 2013)
13. http://news.findlaw.com/hdocs/docs/enron/enronbk72803app-c.pdf (access: February 20, 2013)

Software Component Replication for Improved Fault-Tolerance: Can Multicore Processors Make It Work?*

João Soares, João Lourenço, and Nuno Preguiça

CITI/DI-FCT-Univ. Nova de Lisboa

Abstract. Programs increasingly rely on the use of complex component libraries, such as in-memory databases. As any other software, these libraries have bugs that may lead to the application failure. In this work we revisit the idea of software component replication for masking software bugs in the context of multi-core systems. We propose a new abstraction: a *Macro-Component*. A *Macro-Component* is a software component that includes several internal replicas with diverse implementations to detect and mask bugs. By relying on modern multicores processing capacity it is possible to execute the same operation in multiple replicas concurrently, thus incurring in minimal overhead. Also, by exploring the multiple existent implementations of well-known interfaces, it is possible to use the idea without incurring in additional development cost.

1 Introduction

Despite the large number of techniques developed for detecting and correcting software bugs during development and testing phases, software bugs remain a major problem in production releases [13,4]. Software updates or patches, designed to correct existing bugs often end up introducing new bugs - studies show that up to 70% of patches are buggy [22].

Multicore processors have made a push for increased concurrency in applications, leading to an increase of concurrency related bugs. This problem is being actively investigated, with a large number of works addressing the subject in the last few years [19,14,10,16,4,23].

The increasing complexity of software has led programmers to build their applications relying on the (re)use of third party off-the-shelf libraries and components, such as in-memory databases and XML parsers. If some of these components undergo systematic quality control procedures, others are provided by communities that cannot afford such procedures. If in the former case components already include bugs [8], we can expect the situation to be far worse in the latter case. Thus, these components become an important source of bugs for applications. The situation is magnified by the fact that application programmers have little or no control over these components.

* This work was partially support by FCT/MCT projects PEst-OE/EEI/UI0527/2011 and PTDC/EIA-EIA/108963/2008. João Soares was partially supported by FCT/MCTES research grant # SFRH/ BD/ 62306/ 2009.

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 173–180, 2013.

For dealing with faults caused by software bugs, several fault tolerance techniques have been proposed [17]. Some of these techniques improve software quality by relying on replication and redundancy techniques, normally combined with design diversity. The drawbacks of these approaches are an increase in development time and costs, since diverse solutions need to be designed, implemented and tested, and a compromise in application performance, due to result and state validation.

In this work we revisit the idea of software component replication for detecting and masking bugs, by proposing the *Macro-Component* abstraction. A *Macro-Component* is a software component that includes internally several diverse component replicas that implement the same interface. Assuming that different component replicas exhibit different bugs [3,6,7], by executing each operation in all replicas and comparing the obtained results, it is possible for a *Macro-Component* to detect and mask software bugs.

By exploring the power of multicore processors, the same operation can be concurrently executed in multiple replicas with minimal overhead. By exploring the multiple available implementations for the same standard interfaces, it is possible to create *Macro-Components* without incurring in additional development time or cost. This allows to put in practice the old idea of N-Version Programming [3] at the component level.

Although the idea seems simple, putting it to work involves a number of technical challenges that we explore in the remaining of this paper. In particular, we show how to minimize computational overhead by executing operations in as few replicas as possible and by minimizing the required number of coordination points among replicas. Our preliminary results suggest that this approach is promising, exhibiting acceptable performance. The results also show an important result for the practicality of the solution: the amount of memory used is not directly proportional to the number of replicas. The reason for this is that a large number of objects can be shared among the replicas - e.g. strings in database fields.

The remainder of this paper is organized as follows. The next section discusses related work. Section 3 introduces the *Macro-Component* model. Section 4 presents our current prototype, the implementation of Macro-Components for in-memory databases and presents some preliminary evaluation. Section 5 concludes the paper with some final remarks.

2 Related Work

Macro-Components share an identical model with n-Version Programming (NVP) [3,1]. Contrarily to the original NVP, *Macro-Components* work at the component granularity [18], taking advantage of third party components to minimize the impact in development time and costs. Additionally, unlike previous works, our design addresses modern multicore processors, which seem an suitable architecture to make software component replication work with minimum performance impact.

Fig. 1. Macro-Component

Replication has been a highly researched topic in distributed systems [9], with most of the proposed techniques addressing only fail stop faults. Byzantine fault tolerance techniques [12,20,2] have been proposed for dealing with other fault models. Some works, e.g. Eve [11], have been addressing replication in distributed settings with multicore machines. In our design, we re-use some of the ideas proposed in these works.

Most of the research on concurrency bugs has focused on techniques for finding and avoiding bugs [10,13,14,4,23]. Our work share some of the goals with these works, but differs from most of these approaches by relying on diverse replication.

Gashi et al. [6,7] have also focused on using third party components for improving fault tolerance in SQL Database Servers and Anti-Virus engines. Our proposal differs from these works, as it provides a generic framework and runtime support for producing fault tolerant components, based on diverse implementations of the same common interface.

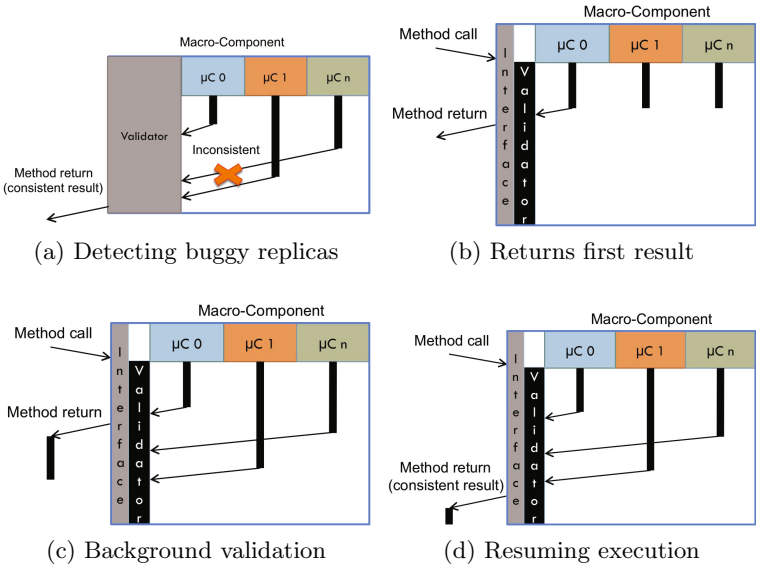
3 *Macro-Components*

A *Macro-Component* is a software component implemented using a set of diverse components (called replicas) that implement the same interface, as presented in figure 1. Diversity allows for each replica to have its own implementation while offering the same functionality and maintaining the same abstract state as all other replicas. This provides the means for *Macro-Components* to detect buggy behaviour of replicas, by identifying state or result divergences amongst replicas, thus preventing these bugs from being exposed and affecting the reliability of applications.

With this approach, an application can use a *Macro-Component* as it would use any other component. The only difference is that a *Macro-Component* has improved reliability. Thus, a single application may include a large number of *Macro-Components*.

Next, we detail how *Macro-Components* can be used to address several goals.

Detecting and Masking Bugs: Macro-Components, as NVP, follows the assumptions that different implementations incur in different bugs, and that a divergent result from the majority occurs due to the presence of bugs. Thus, to detect buggy behaviour, diversity in component replicas is crucial, and detection is achieved by comparing the results from the several replicas. Whenever a method is invoked on a *Macro-Component*, the following steps (illustrated in figure 2a)



(a) Detecting buggy replicas (b) Returns first result

(c) Background validation (d) Resuming execution

Fig. 2. *Macro-Component* Method Calls

occur: *i*) the corresponding method is invoked on all component replicas; *ii*) replicas execute the same method concurrently; *iii*) wait for $f + 1$, i.e., the majority, equal results from component replicas; and *iv*) return the result from the majority of replicas.

For keeping the overhead low, unlike solutions that validate both results and object state [24,21], *Macro-Components* detect buggy behaviour primarily by validating results, while object state is compared periodically in background. Whenever inconsistent results are detected, the corresponding faulty replicas are marked for recovery, and temporarily removed from the set of active replicas. Also, if some replica is unable to produce a result within a certain time limit, the replica is considered faulty and threads executing in the replica are aborted. The time limit is defined by the time taken by the majority of the replicas to reply plus an additional tolerance.

Detecting Concurrency Bugs: Macro-Components are not restricted, in any way, to the use of diverse replicas. When using homogenous replicas, Macro-Components can still be used for detecting and masking concurrency bugs.

To this end, the following approaches are possible. First, the imposed overhead due to the *Macro-Component* runtime may result in different inter-leavings of concurrent operations in different replicas. Second, it is possible to impose random delays on the method execution in different replicas, thus leading to different inter-leavings. Third, it is possible to execute method invocations sequentially in some of the replicas (as in [5]).

For minimizing the overhead, the latter two solutions can be used only when some problem is detected by running operations in the default mode. Additionally, sampling can be used when the latter two approaches are being used.

4 Implementation and Runtime

We are currently building a system for supporting applications that use Macro-Components, in Java. In this section, we present our current prototype.

A Macro-Component is composed by three main components: the *manager*, responsible for coordinating method execution on the replicas, the *validator*, responsible for validating the results returned by the replicas, and the *replicas*, the components responsible for maintaining the state. Applications remain oblivious to the replicated nature of Macro-Components since it offers a single copy view of the underlying state. To this end, each replica maintains an associated *version*, that registers the number of updates performed on the replica. The *manager* guarantees that operations execute on replicas in the same state, i.e., with the same version. This version is kept in shared memory, as an atomic *counter*.

The supporting runtime guarantees that when a method call is performed on a Macro-Component, the equivalent method is concurrently executed in all replicas. To this end, method calls are recorded as tasks and queued for execution. These tasks represent the method to be executed, and the replica in which the method is to be executed on. For each replica, an associated thread is responsible for dequeuing assigned tasks and execute them on that replica. Our current prototype currently supports concurrent execution only for operations that do not modify the state of the Macro-Component - operations that modify the state of the Macro-Component are currently executed serially.

This decouples the execution of the callee from the method, i.e., the thread calling the method can be different from the thread that executes it. This allows Macro-Components to provide independent execution models, allowing methods to execute asynchronously from the application threads.

We currently support two execution model. The first, based on non-transparent speculation of results, provides improved performance. The second, based on a prior verification of execution correctness, allows for transparent replacement of components by their Macro-Components siblings.

In the speculative execution mode, a Macro-Component returns the result from the fastest replica (figure 2b), while validating the result on background (figure 2c). If the result is found to be incorrect, the execution must be cancelled and re-started with the correct value (figure 2d). This approach can even improve the performance over standard components, when there are no faults, by exploring the differences in performances for the different replicas.

We currently do not support automatic transparent speculation. Thus, the Macro-Component notifies an error on a previous call when some method is called or when the application queries the Macro-Component for errors. This requires the application to be modified to include support for such calls. In general, this is not too complex as the verification calls can be added in the end of methods that use Macro-Components (or before some externalization of results is done).

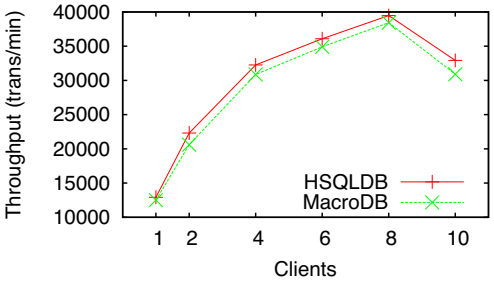


Fig. 3. TPC-C results

	Replicas		
MacroDB	2	3	4
HSQL	1.64×	2.29×	2.46×

Fig. 4. Memory overhead

In the prior-verification model, the results of a method is only returned when a majority of the replicas has returned the same result.

4.1 Database Macro-Component

We now describe the design and implementation of *MacroDB*, a *Macro-Component* for in-memory database systems. *MacroDB* is composed by a set of database replicas, each potentially supported by a different in-memory database engine. Applications remain oblivious of the replicated nature of the system since it offers them a standard JDBC interface, and standard transaction isolation levels.

Applications do not communicate directly with the database engines, instead they communicate with the *manager*, a JDBC compliant front-end which coordinates client operations in the underlying replicas. The *manager* receives statements from clients and forwards them, without modification, to the replicas, guaranteeing their ordered execution by the runtime support system. For statements inside a transaction, the first result is returned to the application while it is compared in background with the results from other replicas. Additionally, (read-only) queries execute initially only on $f + 1$ replicas (with f the number of replicas that can be faulty) - the queries are only executed in other replicas if returned results differ. When the application wants to commit a transaction, if there has been any error detected on the previously returned results, the commit fails. Otherwise, the commit executes is all replicas and returns to the application after it is confirmed. This approach combines the speculative and prior-verification execution models in a way that is transparent to database applications.

MacroDB is still under active development, missing the code for verifying state divergence and the wrappers to support the small differences in multiple database engines [20]. Even though, our solution is already operational with an homogeneous configuration, with all replicas running the same database engine. To provide an approximate value on the overhead that our runtime incurs for providing fault-tolerance, we ran the TPC-C benchmark on the Macro-Component, and compared the obtained results against the standalone database version. In all cases, the HSQLDB in-memory database was used. The results, presented on figure 3, show a small overhead for *MacroDB* version, averaging a 4% decrease in performance.

Although these are preliminary results, we expect that relying on multiple database engines can improve this overhead, as the result from the fastest replica will be returned in each case. On the other hand, the performance will be penalized by checking for the difference in the database state.

As an additional test, we also measured the memory overhead imposed by replicating the database. Contrarily to what was expected, the memory overhead is not proportional to the number of replicas, as presented in figure 4. This is due to the fact that replicas share immutable Java objects, such as Strings. The obtained results show that, a MacroDB configured with HSQL replicas, uses at most *2.5 times* more memory than the standalone engine, when using a 4 replica configuration. This makes deploying MacroDB practical on single machine multicores, even with large numbers of replicas.

5 Final Remarks

In this paper we revisited software component replication techniques, presenting a new abstraction for improving software fault tolerance, called *Macro-Component*. *Macro-Components* put in practice the old idea of N-Version Programming [3] at the component level. Existing software products can benefit from improved fault tolerance, simply exchanging components by their *Macro-Component* siblings, preventing developers from rewriting code, and preserving development methodologies.

We have presented the design of a system that supports the use of *Macro-Components* in applications. Our design focused on keeping the overhead low, by minimizing the overhead of computational resources by executing operations in as few replicas as possible and by minimizing the required number of coordination points among replicas. Our preliminary results suggest that this approach is promising, exhibiting acceptable performance. The results also show an important result for the practicality of the solution: the amount of memory used is not directly proportional to the number of replicas.

Our current prototype still misses some important features, namely result and state comparison, and improved recovery of replicas. We are currently conducting additional experiments to evaluate our prototype with standard benchmarks.

References

1. Avizienis, A.: The n-version approach to fault-tolerant software. IEEE Trans. Softw. Eng. 11(12), 1491–1501 (1985)
2. Bessani, A., Correia, M., Quaresma, B., André, F., Sousa, P.: Depsky: dependable and secure storage in a cloud-of-clouds. In: EuroSys 2011, pp. 31–46 (2011)
3. Chen, L., Avizienis, A.: N-version programming: A fault-tolerance approach to reliability of software operation. In: Proc. FTCS-8, pp. 3–9 (1978)
4. Fonseca, P., Li, C., Singhal, V., Rodrigues, R.: A study of the internal and external effects of concurrency bugs. In: DSN 2010, pp. 221–230 (2010)
5. Fonseca, P., Li, C., Rodrigues, R.: Finding complex concurrency bugs in large multi-threaded applications. In: EuroSys 2011 (2011)

6. Gashi, I., Popov, P., Stankovic, V., Strigini, L.: On designing dependable services with diverse off-the-shelf SQL servers. In: de Lemos, R., Gacek, C., Romanovsky, A. (eds.) Architecting Dependable Systems II. LNCS, vol. 3069, pp. 191–214. Springer, Heidelberg (2004)
7. Gashi, I., Stankovic, V., Leita, C., Thonnard, O.: An experimental study of diversity with off-the-shelf antivirus engines. In: NCA 2009, pp. 4–11 (2009)
8. Ghosh, S., Kelly, J.L.: Bytecode fault injection for java software. Journal of Systems and Software 81(11), 2034–2043 (2008)
9. Helal, A.A., Bhargava, B.K., Heddaya, A.A.: Replication techniques in distributed systems. Kluwer Academic Publishers (1996)
10. Jula, H., Tralamazza, D., Zamfir, C., Candea, G.: Deadlock immunity: Enabling systems to defend against deadlocks. In: OSDI 2008 (2008)
11. Kapritsos, M., Wang, Y., Quema, V., Clement, A., Alvisi, L., Dahlin, M.: All about eve: execute-verify replication for multi-core servers. In: OSDI 2012, pp. 237–250 (2012)
12. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. 4(3), 382–401 (1982)
13. Li, Z., Tan, L., Wang, X., Lu, S., Zhou, Y., Zhai, C.: Have things changed now?: an empirical study of bug characteristics in modern open source software. In: Proc. ASID 2006, pp. 25–33 (2006)
14. Lu, S., Park, S., Hu, C., Ma, X., Jiang, W., Li, Z., Popa, R.A., Zhou, Y.: Muvi: automatically inferring multi-variable access correlations and detecting related semantic and concurrency bugs. In: SOSP 2007, pp. 103–116 (2007)
15. Mariano, P., Soares, J., Preguiça, N.: Replicated software components for improved performance. In: InForum 2010, pp. 95–98 (2010)
16. Musuvathi, M., Qadeer, S., Ball, T., Basler, G., Nainar, P.A., Neamtiu, I.: Finding and reproducing heisenbugs in concurrent programs. In: OSDI 2008, pp. 267–280 (2008)
17. Pullum, L.L.: Software fault tolerance techniques and implementation. Artech House, Inc., USA (2001)
18. Purtilo, J.M., Jalote, P.: An environment for developing fault-tolerant software. IEEE Trans. Softw. Eng. 17, 153–159 (1991)
19. Qin, F., Tucek, J., Sundaresan, J., Zhou, Y.: Rx: treating bugs as allergies—a safe method to survive software failures. In: SOSP 2005, pp. 235–248 (2005)
20. Rodrigues, R., Castro, M., Liskov, B.: Base: using abstraction to improve fault tolerance. In: SOSP 2001, pp. 15–28 (2001)
21. Romanovsky, A.: Class diversity support in object-oriented languages. Journal of Systems and Software 48(1), 43–57 (1999)
22. Sidiroglou, S., Ioannidis, S., Keromytis, A.D.: Band-aid patching. In: HotDep 2007 (2007)
23. Veeraraghavan, K., Chen, P., Flinn, J., Narayanasamy, S.: Detecting and surviving data races using complementary schedules. In: SOSP 2011, pp. 369–384 (2011)
24. Xu, J., Randell, B., Rubira-Calsavara, C., Stroud, R.J.: Toward an object-oriented approach to software fault tolerance. In: Proc. FTPDS 1994, pp. 226–233 (1994)

GRIMACE: GeneRIc MetAmodel for Domain Component modElling

Rahma Bouaziz

IRIT, University of Toulouse
118 Route de Narbonne, 31062 Toulouse, France
rahma.bouaziz@irit.fr

Abstract. Component Based Software Engineering (CBSE) is a popular and widely adopted software engineering paradigm that has proven his usefulness and success to increase reusability and efficiency in various application domains. In this paper, we propose a common metamodel to support CBSE requirements taking into account the specificities of each domain. The resulting modeling framework serves primarily to capture the basic concepts of concerns related to component systems development based on the clear separation between the development process, interactions and the domain knowledge.

1 Introduction

Component-Based Software Engineering (CBSE) [1] has emerged as a promising key technology for developing and maintaining complex systems. CBSE focuses on building large software systems by integrating previously existing software components. The system is constructed by the composition and the connection of these components. It is a good solution to optimize time and cost of software design while still guaranteeing the quality of the software [2].

Various component models have been proposed to deal with system complexity in industrial and academic domains. Variety of those model's applications in constructing systems has proved their usefulness and success. Among these approaches, we can find general-purpose software component models such as Enterprise Java Beans (EJB) [3], CORBA Component Model (CCM) [4] which are well-established for CBSE in generic problem domains. On the other hand, to address a specific domain challenge, specific component models like KOALA [5] are proposed to deal with specialized domains like distributed, embedded or real time systems.

Our first objective is to combine these two approaches – generic and specific component models – in order to propose a metamodel that overcome some of draw backs and take advantage of each approach. In other terms, we propose a common representation of generic and specific component models taking into account domain specific concerns at the design level.

Model-Driven Engineering (MDE) [9] is also another emerging approach in system development. The use of models has become a major paradigm in software engineering. Its use represents a significant advance in terms of level of abstraction,

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 181–184, 2013.

continuity, generality, and scalability. In this paper, we deal with these two technologies–CBSE and MDE–to propose a model-based component framework to get a common representation of component for several domains. The main motivation of this work is that the reuse of knowledge and expertise at high level is fundamental to guarantee quality systems. Inspired by the MDE methodology in which software is developed by constructing high level models, we propose a generic component metamodel–to capture generic concepts of CBSE approach –and then separate domain independent aspects of component model from those that are domain specific. The remainder of this paper is organized as follow. In Section 2, we present the proposed approach and describe the GRIMACE metamodel. Section 3 concludes the paper and presents ongoing work.

2 The GRIMACE Approach

The key idea presented in this work is to propose a common representation to target several domains of systems applications (e.g. distributed, embedded, real time, etc.). This representation allows to work at a higher abstraction level, which may significantly reduce the cost of system engineering. Our goal is to define the component based systems as easily and quickly as possible. To do this, the modeling framework must include simple abstractions known by the software developer. Our proposed architecture is based on models, which specify different levels of abstraction, helping developers to manage the inherent complexity of applications and facilitating the communication between the different contributors of software development.

2.1 An Overview

The approach is based on three levels of abstraction (see Fig. 1): GRIMACE metamodel, DICM model and DSCM model. In the following, we detail each of these models.

GeneRIc MetAmodel for domain Component modElling (GRIMACE). In This level – Metamodel level (M2) –we present a generic component metamodel that represents as its name suggests the abstract concepts of component based approach proposed by a large set of component models to describe software architectures. It provides the basic modeling elements for component based system: Component, Connector, Interface, Ports, etc. These elements are the basis for instantiating different component model.

Domain Independent Component Model (DICM). This model is an instance of GRIMACE. This level is intended to generically represent component independently from the application domain. Hence, we will focus on the representation of generic component concepts, interaction and connection between component conforming to the component metamodel (GRIMACE). A DICM model is conform to the GRIMACE Meta-model. Therefore each element of DICM is associated with an element of GRIMACE.

Domain Specific Component Model (DSCM). This model corresponds to a refinement of the DICM model. It combines component specification presented in DICM and the domain specification such real-time constraints. This model is, at the same time, an instantiation of the Generic Component model (GRIMACE) and a refinement of DICM. In this stage the DICM is tailored to a particular application domain e.g. real time embedded or distributed system. For example we can build from the Fractal model (DICM) several specific component models (DSCM) like Fractal distributed, Fractal real-time or Fractal embedded component models.

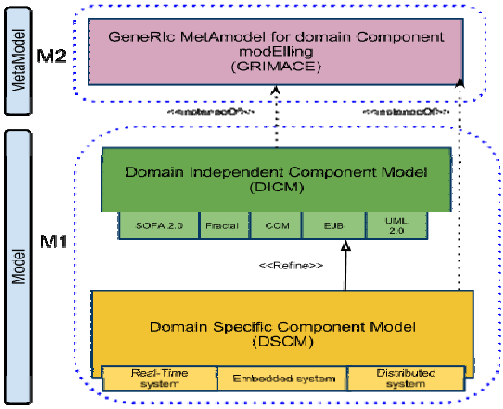


Fig. 1. Overview of the proposed Architecture

2.2 GRIMACE: GeneRIc MetAmodel for Domain Component modElling

Based on a large study of general purpose and specific component models, we have defined the GRIMACE meta-model. It provides a component model state-of-the-art offering most features available in the domain of component-oriented software architectures.

GRIMACE describes concepts and their relations needed to represent component systems. The proposed meta-model is based on CBSE principles, containing the basic entities *Component*, *Interfaces* and *Connector*. Fig. 2 shows the proposed meta-model. These components will be used to describe both Domain-Independent (DI) and Domain-Specific (DS) models i.e., the originality is the use of the domain independent model and the domain specific model which are reflected in the meta-model level. In the proposed meta-model, a system is described by a set of components that represents the units of computation and data. The communication mode and coordination between these components is encapsulated in connectors. GRIMACE meta-model separates the computation concept represented by components and the interaction concept presented by connectors.

As shown in Fig.2 the generic meta-model defines the abstractions of component based concepts and allows the design of generic component based applications. In addition to these classes and associations, relations can be more precisely defined by adding constraints. Because a lack of space we will not present OCL constraints in this paper.

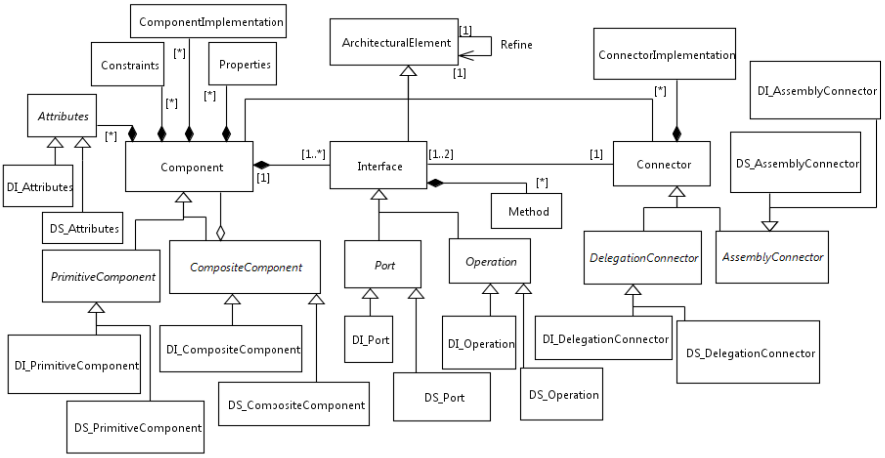


Fig. 2. GRIMACE: GeneRIc MetAmodel for domain Component modElling

3 Conclusion and Future Work

In this paper, we have proposed an overview of concepts–of several component based approaches among academic and industrial fields–to build common representation of component models for several domains. In this way this work gives several ways to go towards universal concepts to model components systems. Among these concepts, we can cite the following: (1) high level of abstraction to capture all the facets of a component assembly, (2) the domain independent design and (3) the domain specific design. Future work includes defining secure models (as instances of GRIMACE metamodel) by using security patterns [6].

References

[1] Brown, A.W., Wallnau, K.C.: The current state of CBSE. IEEE Software 15(5), 37–46 (1998)

[2] Szyperski, C.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley/ACM Press, Wesley, New York (2002)

[3] DeMichiel, L.G. (ed.): Enterprise JavaBeans Specification, Version 2.1., November 12. Sun Microsystems, Inc., Santa Clara (2003)

[4] OMG, CORBA Component model, `http://www.omg.org/technology/documents/formal/components.html`

[5] van Ommering, R., van der Linden, F., Kramer, J., Magee, J.: The Koala Component Model for Consumer Electronics Software. Computer 33(3), 78–85 (2000)

[6] Bouaziz, R., Hamid, B., Desnos, N.: Towards a better integration of patterns in secure component-based systems design. In: Murgante, B., Gervasi, O., Iglesias, A., Taniar, D., Apduhan, B.O. (eds.) ICCSA 2011, Part V. LNCS, vol. 6786, pp. 607–621. Springer, Heidelberg (2011)

Improving the Transfer of Safety and Security Competences to Industry: The RISKY Approach

Joaquín Gracia-Morán, Juan-Carlos Ruiz, Juan-Carlos Baraza-Calvo, David de Andrés, and Pedro Gil

Departamento de Informática de Sistemas y Computadores (DISCA)
Universitat Politècnica de València
Camino de Vera s/n, 46022 Valencia, Spain
{jgracia,jcruizg,jcbaraza,ddandres,pgil}@disca.upv.es
http://www.upv.es/entidades/DISCA/indexi.html

Abstract. Transfer to industry is typically understood by academia as a transfer of methodologies and technologies, thus neglecting transfer of knowledge. However, academia is very well placed to improve industry competitiveness through continuous training. Coping with transfer of knowledge is not only a matter of providing courses, but also of considering the lifelong training requirements of professionals and the professional competence framework existing in each domain of expertise. The RISKY project takes into account the current european framework for the transfer of competences across Europe, and the existing certifications promoted by professional bodies, in order to develop and use methods and tools adapted to the training of Security and Safety professionals.

Keywords: Safety, Security, Vocational Education and Training, Competences Assessment.

1 Introduction

Although the IEEE-CS and the ACM [1] curriculum guidelines are between those more widely followed in universities, they present important lacks on training related to Safety and Security. The industry demands professionals with practical skills in these important domains of expertise. This is mainly why related vocational training typically relies on the use of use cases providing return of experience to learners [2]. However, existing certifications are either very domain specific (like those promoted by CISCO [3]) or very dependent on professional bodies and associations (like those promoted by TÜV [4] and ISACA [5]).

In order to place some order, Europe is currently promoting different initiatives, such as the European e-Competence Framework[1], the European e-Skills Forum[2] and the Leonardo da Vinci programme[3] in order to stablish a common European

[1] http://www.ecompetences.eu
[2] http://eskills-week.ec.europa.eu
[3] http://ec.europa.eu/education/lifelong-learning-programme/ldv_en.htm

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 185–189, 2013.

framework for competence-based training and evaluation. Although not specifically focused on security and safety, the goal of these initiatives is to ease the exchange of professional competences across Europe.

This work presents the RISKY approach [6], which aims at providing a work-based learning pedagogical approach to Safety and Security. The main idea is to work closely with professional needs and to design courses according to trainees profile and their job requirements. This will give trainees the opportunity to adjust their skills and competences to cover professional profile requirements.

Section 2 summarizes conventional academia approaches to Safety and Security training. Section 3 shows the framework for the European Credit for Vocational Education and Training. Section 4 describes the RISKY approach, and finally, Section 5 concludes this paper.

2 Conventional Training on Safety and Security

As just commented, undergraduate programs in computing usually follow the curriculum guidelines provided by IEEE-CS and ACM [1]. However, Safety and Security is very shallowly addressed by such training referential. Master and PhD training tries to fix this lack. The MSc program proposed by the RESIST[4] NoE is an example of this. It focuses on Resilient Computing and it structures the curriculum in 120 ECTS[5]. After such training, the student can integrate the industry or perform a PhD to integrate the industry about 3 years later. In both cases, the main problem is that the transfer of knowledge is implicit, since it does not exist any explicit identification of the set of competences provided by the program. Which is thus the profile of professionals issued from such program? What are they able to do? Which kind of tasks are they able to cope with? Reformulating existing training programs to provide explicitly competences (skills and abilities) may help future employers in the selection of personnel issued from Security and Safety programs.

3 Vocational Training in Europe from the Perspective of Safety and Security: Situation and Challenges

The formative path of a Safety and Security professional is a continuous training process. Different organizations promote their own qualifications, divided in a great number of expertise levels based mainly in the number of professional years of experience [6]. Also, each certification provides their own learning outcomes, being very difficult to set common learning paths between certifications.

On the other hand, professional training differs from undergraduate, MSc or PhD training in the sense that it must be further based on experience. This is why Work-Based Learning (WBL) [2] is preferred in such kind of training

[4] http://www.resist-noe.org/Publications/Deliverables/D37-Curriculum.pdf
[5] http://ec.europa.eu/education/lifelong-learning-policy/ects_en.htm

context. WBL promotes not only the acquisition of knowledge, but also the development of skills and the assimilation of attitudinal patterns. In other words, it promotes the use of competences (understood as knowledge, skills and attitudes) in practical work situations.

However, the definition of which are the specific competences required from Safety and Security engineers is not an easy issue, since it must take into consideration the actual, and possible future, necessities of ICT professionals. In addition to this, such competences must be common to, and thus accepted by, all European countries. In this sense, different initiatives, such as the `European e-Competence Framework`[1], the `European e-Skills Forum`[2] and the `Leonardo da Vinci programme`[3], try to identify common Europe-wide competences for professionals working in different domains.

Beyond the development of competences, one should not forget that training professionals implies working with learners with high potential mobility, which can devote a limited amount of time to their studies and whose prior learning can be quite heterogeneous. As a result, units of content should be follow the design of knowledge pills, they should remain available 24/7/365 (so they must be online) and, for each unit, the prior leaning should be precisely identified, and assessed as well as the expected unit learning outcomes (knowledge, skills and attitudes).

Agreeing on such learning outcomes, and thus on the related competences, is quite challenging. The European Credit for Vocational Education and Training or ECVET "*aims for better compatibility between the different vocational education and training (VET) systems in place across Europe*". As it is done with the ECTS[5], qualifications are defined based on *learning outcomes*. Procedures and rules are necessary for the assessment, transfer, accumulation and recognition of the obtained credits. This feature makes ECVET a very flexible system, as long as ECVET learning outcomes are assessed and validated for *transfer credits* among different learning paths, that can be taken in different countries or different educational contexts. This idea is currently implemented in the `europass`[7] initiative, where the proposal is to use five documents to describe the skills and qualifications of professionals in Europe.

4 The RISKY Approach

The RISKY project is an ECVET-compliant competence- and WBL-based training framework for Security and Safety professionals. The RISKY project embraces WBL approaches as leading methodologies for leveraging the competences of learners towards credited qualifications. Fig. 1 provides a high-level standpoint of the RISKY approach.

A particular instantiation of the RISKY approach implementation was held on June 2012 at NOT premises (Polish Federation of Engineering Associations) in Ostrołęka (Poland) [7]. A half day seminar was prepared to make professionals aware of the importance of Safety and Security training in their businesses.

[7] `http://europass.cedefop.europa.eu/en/home`

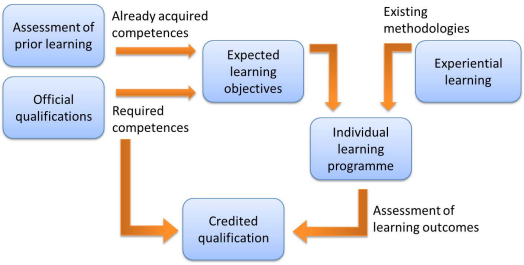


Fig. 1. Proposed roadmap for WBL in the context of the RISKY project

20 professionals with good technical skills, but low to medium security- and very low functional safety-awareness, coming from small and medium NOT-adhered enterprises assisted to the event. A 30 minutes lecture was prepared revolving the IEC 61508 standard. Such a lecture could be declined from different expertise levels. So *assessing the prior learning* of the attendees was of prime importance to correctly focus the aim and scope of the lecture. This information, provided by NOT Ostrołęka in our case, reflected that attendees had a very low level of expertise in the functional safety domain.

Having this in mind, and considering the aforementioned timing constraints, a *simple case study* was devised to embrace the WBL approach promoted by RISKY. This case study guided learners, by means of simple but effective examples, through the safety life cycle phases defined by the IEC 61508 standard, unveiling the competences required by safety practitioners in each phase.

As defined in the proposed approach, *learning outcomes should be assessed* to determine whether they have been achieved or not. In our case, the goal of the lecture was just to increase the awareness of attendees towards the functional safety domain. After the seminar, attendees were invited to join a group, which is currently being dynamised by NOT Ostrołęka, for discussing topics related to safety with polish small and medium enterprises.

5 Conclusions. Future Work

This paper summarizes the work attained so far by the RISKY consortium. The goal is to approach Security and Safety professionals in order to adapt existing WBL training methodologies to their needs, while attending to ECVET-related constraints.

Acknowledgements. This work has been funded by the RISKY Leonardo da Vinci project (#2011-1-FR1-LEO05-24482) from the European Commission. This publication reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

References

1. The Joint Task Force for Computing Curricula: Computing Curricula 2005 - The Overview Report. Technical report, ACM, AIS & IEEE-CS (2006), http://www.acm.org/education/curric_vols/CC2005-March06Final.pdf
2. Flemming, F.K., Rokkjiaer, O., Schrey, K.: Work Based Learning and Facilitated Work Based Learning. Technical report, TREE - Teaching and Research in Engineering in Europe, Special Interest Group D8: "Work Based Learning" (2007)
3. CISCO Systems: Training and Certifications (2013), http://www.cisco.com/web/learning/training-index.html
4. TÜV Rheinland Industry Service GmbH: TÜV Functional Safety Program (2013), http://www.tuvasi.com/en/t-v-functional-safety-program/t-v-functional-safety-program.html
5. Information Systems Audit and Control Association (ISACA): ISACA Certifications: IT Audit, Security, Governance and Risk (2013), http://www.isaca.org/CERTIFICATION/Pages/default.aspx
6. Ruiz, J.C., de Andrés, D., Baraza, J.C., Gracia, J., Gil, P.J., Pelud, L., Echardour, P., Koniewski, K.: RISKY: a Dynamic Vocational Training System for Safety and Security. In: Jornadas de Innovación Educativa 2012, pp. 327–331 (2012)
7. Federacja Stowarzyszeń Naukowo: Technicznych NOT Rada Regionalna w Ostrołęce (2013), http://www.notostroleka.pl

Towards Dependable Measurements in Coastal Sensors Networks

Gonçalo Jesus[1], António Casimiro[2], and Anabela Oliveira[1]

[1] LNEC – National Laboratory for Civil Engineering, Lisbon, Portugal
{gjesus,aoliveira}@lnec.pt
[2] FCUL – Faculdade de Ciências da Universidade de Lisboa, Lisbon, Portugal
casim@di.fc.ul.pt

Abstract. A flood monitoring system incorporates water sensor networks, forecast simulations models, and a decision-support web-based system. The objective of the system is to achieve reliable flood protection and response. This is challenging because of the inherent presence of a cascade of uncertainties in the forecast models, and also uncertainties affecting the timeliness and quality of raw sensor data that is used in the forecasting processes. Achieving real-time and accurate data collection is difficult due to the pervasive nature of the monitoring networks and because sensors and sensor nodes are vulnerable to external disturbances affecting data accuracy. In this paper we motivate for the need of dependable data collection in harsh coastal and marine environments, we overview the main challenges that need to be addressed and we introduce some initial ideas on what needs to be done in order to deal with external disturbances causing faulty measurements.

Keywords: Marine sensors, reliable measurements, dependable sensors, failure detection.

1 Introduction

Preventive strategies against natural and man-made disasters are currently being improved and their effectiveness will lessen the impact of the hazards. Thus, the availability of reliable systems delivering reliable information from reliable sources is vital for the protection of human lives as well as material and natural assets. In flood managing those systems need to issue alerts or to support decisions on mitigation measures to be performed in areas at risk, with the help of reliable forecasts and dependable real-time monitored data, such as water level, flow or precipitation level.

The systems which integrate monitoring networks, large-scale distributed computation (using shared heterogeneous pooled resources across administrative domains) either for forecasting/simulations or data analysis, and a web-based system to support posterior on-time decisions, are addressed as pervasive management support (PMS) systems [1], used mainly to manage environmental threats.

Although PMS systems are complex, the constrained time frame of a response is desirably short, for successful protection of people and assets. When it exceeds the

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 190–193, 2013.

target response time, warnings are useless and might carry significant financial penalties and loss of human lives, so both hardware and software failures are not tolerable at critical moments in any of the tiers.

2 Real-Time Environment Monitoring

Water bodies monitoring for early flood detection and enhanced response has become a vital infrastructure system in order to provide accurate, reliable and timely information to assist local authorities and experts in responding flood events. The availability, coupled with reliability, of the near real-time monitoring information is a difficult requirement in effective implementations of monitoring networks. So, in the last years, many studies on wireless sensor networks have been performed, in order to cover the usually large monitoring area [2] and to fulfill the requirements.

The quality of flood forecasting also depends on the quality of measurements, or on the forecasts of precipitation and other forcings. The better the estimation of the input of water during an event, the better is the chance of producing accurate forecasts. The most common problems in measuring networks are the accuracy of the sensor measurements, the possibility of the equipment getting damaged in environment-related events (leading to data unavailability or data corruption) and the occupancy of the area to be monitored (with influence on the probability that some hazard will happen). Thus some calibration of the effective values of the monitored parameters is generally necessary.

Nowadays the use of real-time sensors in flood monitoring is an important non-structural measure of flood control. One of the most used infrastructures to build the monitoring network of a PMS system is a wireless sensor network [2-4], where the monitoring system can be decomposed into a set of remote wireless communication systems, which log water condition data, and transfer the data to a web-based information center to build a real-time web-based management system. These wireless remote sensors are scattered in the field and separated from each other by a large distance.

3 Coastal Sensors Network Challenges

The increasing level of heterogeneity is a relevant issue in modern distributed systems. Generic instruments or applications that were built or studied to be adaptive, autonomic, dependable, secure and scalable, and tested on controlled situations, now need to operate in increasingly-varied environments, such as a pervasive and complex marine environment, combined with different types of sensors measuring several parameters (many interconnected).

In the design of context-aware monitoring networks, the first thought is that there should be a deep understanding of application requirements in order to develop software and hardware that automatically adapts to aspects of the environment, such as sensor's current location and activity, the time of day, and the presence of other factors in the vicinities [5].

As mentioned above, real-time accurate monitoring of hydraulic parameters is key for flood forecasting. But for these hydraulic applications the sensors may have technological limitations, and so besides the regular requirements a number of additional ones are necessary [3]:

- Power lifetime – the power sources are often not available at all the locations of interest. Moreover, these locations are usually unsafe, unprotected, and, if renewable energy devices are used, they are prone to vandalism or theft. Thus, it is imperative the use of sensor nodes with low-consumption, with standard batteries that should last several months cycle;
- Sensor hardware compatibility – most hydrologic sensor nodes include a data logger device connected through a cable to the measurement instruments. The data logger must offer interfaces to communicate with a range of specific sensor hardware, which involves issues of power supply, and selective time for power dispatching, involving a optimal power management and facilitation of the expansion of connected instruments;
- Reliability – the harsh weather conditions may cause failures in the measurements and in the wireless communication over the monitoring network. Backup mechanisms in local sensor data loggers must be used to avoid information losses in unexpected crashes;
- Long-range communication – hydraulic measurement locations are commonly sparse over large areas, and not even in the same area of the control center.

Thus, with these requirements, it is imperative to understand the different factors that affect the operation of the floating or diving sensors. Each type has its own and unique characteristics that should be reflected on the sensor's measurements. A careful study on these factors should be conducted and a comparative analysis of the different characteristics should be carried out.

4 Study Considerations and Future Approaches

The paper introduces an on-going research for the development of a framework that will support the dependability on aquatic real-time forecasting systems, starting by the study on the influence of external factors related with coastal and marine environment on the sensor network behavior, more specifically on the sensors measurements, and their consequences on the forecasts and alert systems that will be fed with the monitored data. The correct modeling of the impact of external factors such as weather events and conditions and marine interferences as a cause of measurement failures in interference-prone wireless sensor networks [6-8] is of the utmost importance to define and develop system solutions providing awareness of the delivered sensor data quality, aiming at achieving dependable reliable coastal and marine sensor networks.

The first step is to identify and characterize the probable factors of disturbances, possibly done through a statistic analysis of the negative influence of natural environmental events on the measurement process of the available sensors on the field. This statistical validation of the measurements can offer a robust and inexpensive way

to minimize the influence of these faults on future flood monitoring systems, providing a simple method to increase the fault tolerance aspect of those particular sensors, and the confidence of the data provided.

A second step would be the development of solutions to automatically adjust the sensors measurements to each disturbance accordingly, contributing to an important increase on the quality of the measurements, thus supplying other layers of PMS systems with dependable monitoring data.

However, on further steps, much has to be done in studying all aspects of the criticality of the sensor network in the monitoring process, including the timeliness requirements of the flood warning systems. Future work will include studying the real impact of networking faults and temporal uncertainties and devising solutions to deal with them, namely using approaches based on redundancy and sensor fusion.

Acknowledgements. This study was funded by FCT through the Multiannual Funding Program and through PhD Grant (SFRH/BD/82489/2011) with the support of NTI group of DHA-LNEC and LaSIGE research unit, and also FCT's project Pac:man (PTDC/AAC-AMB/113469/2009), SPRES (Interreg Atlantic Area Transnational Cooperation Programme 2007-2013 project SPRES - 2011-1/168), and FLAD (project CWOS).

References

1. Coulson, G., Kuo, D., et al.: Sensor Networks+ Grid Computing= A New Challenge for the Grid? Distributed Systems Online 7(12), 2 (2006)
2. Smith, P.J., Hughes, D., et al.: Towards the provision of site specific flood warnings using wireless sensor networks. Meteorological Applications 16(1), 57–64 (2009)
3. Marin-Perez, R., García-Pintado, J., et al.: A Real-Time Measurement System for Long-Life Flood Monitoring and Warning Applications. Sensors 12(4), 4213–4236 (2012)
4. Hughes, D., Greenwood, P., et al.: An intelligent and adaptable grid-based flood monitoring and warning system. In: Proceedings of the 5th UK eScience All Hands Meeting (2006)
5. Henricksen, K., Robinson, R.: A survey of middleware for sensor networks: state-of-the-art and future directions. In: Proceedings of the International Workshop on Middleware for Sensor Networks, pp. 60–65. ACM (2006)
6. Boano, C.A., Tsiftes, N., et al.: The impact of temperature on outdoor industrial sensornet applications. IEEE Transactions on Industrial Informatics 6(3), 451–459 (2010)
7. Bannister, K., Giorgetti, G., et al.: Wireless sensor networking for hot applications: Effects of temperature on signal strength, data collection and localization. In: Proceedings of the Fifth Workshop on Embedded Networked Sensors, HotEmNets 2008. Citeseer (2008)
8. Dietrich, A., Zug, S., et al.: Detecting external measurement disturbances based on statistical analysis for smart sensors. In: 2010 IEEE International Symposium on Industrial Electronics, ISIE, pp. 2067–2072. IEEE (2010)

Open Challenges in the Resilience Evaluation of Ad Hoc Networks*

Miquel Martínez[1], Jesús Friginal[2], David de Andrés[1], and Juan-Carlos Ruiz[1]

[1] STF-ITACA, Universitat Politècnica de València, Campus de Vera, 46022, Spain
{mimarra2,ddandres,jcruizg}@disca.upv.es
[2] LAAS-CNRS, 7 Avenue du colonel Roche. 31400 Toulouse, France
jesus.friginal@laas.fr

Abstract. Wireless ad hoc networks are spontaneous, self-healing and self-managing systems strongly raising in the last decade. However, their deployment in privacy- or life-critical scenarios still requires a deeper analysis to determine their robustness to faults/attacks and their ability to recover from situations degrading performance and dependability. Unfortunately, several challenges limit the development of practical assessment approaches for ad hoc networks. This paper focuses on identifying these challenges to provide potential evaluators with a guide of the sensitive points that require an especial attention to improve the credibility of results when addressing the resilience evaluation of ad hoc networks.

1 Introduction

Wireless technology is being deployed in many different kinds of devices nowadays, so the number of devices that are able to exchange information through a wireless environment is growing fast. According to [1], by the year 2020 each person will own almost 7 devices with connection capabilities. These devices will require the deployment of self-managing and self-adaptive networks that enable the coexistence and interoperation of multiple heterogeneous devices in constant evolution. Ad hoc networks perfectly suit this requirement, but before they can be deployed in an everyday-life scenario, at home, work or leisure activities, a deeper analysis of their performance and resilience (dependability in spite of changes) must be done.

Many studies have been done to evaluate the performance of ad hoc network with respect to the number of works focusing on their resilience evaluation. Even though there are different evaluation strategies that can be found in the literature, such as, (i) simulation based on network models; (ii) prototyping based on real devices and network deployments; and (iii) emulation encompassing both real and simulated aspects, few proposals consider the evaluation of ad hoc networks in presence of threats, taking into account both performance and resilience measures.

* This work is partially supported by the Spanish project ARENES (TIN2012-38308-C02-01), the ANR French project AMORES (ANR-11-INSE-010), and the Intel Doctoral Student Honour Programme 2012.

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 194–197, 2013.

The misbehaviour of ad hoc networks in critical scenarios such as Vehicular Ad Hoc Networks (VANETs) and Wireless Sensor Networks (WSNs), may cause important economic or human damages. Even the reputation of Wireless Mesh Networks (WMNs) providers may be strongly affected. Thus, resilience evaluation in ad hoc networks should be no longer an option, but a must.

This paper points out the challenges that still remain unaddressed in the resilience evaluation of ad hoc networks. With this purpose, the rest of this paper addresses the challenges affecting the resilience evaluation process of this type of networks. Thus, Section 2 presents some of the most important challenges to be faced during the configuration of experiments, Section 3 shows those from the experimental stage, and Section 4 identifies those regarding the analysis of resilience evaluation results. Finally Section 5 concludes the paper.

2 Challenges of the Experiments Configuration

Including a resilience-aware point of view within traditional performance evaluation involves addressing the notion of resilience when determining (i) the set of measures to characterise the system, (ii) the properties of both the system and target under evaluation, and (iii) the necessary operational profile (comprising the work-, perturbation- and change-load conditions) that will be used to exercise the resilience of the system.

Most of previous works done around the evaluation of ad hoc networks characterise the quality of the system through performance measures as throughput, delay, routing overhead, packet delivery ratio or jitter exhibited by the network [2]. But surprisingly, very few proposed measures to quantify dependability-related aspects. For instance, in [3] the route availability is measured as the percentage of time that a route is available when it is needed. However, there is a lack of dependability measures characterising specific ad hoc features impacting in the resilience of routing protocols, such as the average failure recovery time (understood as the time required to reconfigure the network topology in order to get a new valid route when an existing one is lost).

Most of times, evaluators may face a problem that appears with the need of deploying evaluation targets (discovering, routing, synchronisation and any other communication protocol) within ad hoc networks to support their assessment. The main cause of this problem is the heterogeneity of devices that may compose an ad hoc network (motes, sensors, laptops, smartphones and so on), and the potential different technologies (Bluetooth, Zigbee, IEEE 802.11X, etc.) they may use. Considering this combination of technologies, and the different kinds of ad hoc networks (WSN, WMN, VANET, etc.), defining an operational profile that includes work-, perturbation- and change-load it is a challenging task. The workload must be specific for each type of network, as the workload for a WSN would not reflect the real behaviour of a WMN. The same occurs with the perturbation-load (faults and attacks), which should reflect those perturbations that affect a certain kind of ad hoc network, and the change-load [4], that represents the set of changes that impact a system, like the particular mobility of the nodes.

3 Challenges of the Execution of Experiments

The execution of experiments, especially in practical evaluation platforms, presents challenges related to their design, implementation and deployment. On the one hand, different sources of uncertainty may affect evaluation results, on the other, addressing scalability and portability issues is essential for the practical use of experimental platforms for ad hoc networks.

An important challenge relates to the sources of indeterminism existing in ad hoc networks. These sources (interferences, noise, environment obstacles, etc.) limit the reproducibility of experiments since they introduce uncertainty in the considered experimental conditions. They have also a negative impact on the controllability of the experimental procedure itself, specially when the conditions used to warm-up and animate the system under evaluation/benchmarking, or start and stop each experiment or evaluation campaign, rely on the triggering of a particular environmental condition, or the computatiton of a concrete statistic. An obvious solution can be to rely on simulation to avoid such *real-world* problems, but the price to pay is a reduction of the accuracy of results.

Likewise, the use of real devices will certainly limit the number of nodes that an evaluation platform may contain, thus affecting the scalability. This problem causes that evaluation platforms that make use of real devices rarely expand beyond 10 or 20 nodes [5]. A possible approach to solve this problem consists in simulating the experiments with the very same configuration as used in real deployments, and comparing the trend of results obtained looking for similarities. Unfortunately, to date, simulators are far from recreating the exact conditions of real experiments. The incomplete characterisation of aspects such as the mobility patterns of nodes (including the human factor within network deployment), or the influence of radio channel within network models to approach reality, is a challenging task.

Once again the use of real devices may hinder the portability of a platform. Up to now, the portability of real frameworks has been a hard task in the domain of ad hoc networks given the complexity of redeploying testbeds. Making evaluation platforms more portable would open new opportunities so that platforms could be used by different evaluators, thus sharing the experience of evaluation among users all around the world, consequently improving the knowledge about their behaviour.

4 Challenges of the Analysis of Results

The analysis of results is crucial as the conclusions of the resilience evaluation will rely on this stage. The first challenge consists in determining how to select and filter the adequate information from the whole set of measurements obtained by each node for each experiment, regardless whether this information is located in the same physical repository (typical in case of simulations), or it is distributed into different stores (usual in real-world experiments). Specially in case of considering the execution of experiments that may require several repetitions, it

will be essential to aggregate filtered measurements for all the experiments executing the same configuration. Finally, it will be necessary to compute proper statistic indicators to correlate filtered and aggregated measurements. Statistical confidence intervals may result very useful to justify the reliability of population samples. Strikingly, despite their benefits, very few evaluators use them.

Once measures processed, resilience evaluators face a crucial problem that strongly influences the analysis of results. To be useful, the measures extracted must be correctly interpreted following systematic criteria. The context where the ad hoc network is deployed is a key factor that should be taken into account, as measures of the same network in different contexts could be interpreted in many different ways. Despite the importance of this point, a lot of works on the bibliography still analyse measures generically without applying a proper interpretation about where the network is deployed in.

5 Conclusions

This study is motivated by the lack of works addressing the complex problem of resilience evaluation of ad hoc networks. Listed challenges defy the ability of evaluators to deploy more practical experimentation in this domain. So tackling and covering the issues presented in this paper becomes essential to reduce the gap between the networks we may deploy, and the systems we are able to evaluate.

References

1. Fehske, A., Fettweis, G., Malmodin, J., Biczok, G.: The global footprint of mobile communications: The ecological and economic perspective. IEEE Communications Magazine 49(8), 55–62 (2011)
2. Vivian, D., et al.: Evaluation of QoS Metrics in Ad Hoc Networks with the Use of Secure Routing Protocols. In: Network Operations and Management Symp., pp. 1–14 (2006)
3. Friginal, J., Ruiz, J.-C., de Andres, D., Bustos, A.: Mitigating the impact of ambient noise on wireless mesh networks using adaptive link-quality-based packet replication. In: 2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 1–8 (June 2012)
4. Almeida, R., Vieira, M.: Changeloads for resilience benchmarking of self-adaptive systems: A risk-based approach. In: 2012 Ninth European Dependable Computing Conference (EDCC), pp. 173–184 (May 2012)
5. Hortelano, J., et al.: Castadiva: A Test-Bed Architecture for Mobile AD HOC Networks. In: IEEE Symposium on Personal, Indoor and Mobile Radio Communications, pp. 1–5 (2007)

Using Interleaving to Avoid the Effects of Multiple Adjacent Faults in On-Chip Interconnection Lines

Luis-J. Saiz-Adalid, Pedro Gil, Joaquín Gracia-Morán,
and Juan-Carlos Baraza-Calvo

Instituto ITACA, Universitat Politècnica de València
Camino de Vera, s/n, 46022, Valencia, Spain
{ljsaiz,pgil,jgracia,jcbaraza}@itaca.upv.es

Abstract. As technology shrinks, higher operating frequencies, reduced feature sizes and lower supply voltages allow greater performance, but the reliability has been affected negatively. Smaller devices and wire spacing lead to an increase in the occurrence of multiple adjacent faults. Thus, the system reliability is seriously affected. Error correction codes are a powerful technique that allows higher reliability using information redundancy. This paper focuses on the use of interleaved codes to tolerate faults in on-chip interconnection lines. Interleaving has been extensively used in memories, but not in system buses. To illustrate the features of this technique, an example has been included.

Keywords: Error Correction Codes, Hamming Codes, Interleaving, Multiple Adjacent Faults.

1 Problem Description

The advances in integration have allowed increasing circuit operating frequencies while reducing sizes and supply voltages, achieving a greater performance. This trend has resulted in smaller devices and shorter wire spacing. However, these advances have had a negative impact on the reliability of modern circuits. The increase of manufacturing residuals and process variations in new VLSI technologies are raising the fault rate in such circuits [1]. Faults in microprocessors can lead to hardware errors and failures, which will be propagated to upper system levels [2]. Hardware mitigation techniques allow faster responses and reduce the failures propagated to upper levels [3].

Error control codes (ECCs) protect data using information redundancy [4]. On-chip parallel interconnection lines suffer different kinds of faults. Fault mechanisms can lead to single or multiple faults. Even more, multiple faults are frequently observed as adjacent faults, which usually manifest as a burst error (a multiple error that spans l bits in a word [4]; the separation l is known as burst length). Their causes can be bad solder joints, crosstalk or manufacturing residuals affecting several neighbor locations [5], for example. Single event effects (SEE) [6] are also a cause of adjacent faults in nano-scale devices. As stated above, the spacing of wires in a bus is decreased, but

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 198–201, 2013.

the width of the single-event charge track does not shrink. In this way, one particle probably affects several wires (see Fig.1). This effect can be observed at ground level, but it is especially relevant in space missions, where the occurrence of SEE is higher, and a failure could be difficult to solve and could result in mission fail or risk for human lives.

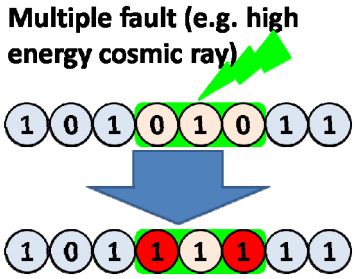


Fig. 1. Multiple fault caused by a Single Event Effect

This paper proposes the use of interleaved codes to mitigate the effects of adjacent faults in on-chip parallel interconnection lines. They have been used extensively in data storage and high volume communications, but usually they are not considered in system buses and other parallel interconnection lines.

Interleaved codes can be useful for on-chip parallel interconnection lines because these lines have special features, such as the reduced word size (64 bits is quite smaller than, for example, the size of a DVD sector, which is bigger than 2 Kbytes), need for fast encoding and decoding (as system buses are massively used in the execution of processor instructions), and desired low area and power overhead.

2 Interleaved Extended Hamming Code (IxH)

As an example of code construction, let us consider a 16-bit input word, where the requirements are: i) to correct burst errors of up to 4 bits; ii) to detect all double random errors (we have considered this requirement because the probability of occurrence of three or more random errors is very low [7]). Thus, the word has to be divided in 4 interleaved sub-words. Now, each one is encoded with an ECC. A SEC code is not able to detect double errors, so we have opted for the SEC-DED extended (8, 4) Hamming code, allowing also better coverage against multiple random errors or longer bursts. The result is a 32-bit encoded word, where each bit of every sub-word is separated by 4 bits.

2.1 Error Coverage

The encoder and the decoder for the proposed code have been software implemented. The decoding process has been simulated for all possible error combinations, obtaining the percentages of detected and corrected errors, shown in Fig. 2. Considering

random errors (see Fig. 2(a)), only 1-bit errors are 100% corrected, and only 2-bit errors are 100% corrected or detected. Also, a high percentage of detection is achieved even for a big amount of random errors. This is one of the advantages of the interleaving approach: as there are different encoded sub-words, the detection in one of them is enough to arrive to the global error detection condition.

The potential of this approach can be appreciated in Fig. 2(b), where all possible burst errors are considered. As expected, all burst errors with length up to 4 are 100% corrected, and all of them with length up to 8 are 100% corrected or detected. In addition, the detection percentage is very high, as just commented, even for longer bursts. It is remarkable that 0% of errors are corrected under the presence of 5-bit or 9-bit burst errors. In a burst error, by definition, at least the first and the last bits are in error. In case of a 5-bit or a 9-bit burst error, those bits belong to the same sub-word (as the sub-words are interleaved), impeding the correction.

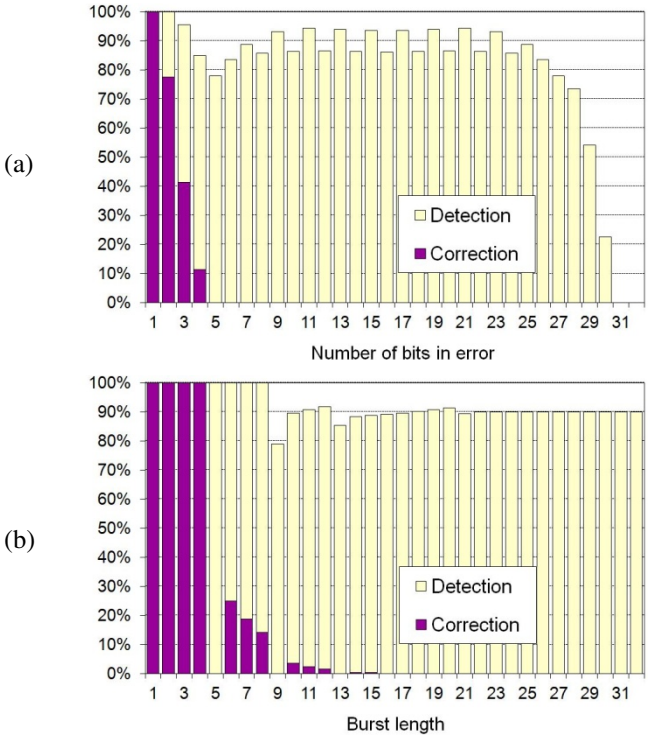


Fig. 2. Error coverage for IxH(32, 16) code: (a) random errors; (b) burst errors

2.2 Advantages and Drawbacks

IxH technique presents a remarkable property: the length of the tolerated burst errors can be configured during the design phase. The redundancy will depend on it, and on the length of each sub-word. The coverage obtained with this method may be enough for many applications. The actual redundancy required, or the power, area and

temporal overhead, will depend on both the code used and the implementation technology. The idea is to use simple codes (like Hamming) to keep these parameters low.

The main drawback is that only 1-bit random errors are 100% corrected. If higher level of correction or detection is required, more complex codes can be used, with the cost of greater redundancy.

3 Conclusions and Future Work

This paper presents the use of interleaved error correction codes to harden on-chip parallel interconnection lines. Interleaving gets good results under the presence of adjacent faults.

The resulting error detection and correction features, as well as the simplicity of the encoder and decoder circuits, will depend on the code to be interleaved. As an example, a SEC-DED (8, 4) extended Hamming code illustrates the implementation of the technique. The results show the expected coverage, as well as a very high percentage of error detection, which may be interesting in microprocessors with recovery mechanisms.

The implementation of the presented approach in the VHDL model of a microprocessor and its validation using simulated fault injection is the first step of our future work. In addition, a deeper analysis of the redundancy required, as well as the power, area and temporal overhead, must be done. Also, this technique should be compared with other existing proposals.

Acknowledgements. This work has been funded by the Spanish Government under the Research Project ARENES (TIN2012-38308-C02-01), and by the Universitat Politècnica de València under the Research Project DesTT (SP20120806).

References

1. Constantinescu, C.: Trends and challenges in VLSI circuit reliability. IEEE Micro 23(4), 14–19 (2003)
2. Wei, J., Rashid, L., Pattabiraman, K., Gopalakrishnan, S.: Comparing the effects of intermittent and transient hardware faults on programs. In: Dependable Systems and Networks Workshops, pp. 53–58 (2011)
3. Gil, P., et al.: Fault Representativeness. Deliverable ETIE2 of Dependability Benchmarking Project, IST-2000-25245 (2002)
4. Fujiwara, E.: Code Design for Dependable Systems. John Wiley & Sons (2006)
5. Constantinescu, C.: Intermittent faults and effects on reliability of integrated circuits. In: Reliability and Maintainability Symposium (RAMS 2008), pp. 370–374 (January 2008)
6. LaBel, K.A.: Proton single event effects (SEE) guideline. Submitted for Publication on the NASA Electronic Parts and Packaging (NEPP) Program web site (August 2009), `https://nepp.nasa.gov/files/18365/Proton_RHAGuide_NASAAug09.pdf`
7. Shamshiri, S., Cheng, K.T.: Error-Locality-Aware Linear Coding to Correct Multi-bit Upsets in SRAMs. In: IEEE International Test Conference, pp. 1–10 (November 2010)

csXception®: First Steps to Provide Fault Injection for the Development of Safe Systems in Automotive Industry

Ricardo Barbosa, Nuno Silva, and João Mário Cunha

Critical Software, S.A., Parque Industrial de Taveiro, Lote 48,
3045-504 Coimbra, Portugal
{rbarbosa,nsilva,jm-cunha}@criticalsoftware.com

Abstract. The increasing complexity on the vehicles electrical and/or electronic components has introduced a challenge to automotive safety. Standardization efforts have already been made, leading to the ISO-26262 functional safety and the AUTOSAR architecture definition, providing a development process that addresses safety and quality issues. With the goal of ensuring safety properties, this paper presents a fault injection tool (csXception®), developed by Critical Software, and the first steps towards injecting faults on ARM® Cortex-M3 microcontroller using the SCIFI technique for assessing AUTOSAR systems.

Keywords: Fault Injection, automotive safety, csXception®, ISO-26262, AUTOSAR, SCIFI.

1 Introduction

Nowadays, the percentage of the E/E (electrical and/or electronic components) embedded on automobiles can be up to 40% of the overall cost of a vehicle, and this value can be even bigger in luxury models. Cars contain on average 30 to 50 electronic control units (ECU), and although today's average cars contain about 10 million lines of code, it is expected that this number will grow to 300 millions in a decade. [1]

The new automotive standard, ISO-26262, defines stringent requirements in order to guarantee the dependability and quality of automotive systems. Fault injection techniques provide a way to cover and simulate the extreme/limit or abnormal cases, and this is recognized in the standard. A case study applied to AUTOSAR based architectures is considered appropriate for demonstration of these techniques.

2 State of the Art: Fault Injection

Fault Injection evaluates the dependability of a target system, studying generated errors and failures. In complex systems it's hard to understand what causes some error/failures, or how/where they begin [2]. Fault Injection deals with the calculated/controlled insertion of artificial faults into a target system, or a simulation of it. The most common techniques are described next.

M. Vieira and J.C. Cunha (Eds.): EWDC 2013, LNCS 7869, pp. 202–205, 2013.

- **Hardware Implemented Fault Injection (HWIFI):** uses extra hardware to inject faults on a target system. Example: Produce radiation on the target system.
- **Software Implemented Fault Injection (SWIFI):** uses software applications to inject and control fault injections on a target system**.** Example: change a register value during program execution.
- **Scan-Chain Implemented Fault Injection (SCIFI):** takes advantage of the boundary scan-chains and internal scan-chains present in target system providing more reachability, controllability and observability [3].
- **Robustness Fault Injection:** tests for abnormal inputs to reachable interfaces or function calls - oriented to a particular programming language. Example: inject possibly wrong values on methods/functions input values.

3 ISO-26262 Standard

The main purpose of ISO-26262 is to ensure the safety of road vehicles by providing a set of guidelines to help product development. ISO-26262 introduced the ASIL (Automotive Safety Integrity Levels) classification, based on the combination driver/other road user's probability of exposure to the hazards, as shown on Fig. 1.

Fig. 1. ASILS's risk estimation

Table 1 shows the ASIL levels for each test activity of the ISO-26262.

Table 1. Fault Injection mapping on ISO-26262 test activities

ISO-26262 test activities	ASILs
System Level (Part 4)	
Correctness of implementation of system design specification and technical safety requirements.	B, C, D
Effectiveness of diagnostic coverage of hardware fault detection mechanisms.	C, D
Correctness of implementation of system design specification, technical and functional safety requirements.	C, D
Effectiveness of diagnostic failure coverage of safety mechanisms at item level.	C,D
Correctness of implementation of functional safety requirements.	A, B, C, D
Effectiveness and failure coverage of safety mechanisms at vehicle level.	C, D
Hardware Level (Part 5)	
Hardware integration tests to verify completeness and correctness of safety mechanisms implementation with respect to hardware safety requirements.	C, D
Software Level (Part 6)	
Software unit testing	D
Software integration testing	C, D

From here one can conclude that fault injection is recommended for all the ISO-26262 test activities, namely for the ones with higher levels of criticality (C and D).

Every safety requirement is assigned an ASIL of the scale A, B, C or D, with D being the most safety-critical level. This functional safety standard divides the product development process in three main parts: system level integration (part 4), hardware development (part 5) and software development (part 6). ISO-26262 is the first standard to present fault injection as highly recommended technique to be used at different criticality levels. Depending on the level where it is used, the purpose differs.

4 The prototype: ARM® Cortex-M3 Plug-In

The application of the Cortex-M on automotive industry, particularly on systems with safety functions (airbag, anti-lock braking, etc.), is widespread. An example is the Toshiba electrical vehicle motor control system [6], implemented by means of ARM® Cortex-M3 microcontrollers and compliant with the ISO-26262 standard.

In order to inject faults one needs to define a fault model. It is an engineering model of erroneous events. Typically, fault models are defined considering four dimensions (Location, Duration, Trigger, Type), each one with its own characteristics. An example of the contents such a model for the ARM® Cortex-M3 is shown in Table 2.

Table 2. Cortex-M3 plug-in fault model example

Fault Id	Location	Duration	Trigger	Type
GPR1	General Purpose Register #1	One clock cycle	Instruction Execution	Bit-flip
PSR	Program Status Register	One clock cycle	Memory Access	Reset-value
OCF	On-chip Flash	One clock cycle	Timeout	Specific-value

5 Case Study: Fault Injection in AUTOSAR®

AUTOSAR® (AUTomotive Open System ARchitecture) is an architecture that provides a basic infrastructure to support the development of vehicular software, user interfaces and management for all the application domains. [5]

The main goal of this work is to provide possible fault injection approaches for the validation of AUTOSAR systems, under the goal of assessing the effectiveness of the safety mechanisms in place within the actual architecture.

The first fault injection approach will be to perform Software Component corruption**.** This mainly involves the corruption of key components by injecting faults on the AUTOSAR communication layers and forcing the breaking of communications with RTE (Runtime environment) component. Corrupting NVRAM manager, Watchdog manager or Communication Hardware abstraction will also be addressed.

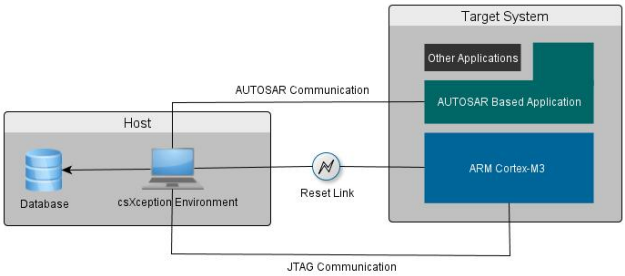


Fig. 2. Proposed csXception ARM® Cortex-M3 solution with AUTOSAR architecture

6 Conclusions and Future Work

Fault injection is an important tool to help ensuring compliance with ISO-26262. In the automotive market, AUTOSAR is becoming the "*de facto*" architecture for developing components. Though both explicitly mention fault related approaches, neither of the standards details the recommended fault injection approach to be used.

Future work on this subject includes the completion of the failure modes and fault model, along with the integration of these concepts within the ARM® Cortex-M3 fault injection prototype. For this, knowledge exchange is foreseen with several automotive OEMs that will provide the required know-how, thus, helping in building a realistic and accurate fault model for the automotive domain.

The csXception® tool will be upgraded and a relevant fault model will be defined for usage by the research community. Currently, a study is being done by Critical Software and University of Coimbra in order to identify the key research groups and researches in fault injection and dependability.

Acknowledgments. This work has been partially supported by the project CECRIS (Certification of CRItical Systems, http://www.cecris-project.eu), Marie Curie Industry-Academia Partnerships and Pathways (IAPP) number 324334 (FP7).

References

1. The Economist, Tech. View: Cars and software bugs, `http://www.economist.com/blogs/babbage/2010/05/techview_cars_and_software_bugs` (visited on: April 09, 2013)
2. Hsueh, M.C., Tsai, T.K., Iyer, R.K.: Fault injection techniques and tools. IEEE Computer 30(4), 75–82 (1997)
3. Ziade, H., Ayoubi, R., Velazco, R.: A survey on fault injection techniques. The International Arab Journal of Information Technology 1(2), 171–186 (2004)
4. ISO International Standard, ISO-26262: Road vehicles – Functional safety
5. AUTOSAR, `http://www.autosar.org` (visited on: February 11 2013)
6. Toshiba, Automotive Cortex M3 Line-up, `http://www.toshiba-components.com/automotive/autocortexm3.html` (visited on: February 15, 2013)

Author Index