DEPEND 2013

The Sixth International Conference on Dependability

ISBN: 978-1-61208-301-8

August 25-31, 2013

Barcelona, Spain

DEPEND 2013 Editors

Pascal Lorenz, University of Haute Alsace, France

DEPEND 2013

Foreword

The Sixth International Conference on Dependability [DEPEND 2013], held between August 25-31, 2013 in Barcelona, Spain, provided a forum for detailed exchange of ideas, techniques, and experiences with the goal of understanding the academia and the industry trends related to the new challenges in dependability on critical and complex information systems.

Most of critical activities in the areas of communications (telephone, Internet), energy & fluids (electricity, gas, water), transportation (railways, airlines, road), life related (health, emergency response, and security), manufacturing (chips, computers, cars) or financial (credit cards, on-line transactions), or refinery& chemical systems rely on networked communication and information systems. Moreover, there are other dedicated systems for data mining, recommenders, sensing, conflict detection, intrusion detection, or maintenance that are complementary to and interact with the former ones.

With large scale and complex systems, their parts expose different static and dynamic features that interact with each others; some systems are more stable than others, some are more scalable, while others exhibit accurate feedback loops, or are more reliable or fault-tolerant.

Inter-system dependability and intra-system feature dependability require more attention from both theoretical and practical aspects, such as a more formal specification of operational and non-operational requirements, specification of synchronization mechanisms, or dependency exception handing. Considering system and feature dependability becomes crucial for data protection and recoverability when implementing mission critical applications and services.

Static and dynamic dependability, time-oriented, or timeless dependability, dependability perimeter, dependability models, stability and convergence on dependable features and systems, and dependability control and self-management are some of the key topics requiring special treatment. Platforms and tools supporting the dependability requirements are needed.

As a particular case, design, development, and validation of tools for incident detection and decision support became crucial for security and dependability in complex systems. It is challenging how these tools could span different time scales and provide solutions for survivability that range from immediate reaction to global and smooth reconfiguration through policy based management for an improved resilience. Enhancement of the self-healing properties of critical infrastructures by planning, designing and simulating of optimized architectures tested against several realistic scenarios is also aimed.

To deal with dependability, sound methodologies, platforms, and tools are needed to allow system adaptability. The balance dependability/adaptability may determine the life scale of a complex system and settle the right monitoring and control mechanisms. Particular challenging issues pertaining to context-aware, security, mobility, and ubiquity require

appropriate mechanisms, methodologies, formalisms, platforms, and tools to support adaptability.

We take here the opportunity to warmly thank all the members of the DEPEND 2013 Technical Program Committee, as well as the numerous reviewers. The creation of such a high quality conference program would not have been possible without their involvement. We also kindly thank all the authors who dedicated much of their time and efforts to contribute to DEPEND 2013. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations, and sponsors. We are grateful to the members of the DEPEND 2013 organizing committee for their help in handling the logistics and for their work to make this professional meeting a success.

We hope that DEPEND 2013 was a successful international forum for the exchange of ideas and results between academia and industry and for the promotion of progress in the field of dependability.

We are convinced that the participants found the event useful and communications very open. We hope Barcelona provided a pleasant environment during the conference and everyone saved some time for exploring this beautiful city.

DEPEND 2013 Chairs:

DEPEND Advisory Chairs
Reijo Savola, VTT Technical Research Centre of Finland, Finland
Sergio Pozo Hidalgo, University of Seville, Spain
Manuel Gil Perez, University of Murcia, Spain
Petre Dini, Concordia University, Canada / China Space Agency Center - Beijing, China

DEPEND 2013 Industry Liaison Chairs
Piyi Yang, Wonders Information Co., Ltd., China
Timothy Tsai, Hitachi Global Storage Technologies, USA

DEPEND 2013 Research/Industry Chair
Michiaki Tatsubori, IBM Research Tokyo, Japan

DEPEND 2013 Special Area Chairs
Cross-layers dependability
Szu-Chi Wang, National Ilan University, Taiwan
Hardware dependability
Peter Tröger, Hasso Plattner Institute / University of Potsdam, Germany
Empirical assessments
Marcello Cinque, University of Naples Federico II, Italy
Security and Trust
Syed Naqvi, CETIC, Belgium

DEPEND 2013

Committee

DEPEND Advisory Chairs

Reijo Savola, VTT Technical Research Centre of Finland, Finland
Sergio Pozo Hidalgo, University of Seville, Spain
Manuel Gil Perez, University of Murcia, Spain
Petre Dini, Concordia University, Canada / China Space Agency Center - Beijing, China

DEPEND 2013 Industry Liaison Chairs

Piyi Yang, Wonders Information Co., Ltd., China
Timothy Tsai, Hitachi Global Storage Technologies, USA

DEPEND 2013 Research/Industry Chair

Michiaki Tatsubori, IBM Research Tokyo, Japan

DEPEND 2013 Special Area Chairs

Cross-layers dependability
Szu-Chi Wang, National Ilan University, Taiwan

Hardware dependability
Peter Tröger, Hasso Plattner Institute / University of Potsdam, Germany

Empirical assessments
Marcello Cinque, University of Naples Federico II, Italy

Security and Trust
Syed Naqvi, CETIC, Belgium

DEPEND 2013 Technical Program Committee

Muhammad Afzaal, University of Naples "Parthenope", Italy
Jose Ignacio Aizpurua Unanue, University of Mondragon, Spain
Murali Annavaram, University of Southern California, USA
Afonso Araújo Neto, University of Coimbra, Portugal
José Enrique Armendáriz-Iñigo, Universidad Pública de Navarra, Spain
Steffen Bartsch, TZI - Universität Bremen, Germany
Ian Bayley, Oxford Brookes University, U.K.
Jorge Bernal Bernabé, University of Murcia, Spain
James Brandt, Sandia National Laboratories, U.S.A.

Mikel Larrea, University of the Basque Country - UPV/EHU, Spain
Inhwan Lee, Hanyang University - Seoul, Korea
Matthew Leeke, University of Warwick, UK
Jane Liu, Taiwan Academia Sinica, Taiwan
Yun Liu, Boeing Company, USA
Paolo Lollini, University of Firenze, Italy
Miroslaw Malek, Humboldt-Universitaet zu Berlin, Germany
Amel Mammar, Mines Telecom/ Telecom SudParis, France
Antonio Mana Gomez, University of Malaga, Spain
Rivalino Matias Jr., Federal University of Uberlandia, Brazil
Yutaka Matsuno, Nagoya University, Japan
Manuel Mazzara, Newcastle University, UK / UNU-IIST, Macau
Per Håkon Meland, SINTEF ICT, Norway
Carlos Julian Menezes Araujo, Federal University of Pernambuco, Brazil
Francesc D. Muñoz-Escoí, Universitat Politècnica de València, Spain
Jogesh K. Muppala, The Hong Kong University of Science and Technology, Hong Kong
Jun Na, Northeastern University, China
Syed Naqvi, CETIC, Belgium
Sarmistha Neogy, Jadavpur University, India
Mats Neovius, Åbo Akademi University - Turku, Finland
Hong Ong, e-Manual System Sdn Bhd, Malaysia
Frank Ortmeier, Otto-von-Guericke-Universität Magdeburg, Germany
Roberto Palmieri, ECE Department/Virginia Tech, U.S.A.
Aljosa Pasic, ATOS Origin, Spain
Karthik Pattabiraman, University of British Columbia, Canada
Alfredo Pironti, INRIA Paris Rocquencourt, France
Wolfgang Pree, University of Salzburg, Austria
Rolf Riesen, IBM Research, Ireland
Paolo Romano, INESC-ID/IST, Portugal
Christian Rossow, VU University Amsterdam, Netherlands
Francesca Saglietti, University of Erlangen-Nuremberg, Germany
Felix Salfner, SAP Innovation Center - Potsdam, Germany
Reijo Savola, VTT Technical Research Centre of Finland, Finland
Sahra Sedighsarvestani, Missouri University of Science and Technology, U.S.A.
Jean-Pierre Seifert, Technische Universität Berlin & Telekom Innovation Laboratories, Germany
Dimitrios Serpanos, University of Patras & ISI, Greece
Navjot Singh, Avaya Labs Research, USA
Komminist Sisai, Fondazione Bruno Kessler, Italy
Alessandro Sorniotti, IBM research - Zurich, Switzerland
George Spanoudakis, City University London, U.K.
Kuo-Feng Ssu, National Cheng Kung University, Taiwan
Vladimir Stantchev, Berlin Institute of Technology, Germany
Neeraj Suri, TU-Darmstadt, Germany
Kenji Taguchi, National Institute of Advanced Industrial Science and Technology (AIST), Japan
Oliver Theel, University Oldenburg, Germany Sergio Pozo Hidalgo, University of Seville, Spain
Kishor Trivedi, Duke University - Durham, USA
Peter Tröger, Hasso Plattner Institute / University of Potsdam, Germany
Elena Troubitsyna, Aabo Akademi -Turku, Finland

Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission or reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article is does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

Table of Contents

Estimation of Performance and Availability of Cloud Application Servers through External Clients

Sune Jakobsson

Department of Telematics
NTNU

Trondheim, Norway

sune.jakobsson@comoyo.com

***Abstract*— This paper investigates two aspects of the QoS offered by some cloud providers on the Internet, the availability and the dedicated capacity in terms of how well a user process is isolated from other users of the same application server instance. By using standard components and software utilities, a small external measurement client is able to gather the necessary information about the cloud application servers in question, and hence, addresses the measured availability and capacity over time. In summary, I show that the biggest cloud providers indeed demonstrate good isolation and availability.**

Keywords-component; QoS; Java virtual machines; garbage collection; application servers; availability

I. INTRODUCTION

More and more of our computing needs are being moved to the "cloud", but what does this really mean with respect to dedicated capacity, availability, and reliability? The terms used in this paper are defined in [6]. This paper describes a limited experiment with cloud providers on the Internet that commercially provides application server instances, with the objective to investigate the quality of service one can observe from these providers with a client connected to multiple providers as shown in Figure 1. The focus in this paper is on application availability, and to some extent dedicated capacity. Some major providers offer application server instances for free for a limited period of time and others provide them at a reasonable cost. None of the providers provide detail regarding their availability or their internal structure. The best one can find ahead of signing up are the relative up-time during the last period often without stating what the period is. So how can one assess their offer in reasonable time and at a reasonable cost?

Several papers point out failures on servers present on the Internet, [2, 10]. There are several papers addressing the isolation among virtual machines, [11], but in these particular cases there is no prior knowledge available of the underlying system, and hence, a different approach is needed. The approach chosen here is to probe live application servers, and collect externally available data from their operation. This paper presents the approach, and discusses what is possible to observe from the outside of the application server providers.

By analyzing just a few isolated parameters, one may yield significant conclusions regarding their behavior. In other papers related to this subject [1, 13], it is shown that it is sufficient to observe the response time and the amount of free memory in the application container to experimentally predict their long term behavior.

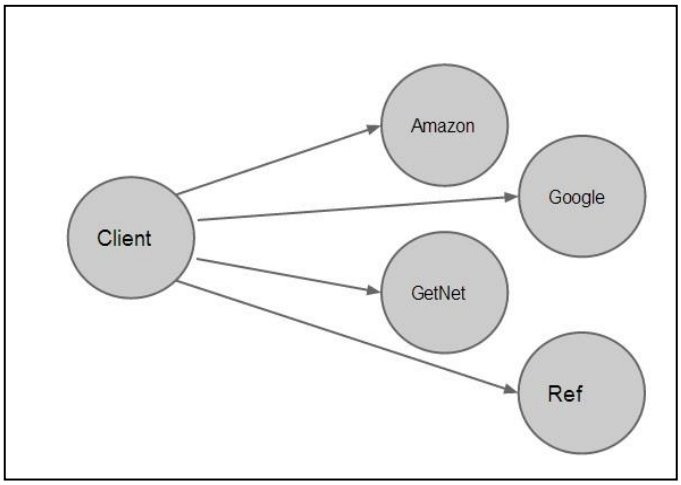


Figure 1. The measurement system.

For the current experiment, a simple client and server instance is developed and run for more than a month. The external client, in turn, invokes four application instances at different nodes, i.e., virtual machines provided as cloud services. All virtual machines are running with the same server software. An application instance is defined as an application container running the application software in a virtualized environment. The client measures the response time of the invocation and logs the amount of free memory reported by the respective application servers. The operating systems provide a millisecond clock on the client side for time stamps, and by choice the trial was run over a period of one month, sampling the unavailability at 2.5 second intervals to obtain enough samples to validate the claims. The amount of free memory from the servers is reported in bytes, and varies from instance to instance, but their overall behavior is similar. In Section II the measurement system is described. Section III gives some background on memory allocation. The experiment and the kind of results obtained are presented in Section IV. Finally, the results are presented with a discussion related to the offerings from the vendors in Sections V.

 ISBN: 978-1-61208-301-8

II. THE MEASUREMENT SYSTEM

One key element was to investigate cloud based servers and to see if the application server instances the providers provide are indeed isolated from other usage or not. Since Amazon EC2 [4] and Google App-server [3] are two big players in this domain, they were selected. To have some comparison and base line, a smaller vendor and a personal reference server connected to the Internet was part of the experiment as shown in Figure 1. All invocations were done from the same place, to avoid or filter out close and near network issues. With the chosen invocation rate of 2.5 seconds, unavailability of less than 2.5 seconds is not detected, see Figure 2. The client logs the result code of the invocation, the invocation duration time, and the reported server side free memory.

The client and the servers running on the virtual machines were all programmed in Java [7].

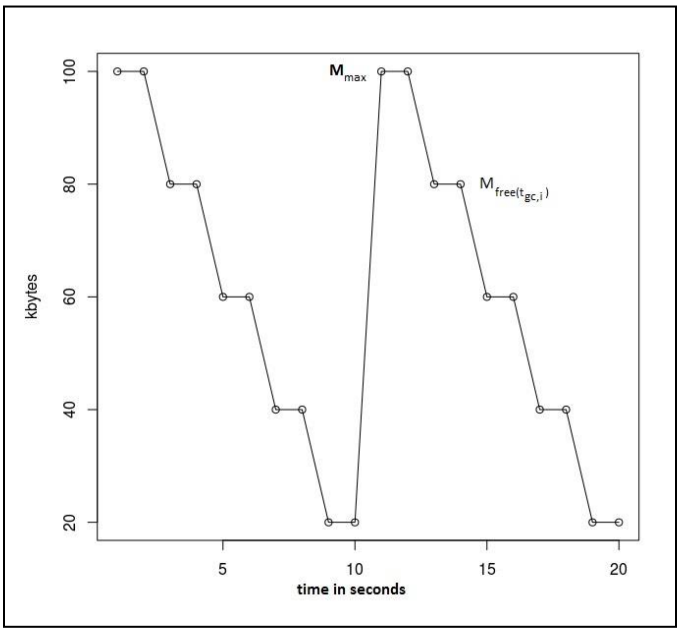


Figure 2. Amount of free memory.

Some of the cloud providers use intrusion detection systems, and to avoid having a client black listed when the server is unavailable, a back-off mechanism is included in the client. This is implemented so that if there are more than a fixed number of non-finished requests, the client waits until the server responds or the network connection times out.

III. DEDICATED CAPACITY

Each server allocates an amount of memory to process the request and in this context the amount is fixed and only dependent on the server side implementation. In this experiment the amount of free memory is logged by the client each time a server is invoked, typically resulting in steps of approximately 10k bytes as shown in Figure 2. In an idealized instance the amount of free memory would then produce a downward-stepping shaped curve until the garbage collector runs and restores the curve to the "max" value, or what memory it is able to free up, as shown in Figure 2. This curve can be modeled as follows. Let us assume the garbage collector runs at instances$t_{gc,1}$,…, $t_{gc,i}$,… and let M_{max} be the memory available to the virtual machine immediately after the garbage collection is finished. The free memory available to the virtual machine can then be expressed as

$$M_{free(t_{gc,i})} = M_{max} \quad \text{and} \tag{1}$$

$$M_{free}(t_{gc,i}) = M_{max} - m \int_{t_{gc,i}}^{t} p(t)dt, t_{gc,1} < t \leq t_{gc,i+1} \tag{2}$$

Where p(t) is the instant load on the virtual machine at time *t*. For equidistant and constant invocations, the result is an almost linear behavior as shown in Figure 2.

In a server that is little used, the period between garbage collections is long compared to the invocation time, and when plotted on a curve it will show a downward stepping function until the minimum memory point is reached and the garbage collector recovers the memory at which point it starts at the "M_{max}" point again. If one removes the garbage collection steps and only looks at the slope of the steps there is a correlation between how often the server is invoked and how much memory is used for each invocation. This correlation is then proportional to the load that the server process processes. If the load is constant then the slope is only dependent on the invocation frequency and if the slope is constant this implies that there are no others invoking the server in that same time period. Assuming that there is little amount of processing in the request, the requests will be memory intensive and show the amount of available memory at any time.

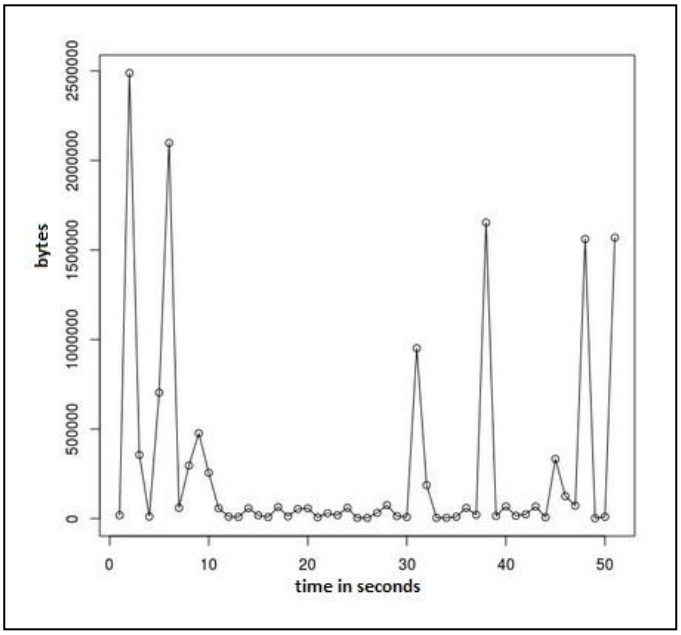


Figure 3. Distorted delta free memory

The time between when the garbage collector is run then depends on the amount of initial memory and the load on the server. The higher the load on the server, the more memory is allocated, and the more often the garbage collector runs [8]. Also if there are other users of the same application server instance the relatively modest usage of the application server is then cluttered by other invocations and the smooth downward-stepping pattern is not observed. Figure 3 shows a delta free memory plot that is showing distorted available free memory.

If one plots the memory usage over a long period, as in Figure 4, one gets a macro view of the data, where the amount of free memory shows up as a fat black line on the graph, this is due to the number of samples, but if the time axis is expanded, the figure would show a pattern as in Figure 2. In Figure 4, the invocation times are shown in green, however, the details get lost in the macro view except in the cases where the server does not respond and the free memory sample is unavailable, and hence, is zero.

The minimum invocation time illustrates the physical transport times and the clustering of these indicate that the route the requests took was the same over longer periods of time which establishes the pattern seen in the Figure 4.

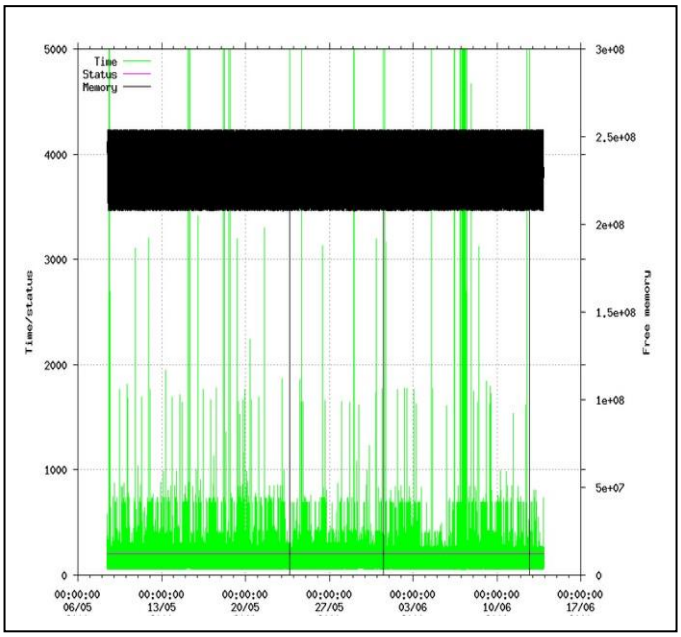


Figure 4. Macro view of the collected data from a Amazon EC2 instance.

By sorting the green invocation times in descending order and plotting them according to their value as shown in Figure 5, the different invocation times appear in better detail and the different "plateaus" seen in Figure 4 are recognized. Note the logarithmic scales. Where the red horizontal and vertical lines cross, the lower right quadrant contains the "successful" invocations meeting the requirement of a response within 5 seconds.

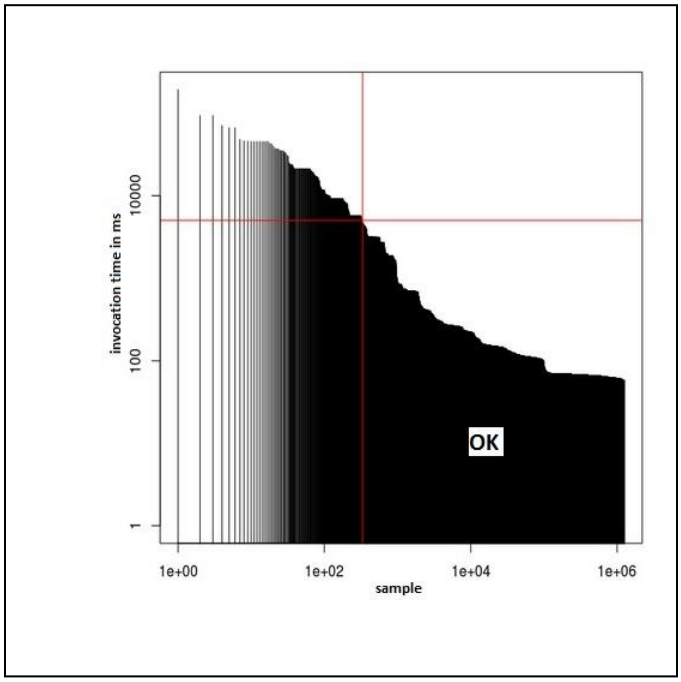


Figure 5. Sorted invocation times from a Amazon EC2 instance.

IV. INVOCATION TIME

When a server is invoked the invocation consists of multiple parts:

1. Connection set-up time
2. Request transport time
3. Request acknowledge time
4. Processing time
5. Response transport time
6. Response acknowledge time

In this experiment the client is the same running instance sending requests in parallel to all servers. The amount of data in the request and the response is small enough to fit into a single transport unit, so there is no consideration of reassembly of transport packets in this experiment. To avoid blacklisting by intrusion detection systems, a back-off mechanism was implemented, so that in the cases where a server ceases to respond, no more than 5 new requests were issued before responses started returning, hence, effectively changing the sampling interval until the server starts responding again.

Note that if the client in question was a person interacting with a graphical user interface and consuming a service and the invocation takes more than 5 seconds their interpretation of the situation would be that something may have gone wrong.

The invocation times were measured with the internal system clock with millisecond accuracy. The invocation

 ISBN: 978-1-61208-301-8

times are typically one to a few hundred milliseconds range as shown in Figure 5.

V. AVAILABILITY STATES

The invocations are logged on the client side and they are categorized using the following states depending on their outcome.

1. Invocation succeeded (***Ok***).
2. Invocation succeeded, but the invocation took more than 5 seconds (***Slow***).
3. Invocation failed, and connection was established, but a timeout occurred when reading or writing the data to the client (***Time-out***).
4. Invocation failed due to lack of connectivity, no route to server was available (***No conn.***).
5. Invocation of server was not started since there are currently more than five outstanding unanswered requests in a row (***Un-answer.***).
6. Skipped requests due to errors in a row (***Skip***).

In order to avoid the black-listing of the client, the client stops sending new requests when the server does not respond in a timely manner. These impacts the availability in state 5 by staying in that state until the issued request receives a response or the network connection times out. A consequence of this is that the server or network might have recovered earlier than the built-in retry times at the TCP transport layer. The results of the invocation per server are summarized in Table I.

TABLE I. STATES OBSERVED BY INVOCATIONS

Servers	Availability states					
	1. Ok	*2. Slow*	*3. Tim-out*	*4. No Conn.*	*5. Un-answer.*	*6. Skip*
Amazon EC2	1259371	326	7	10	170	89
Google App. Engine	1258780	1167	471	36	761	3319
GetNet	1258890	2288	380	17	651	5320
Reference system	1259191	320	25	17	350	901

VI. OPERATIONS

The client was deployed in the NTNU network and was run for a period of over a month. Initially some adjustments had to be made on the client to avoid getting blacklisted on intrusion detection systems if one issued request towards a server that was taken down for maintenance. All but one of the providers provided continuous service, while one took a restart every night in order to restore the system to a known state, also called software rejuvenation [5, 9]. This resulted in a down-period, at a fixed time every night (2AM), however, this is not mentioned anywhere in the terms or conditions for their site. The biggest players announce their target figures for the availability of services and instances. In the case of not meeting their targets, customers may get credits for future free usage. Amazon states that they will use commercially reasonable efforts to make Amazon EC2 available with an "Annual Uptime Percentage" of at least 99.95% during the Service Year. "Annual Uptime Percentage" is calculated by subtracting from 100% the percentage of 5 minute periods during the Service Year in which Amazon EC2 was in the state "Region Unavailable". The Google Apps Covered Services web interface will be operational and available to Customer at least 99.9% of the time in any calendar month. Since the experiment is run over a one month time the results show that they are indeed close to their targets, where Amazon EC2 availability is conservative, however, it is difficult to predict how much the results would differ over a longer period of time.

VII. CONCLUDING REMARKS

As this simple experiment indicates, the cloud providers may provide application servers with similar or better availability than what one can obtain with a traditional non-redundant approach using standard of the shelf hardware and software. Using the results for "Ok" state in Table I, as available, the availability is calculated and the results are shown in Table II.

TABLE II. AVAILABILITY SUMMARIZED

Servers	Monthly values		
	Announced Availability	*Observed Availability*	*Accumulated obs. downtime in minutes*
Amazon EC2	0.9995	0.999522	25
Google App. Eng.	0.999	0.995432	239
GetNet	No info	0.993128	360[a]
Reference system	No info	0.998719	67

a. Nightly restarts.

The reference system in this experiment is a standard PC running Ubuntu operating system [12] and connected to the NTNU campus network.

In summary the big players announce the same numbers for availability as obtained in this experiment.

Concerning how well an instance in the "cloud" is really isolated from other instances, if no prior information exists, is possibly by the limited measurements, as described in this paper. One can observe if the instance is alone or disturbed by other usage from that provider. By comparing the macro results with the micro results it is then possible to assess the offer at hand and make qualified choices regarding the cloud providers in question.

By looking at details of invocation times and amount of free memory at both macro and micro levels, different types of information emerges, to support decisions on scaling and availability.

 ISBN: 978-1-61208-301-8

VIII. FURTHER WORK

Given the dynamic nature of cloud computing and the possibility to both scale up and scale down by requesting more or bigger instances from the cloud providers, finding a simple means to detect when this should be done is of economic interest for the users, however, starting new instances of servers requires some startup time, and finding good predictors on when new instances are required, or when redundant instances can be stopped and shut down, are still an issue for further work.

ACKNOWLEDGMENT

I would like to thank Professor Rolv Bræk and Professor Bjarne E. Helvik at Department of Telecommunication at NTNU, for their advice and guidance in my research work, and my wife for proof reading my papers.

REFERENCES

[1] S. Jakobsson "Timing Failures Caused by Resource Starvation in Virtual Machines ", DEPEND 2011, ISBN: 978-1-61208-149-6

[2] Ryan K.L. Ko, Stephen S.G. Lee and Veerappa Rajan, "Understanding cloud failures", Spectrum, IEEE , vol.49, no.12, pp.84,84, December 2012 doi: 10.1109/MSPEC.2012.6361788

[3] https://code.google.com/status/appengine/ (last seen June 2013)

[4] http://aws.amazon.com/ec2/ (last seen June 2013)

[5] W. G. Bouricius, W. C. Carter and P. R. Schneider, "Reliability Modelling Techniques for Self-Repairing Computer Systems" Proceedings 24th National Conference ACM, 1969.

[6] A. Avižienis , J. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," IEEE Trans. Dependable and Secure Computing, vol. 1,no. 1,pp. 11-33, Jan.-Mar. 2004.

[7] J. Engel: "Programming for the Java Virtual Machine", Addison-Wesley, 1999. ISBN 0-201-30972-6

[8] R. Jones, "The Garbage Collection Page", http://www.cs.kent.ac.uk/people/staff/rej/gc.html (last seen June 2013)

[9] Huang, Yennun, et al. "Software rejuvenation: Analysis, module and applications." Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium on. IEEE, 1995.

[10] S. Pertet and P. Narasimhan, "Causes of Failure in Web Applications (CMU-PDL-05-109)". Parallel Data Laboratory. Paper 48. http://repository.cmu.edu./pdl/48 (last seen June 2013)

[11] http://en.wikipedia.org/wiki/Temporal_isolation_among_virtual_machines (last seen June 2013

[12] http://www.ubuntu.com (last seen June 2013)

[13] S. Jakobsson, "A Token Based Approach Detecting Downtime in Distributed Application Servers or Network Elements", Networked Services and Applications - Engineering, Control and Management, 16th EUNICE/IFIP WG 6.6 Workshop, EUNICE 2010, Trondheim, Norway, June 28-30, 2010. ISBN 978-3-642-13970-3 Proceedings, pp. 209-216

 ISBN: 978-1-61208-301-8

Byzantine Self-Stabilizing Clock Distribution with HEX: Implementation, Simulation, Clock Multiplication

Martin Perner, Martin Sigl, Ulrich Schmid
Vienna University of Technology
Vienna, Austria
{mperner,msigl,s}@ecs.tuwien.ac.at

Christoph Lenzen
Massachusetts Institute of Technology
Cambridge, MA, USA
clenzen@csail.mit.edu

***Abstract*—We present a prototype implementation and simulation-based evaluation of a recently proposed novel approach for Byzantine fault-tolerant and self-stabilizing clock distribution in multi-synchronous GALS architectures. Fault-tolerant clock generation and clock distribution is a mandatory prerequisite for highly dependable multicore processors and Systems-on-Chip, as it removes the single point of failure typically created by central oscillators and conventional clock distribution trees. Our scheme, termed HEX, is based on a hexagonal grid topology, which connects simple intermediate nodes implemented using the UMC 90 nm standard cell library. Their purpose is to (i) forward synchronized clock signals throughout the grid and (ii) to supply the clock to nearby application modules. To achieve (ii), we show how to construct a *fast clock* on top of the clock signal provided by HEX and analyze its properties. In sharp contrast to existing solutions, HEX is not only Byzantine fault-tolerant, but also self-stabilizing, i.e., it can recover from arbitrarily corrupted system states. ModelSim-based simulation experiments confirm the excellent performance and fault-tolerance properties of our approach achieved in practice, which were already suggested by an earlier theoretical worst-case analysis.**

Keywords*—*fault-tolerance; self-stabilization; clock distribution; fault-injection; simulation analysis

I. INTRODUCTION

Thanks to the advances in *very large scale integration* (VLSI) technology, which nowadays allows clock speeds in the GHz range, complex hardware architectures such as multi-core processors and *Systems-on-Chip* (SoC) typically comprise multiple clock domains: Individual system components (e.g., a single core) execute synchronously, driven by a common clock signal. Different components may reside in different clock domains, which, however, are driven by clock signals generated by multiple unsynchronized clock sources. Communication between components in different domains must hence be synchronized or performed asynchronously, as in classic *globally asynchronous locally synchronous architectures* (GALS) [1].

This material is based upon work supported by the National Science Foundation under Grant Nos. CCF-AF-0937274, CNS-1035199, 0939370-CCF and CCF-1217506, the AFOSR under Contract No. AFOSR Award number FA9550-13-1-0042, the Swiss Society of Friends of the Weizmann Institute of Science, the German Research Foundation (DFG, reference number Le 3107/1-1), and the Austrian Science Foundation (FWF) project FATAL (P21694).

Multisynchronous GALS [2][3] architectures assume bounded synchrony (a maximum skew of a few clock cycles) also between different clock domains. This *mesochronous* clocking [4] not only allows to build a global notion of time throughout the chip, which, e.g., facilitates time-triggered transmission scheduling/routing in *Networks-on-Chip* (NoC) like Aelite [5], but also enables metastability-free high-speed cross-domain communication via FIFO buffers [6][7].

As mesochronous clocking inherently facilitates *distributed* clock generation, it also allows to address the lacking robustness of conventional centralized clocking approaches: If the central oscillator or some wire in the clock tree (used for distributing the clock signal to the functional units on the chip) breaks in such an architecture, even replicated functional units are of no use. This is also true for the few non-fault tolerant approaches for distributed mesochronous clock generation [8][9][10][11][12][13] described in literature, which either use distributed ring oscillators or distributed *phase-locked loops* (PLL). The only fault-tolerant clock generation approaches we are aware of are our Byzantine fault-tolerant DARTS [14][15] and our self-stabilizing Byzantine fault-tolerant FATAL approach [16]. However, both approaches focus on the distributed generation of a small number of synchronized clock signals and do not address the question of how to distribute these to a large number of functional units.

In [17], we proposed a novel approach, termed HEX, for reliably distributing a synchronized clock signal in multi-synchronous GALS systems. It works with arbitrary clock sources, ranging from a central oscillator to a fully distributed synchronized clock generation scheme like DARTS or FATAL. HEX is based on a sufficiently connected wiring topology, namely, a *hexagonal grid*. Intermediate nodes placed at each grid point control when the clock signal transitions are forwarded to adjacent nodes, and supply the clock signal to the functional units in their vicinity via small local clock trees. This way, HEX ensures that the clock signals at physically close nodes are well-synchronized throughout the chip.

The analytical worst-case analysis presented in [17] revealed excellent synchronization and fault-tolerance properties. In particular, in sharp contrast to the alternative approaches [8][9][10][11][12][13], HEX supports multiple synchronized clock sources, tolerates Byzantine failures of both clock

 ISBN: 978-1-61208-301-8

sources and nodes as well as link failures, and self-stabilizes [18] quickly from arbitrary states, as, e.g., caused by an overwhelming number of transient failures. Its resilience to failures scales with the size of the grid, in the sense that it tolerates a constant density of isolated Byzantine nodes; it can handle an even larger number of more benign failures like broken wires and mute clock sources and nodes. Moreover, HEX obeys a fault locality property: The adverse effect of failures in the grid on the clock skew between non-faulty neighbors decreases with the distance.

Contributions: The present paper provides the first step towards a physical implementation of HEX and complements the theoretical analysis provided in [17]:

(i) We show how the HEX nodes can be implemented and synthesized using the UMC (United Microelectronics Corporation) 90 nm standard cell library.

(ii) We study the average-case performance of our HEX implementation, in particular, in the presence of faults, using ModelSim-based simulation and fault-injection in a custom testbed. Our results confirm that the quite "exotic" worst-case scenarios identified in [17] are extremely unlikely to occur in practice even in the presence of failures; note that the MATLAB-simulations described in [17] have only been devoted to the fault-free case.

Moreover, we address the question of how to utilize the synchronized clock generated by HEX in applications. We consider a high-speed point-to-point communication subsystem for this purpose, which is, e.g., instrumental for multi-hop communication between HEX nodes and thus for any higher-level communication service. It primarily requires a synchronized clock with high frequency (to facilitate high-speed communication) and small clock skew (to avoid large data buffers). The last major contribution of our paper is hence the following:

(iii) We show how to augment HEX nodes to generate a synchronized *fast clock*, and analyze its skew.

Our paper is organized as follows: Section II provides an overview of HEX and some results from our previous work [17]. All subsequent sections present novel results: In Section III, we briefly describe how to add a particular communication subsystem to the HEX nodes, and provide the resulting requirements for the fast clock. Section IV gives an overview of our HEX implementation and the features of our custom testbed, which has been used to obtain the results described in Section V. Finally, Section VI is devoted to the implementation and analysis of a fast clock suitable for our communication subsystem. Some conclusions and directions of further research in Section VII round off our paper.

II. Overview of HEX

HEX [17] assumes a system consisting of (simple) computing nodes that exchange zero-bit messages (i.e., the only information they contain is their occurrence time) over a directed cylindrical hexagonal grid $G = (V, E)$ defined as follows (see Figure 1): With $L \in \mathbb{N}$ denoting its length and $W \in \mathbb{N}$ its width, the set of nodes V is the set of tuples $(\ell, i) \in [L+1] \times [W]$. Herein, $[L+1] := \{0, \dots, L\}$ denotes the row index set, referred to as *layers*, and $[W] = \{0, \dots, W-1\}$ the column index set, referred to as *columns*, of the nodes in the grid. For each node $(\ell, i) \in V$, $\ell \in [L+1]$, $i \in [W]$, the following links are in E: (i) Incoming and outgoing links to neighboring nodes of the same layer, namely from (ℓ, i) to $(\ell, i-1 \bmod W)$, called the left neighbor of (ℓ, i), and to $(\ell, i+1 \bmod W)$, called the right neighbor, and vice versa from the left and the right neighbor to (ℓ, i); (ii) if (ℓ, i) is in a layer greater than 0, incoming links from $(\ell-1, i)$, called its lower left neighbor, and $(\ell-1, i+1 \bmod W)$, called its lower right neighbor; (iii) if (ℓ, i) is in a layer smaller than L, outgoing links to $(\ell+1, i-1 \bmod W)$, its upper left neighbor, and $(\ell+1, i)$, its upper right neighbor.

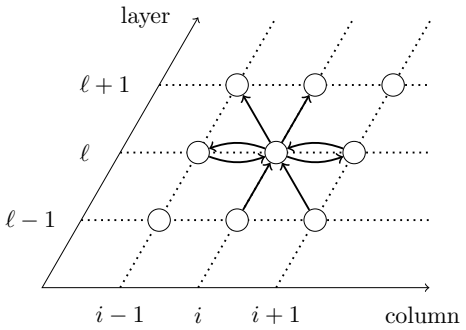


Fig. 1. Node (ℓ, i) and its incident links in the cylindrical hexagonal grid topology. Columns are modulo W and layers (rows) are between 0 and L.

1: **once** received trigger messages from (left and lower left) or (lower left and lower right) or (lower right and right) neighbors **do**
2: broadcast trigger message; ▷ local clock pulse
3: sleep for some time within $[T^-, T^+]$;
4: forget previously received trigger messages;

Fig. 2. Pulse forwarding algorithm for nodes in layer $\ell > 0$.

The failure model of HEX assumes that every node has at most one Byzantine faulty neighbor on an incoming edge, which can exhibit arbitrary behavior. However, a faulty node must not be able to prevent a correct neighbor from triggering correctly on behalf of its other neighbors. Hence, it is not allowed to output excessive voltages that destroy the receiver, or cause metastability that also upsets the receiver, for example. Note that tolerating two or more faulty neighbors would require an in-degree of at least 5, resulting in a non-planar communication graph due to the directed propagation of pulses utilized in HEX. All fault-free links in the graph respect FIFO order, with end-to-end delays that may vary non-deterministically within $[d^-, d^+] \subseteq (0, \infty)$.

Nodes at layer 0 execute a clock pulse generation algorithm like the ones of [15][16], whose purpose is to generate synchronized and well-separated initial messages (i.e., pulses). Nodes at layers larger than 0 run the simple pulse forwarding algorithm specified in Figure 2. Basically, nodes forward pulse k once they received trigger messages for pulse k from two adjacent neighbors. Therefore, synchronized pulses generated by the layer 0 nodes propagate as "waves" through the HEX grid up to the very last layer (cf. Figure 5).

We denote the distance of column i and j in the grid by $|i-j|_{\mathrm{w}} := \min\{(i-j) \bmod W, (j-i) \bmod W\}$ and the k^{th}

triggering time of node (ℓ, i), i.e., the time when it forwards the k^{th} pulse, by $t_{\ell,i}^{(k)}$. In [17], quite "exotic" (but possible) worst-case propagation paths have been used to develop the worst-case skew bounds. These bounds are very robust against initial skews, which are captured by the *skew potential on layer* 0 denoted by $\Delta_0 := \max_{i,j\in[W]}\{t_{0,i} - t_{0,j} - |i-j|_{\mathrm{W}} d^-\}$. The latter is always non-negative, and is 0 when nodes in layer 0 that are separated by h hops trigger their pulse at times that are at most hd^- apart. Dropping the superscript (k) for readability, we restate the main result from [17] in Theorem 1. From here on we require that $\varepsilon = d^+ - d^- \leq d^+/7$ holds.

Theorem 1 (Skew Bounds—Fault-free Case [17, Thm. 3.8]): Suppose that $\varepsilon \leq d^+/7$. Then the following upper bounds hold on the intra-layer skew $\sigma_\ell := \max_{i\in[W]}\{|t_{\ell,i} - t_{\ell,i+1}|\}$ in layer ℓ: If $\Delta_0 = 0$, then σ_ℓ is uniformly bounded by $d^+ + \lceil W\varepsilon/d^+\rceil\varepsilon$ for any $\ell \in [L+1]$. In the general case,

$$\forall \ell \in \{1,\dots,2W-3\}: \ \sigma_\ell \leq d^+ + 2W\varepsilon^2/d^+ + \Delta_0. \quad (1)$$

$$\forall \ell \in \{2W-2,\dots,L\}: \ \sigma_\ell \leq d^+ + \lceil W\varepsilon/d^+\rceil\varepsilon. \quad (2)$$

Moreover, regarding the inter-layer skew of layer $\ell \in [L]$ to its neighboring layer(s), it holds for all $i \in [W]$ that

$$t_{\ell,i} - \sigma_\ell + d^- \leq t_{\ell+1,i} \leq t_{\ell,i} + \sigma_\ell + d^+ \text{ and} \quad (3)$$

$$t_{\ell,i+1} - \sigma_\ell + d^- \leq t_{\ell+1,i} \leq t_{\ell,i+1} + \sigma_\ell + d^+. \quad (4)$$

In the presence of failures, the above worst-case skew is only moderately increased (depending on the number of faulty nodes encountered along the worst-case propagation path). In addition, Theorem 2 shows that HEX self-stabilizes quickly from an arbitrary system state, e.g., caused by excessive transient failures.

Theorem 2 (Stabilization [17, Thm. 3.11]): Suppose $\max_{k\in\mathbb{N}}\{\Delta_0^{(k)}\} \leq \Delta$ and denote $\sigma_0 := \Delta + d^-$. Assume that

$$\min_{i\in[W]}\{t_{0,i}^{(k+1)}\} \geq \max_{i\in[W]}\{t_{0,i}^{(k)}\} + Wd^+ + L\varepsilon + T^+, \quad (5)$$

that $T^- > \sigma_\ell + d^+ + \varepsilon$ for all $\ell \in [L+1]$, where σ_ℓ is as in Theorem 1 with $\Delta_0 = \Delta$, and that the pulse generation algorithm employed at layer 0 is self-stabilizing. Then, HEX self-stabilizes within L pulses once layer 0 stabilized, in the sense that each node triggers exactly once per pulse, and for each pulse the bounds from Theorem 1 apply.

III. An Application: HEX-based Communication

In order to add a point-to-point communication subsystem to HEX, the canonical choice is to augment every (unidirectional) link in the HEX grid by additional signal wires for data communication, in both directions. The result is a (cylindrical) hexagonal grid with bidirectional communication between adjacent nodes. Every node in the grid now consists of two distinct parts, an extended HEX node and a communication node that uses the HEX clock.

In order to demonstrate the benefits of a synchronized clock for communication, and to reason about desired properties, we briefly describe a communication solution based on bi-synchronous *first in first out* (FIFO) buffers (shortly denoted FIFO) that has been adopted from [19]. More specifically, every communication node has a FIFO for each incoming communication link. Each FIFO has a write interface for writing data into and a read interface for reading data from the FIFO. The write interface is accessed and clocked by the sender, the read interface by the receiver.

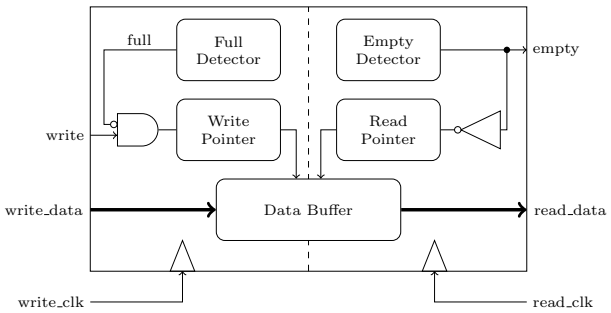


Fig. 3. Bi-Synchronous FIFO. Every Node has one FIFO for each incoming communication link. The sender is connected to the write interface, the receiver to the read interface.

The bi-synchronous FIFO is composed of five modules: *data buffer*, *read pointer*, *write pointer*, *full detector* and *empty detector* (see Figure 3). For the *data buffer*, we use a dual-ported *random access memory* (RAM). The sender- and the receiver-data lines are directly connected to the *data buffer*. The *read pointer*, as well as the *write pointer*, consist of a shift-register. The *read pointer* resp. *write pointer* determines the current read resp. write address. Furthermore, the two pointers are used by the *full detector* and the *empty detector*. *Write_data* is queued into the FIFO if *write* is high and *full* is low at the rising edge of *write_clk*. Data is dequeued to *read_data* if *empty* is low at the rising edge of *read_clk*. With every write access, the write address is incremented by one unless the FIFO is full. The read address is incremented by one with every read access, unless the FIFO is empty.

We have chosen this implementation because of its simple and metastability-aware design and its good fault-tolerance and containment properties. Notice that the obvious way, for a faulty sender to corrupt the corresponding receiver, is by writing to the address that is currently read and thus inducing metastability. Even if a faulty sender permanently sends messages, the full flag is set before the sender can write into the current read address. Thus the sender is prevented from corrupting the receiver. Another advantage arises from the fact that HEX provides bounded synchrony: A FIFO with a depth twice the synchronization precision plus one (plus a constant value to compensate the difference in end-to-end delays) is sufficient to ensure no message loss. Note carefully that this holds independently of the clock frequency (which determines the communication speed) and for any clock duty-cycle. In fact, the implementation also works with a quite irregular clock - as long as it remains synchronized. In Section VI, we will show how to build a fast clock atop of the synchronized HEX clocks, which allows high-speed communication.

IV. HEX Implementation and Testbed

Every HEX Node consists of a couple of very simple design entities. Its core is the asynchronous state machine shown in Figure 4, which implements the algorithm shown in Figure 2. It consists of three states only: *fire*, *sleep* and *ready*. The initial state is *ready*, where the state machine waits

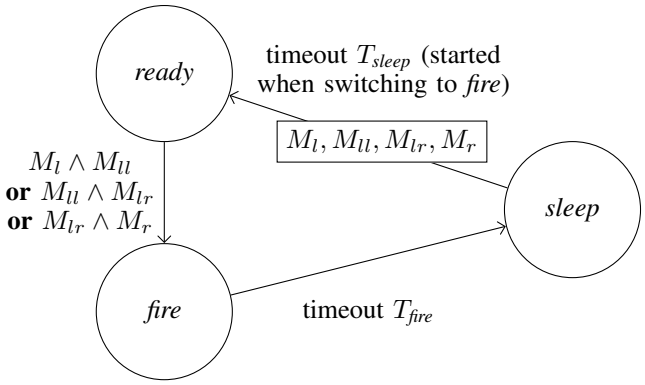


Fig. 4. Main state machine of a HEX node. Circular nodes represent states connected by transitions, labeled with their transition guards. A boxed node lists the memory flags to be cleared when taking the transition.

for the trigger condition in Figure 2 to become true. Resettable *memory flags* are used to memorize the occurrence of a pulse (indicated by active high (1)) from the neighbors. For example, the left neighbor is represented by the memory flag M_l. It is reset to low (0) when the state machine takes the transition from *sleep* to *ready*, as indicated by the boxed node. M_l is set to 1 when the input from the left neighbor becomes 1 after the most recent reset of M_l.

When the state *fire* is entered, both the sleep timer T_{sleep} and the minimal pulse duration timer T_{fire} are started and 1 is outputted on all outgoing links, to signal the pulse. After the timeout T_{fire} has occurred, the output is reverted to 0 and the state *sleep* is entered. The state *sleep* is a dormant state that is only left when the timeout of T_{sleep} (within $[T^-, T^+]$, as specified in Figure 2) occurs. Note that the reset of all memory flags occurring upon the transition to *ready* causes the node to forget all pulses sensed before.

The implementation of the above state machine is completely asynchronous and has been generated using the Petrify tool [20]. Both timers T_{sleep} and T_{fire} are driven by the same start/stoppable ring oscillator, which (like the memory flags) has already been used in our implementation of the FATAL$^+$ clock generation approach [16]. The resulting design of the HEX node was finally synthesized with Synopsis® Design Compiler version C-2009.06-SP4, using the UMC 90 nm standard cell library [21]. Note that we had to augment this library by a custom Muller C-Gate [22] developed in the context of the DARTS project [15][23]. Since an accurate timing characterization of this C-Gate is not available, its timing information was just copied from an AND-Gate of the standard library. The resulting inaccuracy is negligible w.r.t. our purposes, though.

The HEX grid itself was implemented by means of a custom testbench, which has the following purposes:

(1) Set the grid size and instantiate the corresponding number of nodes and interconnecting wires.
(2) Provide the layer 0 clock sources, i.e., generate the clock pulse of node $(0, i)$, $i \in [W]$, at time $t_{0,i}$, with some pre-selected skews. Clock sources for a single pulse and for multiple pulses are supported.
(3) Control the individual link delays during the simulation. Both random delays (uniform within lower and upper bound) and deterministic delays are supported.
(4) Control fault injection during the simulation: Both nodes and links can be declared *correct*, *Byzantine* (for each pulse and link, randomly choose output constant 0 or 1, which corresponds to no/fast pulse), or *fail-silent* (output constant 0). The selection of faulty nodes and/or links can be done deterministically or randomly (but static for multi-pulse simulations).

In order to support a reasonably systematic evaluation, we developed a software infrastructure in Haskell that allowed us to generate testbenches for different parameter settings (1)-(4). Every such testbench was then evaluated by means of pre-layout timing simulations, using Mentor Graphics® ModelSim 10.1d. The simulation results were recorded via event lists, which facilitated post-processing by our software infrastructure.

V. Simulation Results

The primary purpose of our simulation experiments is to complement the analytic worst-case bounds by statistics and average-case results, which are difficult to determine analytically. In view of the quite exotic worst-case scenarios obtained in [17], we (correctly) conjectured that the latter should be much better in reality. In the fault-free case, this has already been confirmed by means of high-level MATLAB-simulations of the algorithm in Figure 2 in [17]. The primary focus of the simulations presented in this paper, which employ the digital HEX implementation described in Section IV, lies (A) on scenarios including faulty nodes and (B) on stabilization time.

More specifically, using the testbed described in Section IV, we conducted the following simulation experiments:

(A) Statistical evaluation of the neighbor skews. These experiments require simulations involving a single pulse only. The primary quantities of interest here are:

- the (absolute) *layer ℓ intra-layer neighbor skews* $|t_{\ell,i} - t_{\ell,i-1}|$ of every node (ℓ, i) in layer ℓ,
- the (signed) *inter-layer neighbor skews* $t_{\ell,i} - t_{\ell-1,i}$ and $t_{\ell,i} - t_{\ell-1,i+1}$ of every node (ℓ, i) relative to its direct layer $\ell - 1$ neighbors $(\ell - 1, i)$ and $(\ell - 1, i + 1)$, respectively.

We remark that the former is defined in terms of the absolute values due to the symmetry of the topology (and thus skews) within a layer, whereas the latter respects the sign and thus correctly captures the non-zero bias (of at least d^-) in the inter-layer neighbor skew. Note that such a known bias can be compensated at the application layer if desired, see Section VI. Let $\sigma_\ell^{\mathrm{op}} := \mathrm{op}_{i\in[W]}\{|t_{\ell,i} - t_{\ell,i+1}|\}$ with $\mathrm{op} \in \{\mathrm{avg}, \max\}$ denote the average and maximum (absolute) layer ℓ intra-layer skew, respectively. Similarly, let $\hat{\sigma}_\ell^{\mathrm{op}} := \mathrm{op}_{i\in[W]}\{t_{\ell,i} - t_{\ell-1,i}, t_{\ell,i} - t_{\ell-1,i+1}\}$, where $\mathrm{op} \in \{\min, \mathrm{avg}, \max\}$, be the (signed) inter-layer skew between layer ℓ and $\ell - 1$. The global intra-layer resp. inter-layer skews in the entire system are defined as $\sigma^{\mathrm{op}} = \mathrm{op}_{\ell\in[L+1]}\{\sigma_\ell^{\mathrm{op}}\}$ resp. $\hat{\sigma}^{\mathrm{op}} = \mathrm{op}_{\ell\in[L+1]\setminus\{0\}}\{\hat{\sigma}_\ell^{\mathrm{op}}\}$.

(B) Statistical evaluation of the stabilization time. These experiments require multiple pulses. Essentially, the system is started, with every node in an arbitrary state, and then used to forward a sequence of correct pulses generated at layer 0.

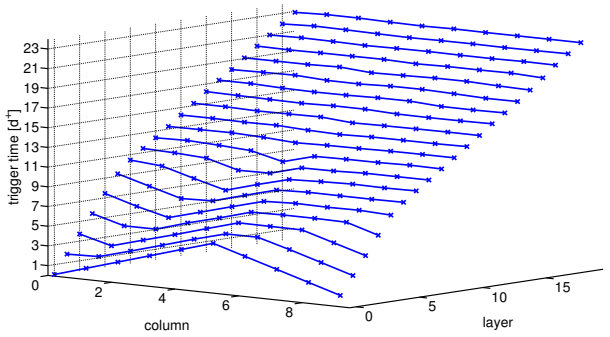


Fig. 5. Pulse wave propagation for uniformly chosen link delays in $[7, 8]$ and layer 0 neighbor skews ramping up/down by d^+.

TABLE I. INTRA- AND INTER-LAYER SKEW σ^{op} AND $\hat{\sigma}^{\text{op}}$ OF ALL NODES IN A 100×25 GRID FOR UNIFORM RANDOM END-TO-END DELAYS IN $[7.125, 8.165]$ NS. VALUES ARE DETERMINED OVER 300 TEST RUNS.

	intra-layer σ^{op}		inter-layer $\hat{\sigma}^{\text{op}}$		
init. layer 0	avg	max	min	avg	max
0	0.40	3.63	7.13	7.91	11.58
rand. $[0,d^-]$	0.46	6.97	7.13	7.94	15.07
rand. $[0,d^+]$	0.46	7.86	7.13	7.94	15.88
ramp d^+	1.41	8.16	0.96	8.36	16.28

Using post-processing of the recorded triggering times, we compute the stabilization time as the number of pulses needed for all intra- and inter layer skews to persistently go below their respective thresholds (determined according to $\sigma_\ell^{\max}$ and $\hat{\sigma}_\ell^{\max}$ determined in (A)). Unfortunately, lack of space does not allow us to adequately present these results in this paper.

Both types of experiments were performed with and without faulty nodes of different types. Note that the triggering times of faulty nodes are of course not considered when computing the inter- and intra-layer skews.

A. Neighbor skew evaluation

As an appetizer, Figure 5 shows a 3D plot of a typical pulse propagation wave in a fault-free grid with $W = 10$ and $L = 20$, with uniformly distributed link delays in $[7.125, 8.165]$ ns and layer 0 skews ramping up/down by d^+. The grid (sliced between width $W - 1$ and $0 \equiv W$) lies in the $(\ell \in [L + 1], i \in [W])$ plane, the z-axis gives the triggering time $t_{\ell,i}$ of the corresponding node (ℓ, i). To improve the readability of intra-layer skews, we connected all points $(\ell, i, t_{\ell,i})$ and $(\ell, i + 1, t_{\ell,i+1})$, $i \in \{0, \dots, W - 2\}$. It is apparent that the wave propagates evenly throughout the grid, nicely smoothing out differences in link delays and the large skews on layer 0.

To allow some qualitative comparison with the MATLAB-simulation results provided in [17], Table I shows the average (σ^{avg}) and maximal ($\sigma^{\max}$) intra-layer skew and the minimal ($\hat{\sigma}^{\min}$), average ($\hat{\sigma}^{\text{avg}}$), and maximal ($\hat{\sigma}^{\max}$) inter-layer skew, respectively, in the absence of faulty nodes. These values were computed over all nodes and 300 simulation runs, in the following setting (used throughout the remainder of this section): $L = 100$, $W = 25$ and link delays uniformly chosen within $[d^-, d^+]$ for $d^- = 7.125$ ns and $d^+ = 8.165$ ns ($\varepsilon = 1.04$ ns); these values result from combining assumed wire and routing delays within $[7, 8]$ ns with the switching delay bounds $[0.125, 0.165]$ ns determined during the HEX node synthesis. For the timeout T_{sleep}, a nominal value has been chosen that ensures a timeout within $[T^-, T^+]$ ns with $T^- = 20.393$ ns and $T^+ = 23$ ns, given the ring oscillator drift bounds determined during synthesis.

Table I shows the results of our experiments, using four different choices for the layer 0 skews between neighbors: The triggering times of the layer 0 nodes $t_{0,i}$ are (i) all 0 (resulting in $\sigma_0 = 0$ and skew potential $\Delta_0 = 0$), (ii) uniformly in $[0, d^-]$ (i.e., $\sigma_0 \approx d^-$ and $\Delta_0 = 0$), (iii) uniformly in $[0, d^+]$ (i.e., $\sigma_0 \approx d^+$ and $\Delta_0 \approx \varepsilon$), and (iv) ramping-up/down by d^+, i.e., $t_{0,i+1} = t_{0,i} + d^+$ for $0 \le i < W/2$ and $t_{0,i+1} = t_{0,i} - d^+$ for $W/2 \le i < W - 1$ (i.e., $\sigma_0 = d^+$ and $\Delta_0 \approx W\varepsilon/2 \approx 13$). Note that (iii) resp. (iv) reasonably model the average case and worst-case input provided by a layer 0 clock generation scheme with neighbor skew bound d^+, respectively. It is noteworthy that not a single instance in the collected data showed a skew of $\hat{\sigma}_\ell^{\max} > 2d^+$, and in scenarios (i) to (iii) we always had $\hat{\sigma}_\ell^{\min} < d^-$, i.e., all nodes were always triggered by their lower neighbors (obviously, this latter property is violated in scenario (iv) due to the excessive initial skews). The histogram of the skew distributions in case (i) are shown in Figure 6; the other cases look similar.

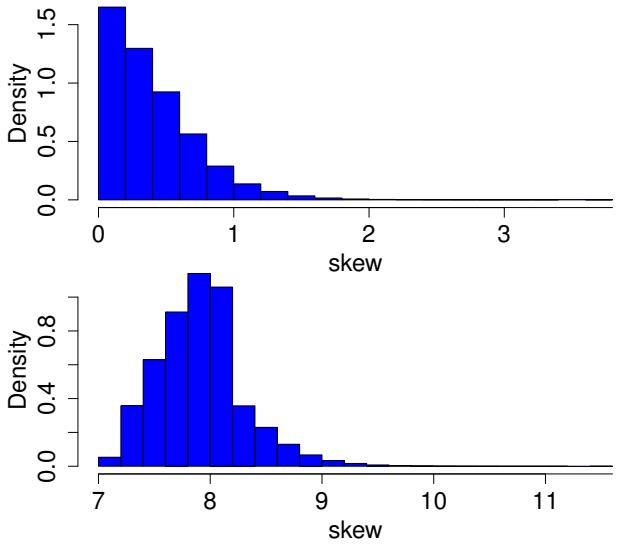


Fig. 6. Cumulated histograms for global intra-layer (top) and inter-layer (bottom) skew (in ns), from 300 simulation runs in scenario (i).

A comparison with the results of Theorem 1, which predicts $\sigma^{\max} \le 14$ ns and $[\hat{\sigma}^{\min}, \hat{\sigma}^{\max}] \subseteq [-5, 22]$ ns for scenarios (i) to (iii), reveals a much better behavior in the average case. Moreover, Figure 6 shows a sharp concentration with an exponential tail, i.e., *most* skews are very small in comparison to the worst-case bounds. For scenario (iv), the large initial skew (cf. Figure 5) leads to larger observed skews, which are also in accordance with our theorem, however.

Given the considerable differences between the minimum ($\hat{\sigma}^{\min}$) and maximum ($\hat{\sigma}^{\max}$) inter-layer skew in Table I, in conjunction with its non-zero bias, the question of layer-dependence arises. Table II provides $\hat{\sigma}_\ell^{\min}$, $\hat{\sigma}_\ell^{\text{avg}}$ and $\hat{\sigma}_\ell^{\max}$ for selected layers ℓ in case of scenario (iii). It reveals that there is no significant layer-dependent bias. Last but not least, the strong concentration of skews on each layer around the average

 ISBN: 978-1-61208-301-8

TABLE II. AVERAGE AND STANDARD DEVIATION OF $\hat{\sigma}_\ell^{\min}$, $\hat{\sigma}_\ell^{\text{avg}}$, AND $\hat{\sigma}_\ell^{\max}$, TAKEN OVER 250 SIMULATION RUNS OF SCENARIO (III).

layer ℓ	$\hat{\sigma}_\ell^{\min}$	$\hat{\sigma}_\ell^{\text{avg}}$	$\hat{\sigma}_\ell^{\max}$
1	7.19±0.04	9.01±0.22	14.22±0.78
2	7.20±0.05	8.41±0.15	12.88±1.03
3	7.21±0.05	8.18±0.12	11.73±1.20
5	7.21±0.06	8.02±0.08	10.33±0.97
9	7.23±0.07	7.94±0.06	9.36±0.52
10	7.22±0.06	7.94±0.06	9.29±0.47
11	7.22±0.06	7.92±0.06	9.23±0.45
12	7.22±0.06	7.93±0.06	9.17±0.45
15	7.23±0.07	7.92±0.06	9.11±0.35
19	7.21±0.06	7.92±0.06	9.09±0.36
20	7.23±0.06	7.92±0.06	9.11±0.36

(i.e., the very small standard deviation of $\hat{\sigma}_\ell^{\text{avg}}$), in particular in layers $\ell > 5$, shows that the smoothing out of local skews observed in Figure 5 is very typical.

Next, we consider the case where a certain number f of randomly chosen isolated nodes may be Byzantine faulty. All our simulations use the "average-case" scenario (iii) and the "worst-case" scenario (iv) described above. First, we choose f locations in the grid where faulty nodes are to be placed, subject to the constraint that no node has more than one faulty neighbor on an incoming edge. For a small number and a uniform distribution of faults, it is very likely that this constraint is satisfied. For an excessive number of faults, the birthday paradox kicks in: for $f = 5, 10, 20, 30$, the respective probabilities are roughly 0.95, 0.79, 0.36, and 0.09, respectively.

The behavior of the Byzantine faulty nodes is chosen randomly in every run on a per-link basis. Due to the usage of memory flags on the link port at the receiver side, a Byzantine faulty node has only two options: slowing down the firing of a node, or speeding it up. These two options are implemented as constantly sending pulses (i.e., constant high) and sending no pulse at all.

Then, 300 simulation runs of single pulse propagation are executed, each of which takes less than 2 minutes to simulate. We then later compute average and maximum of the intra-layer skew and minimum, average and maximum of the inter-layer skew. These computations where made several times on the generated data set, each time considering only pairs of correct nodes that are not reachable from any faulty node within at most $h = 0, 1, 2, 3$ directed hops in the grid. From these runs, we finally compute the average and standard deviation of σ^{avg}, $\sigma^{\max}$ and $\hat{\sigma}^{\min}$, $\hat{\sigma}^{\text{avg}}$, $\hat{\sigma}^{\max}$.

The results are shown in Table III. It is apparent that HEX copes very well with a considerable number of faults; minimal, average, and maximal skews are almost insensitive to the number of faults. For scenario (iv), we observe that standard deviations are generally larger, and that, in particular, nodes in directed distance 1 or 2 from faulty nodes experience larger skews. This is quite natural, given the large skews of the input; if the system suffers from a considerable number of faults, this limits its ability to reduce the initial skews.

Table IV shows the analogous results for fail-silent nodes; for brevity, we leave out $f = 2, 3$, for which we observed a behavior that interpolates between $f = 1$ and $f = 5$. The main difference to Byzantine faults are much more stable, but worse skews, the latter in particular demonstrated by $\hat{\sigma}^{\min}$. This can be easily understood, since the random behavior of Byzantine nodes increases the volatility of the setup, but decreases the number of "dead" links that inhibit the propagation of the pulse wave. With many faults, the wave needs to navigate a "maze" of dead nodes.

TABLE III. AVERAGE ± STANDARD DEVIATION OF σ^{avg}, $\sigma^{\max}$, $\hat{\sigma}^{\min}$, $\hat{\sigma}^{\text{avg}}$, AND $\hat{\sigma}^{\max}$, EXCLUDING ALL NODES WITH DIRECTED DISTANCE $\leq h$ FROM f ISOLATED BYZANTINE FAULTY NODES, OVER 300 SIMULATION RUNS OF SCENARIO (III) [TOP] AND (IV) [BOTTOM].

		Scenario (iii) Byzantine				
f	h	σ^{avg}	$\sigma^{\max}$	$\hat{\sigma}^{\min}$	$\hat{\sigma}^{\text{avg}}$	$\hat{\sigma}^{\max}$
1	0	0.54±0.05	7.76±0.92	6.95±0.36	7.98±0.03	15.56±1.02
1	1	0.54±0.05	7.35±0.58	7.12±0.06	7.98±0.03	15.13±0.63
1	2	0.53±0.05	7.21±0.58	7.13±0.03	7.97±0.02	14.99±0.62
1	3	0.53±0.05	7.11±0.57	7.13±0.02	7.97±0.02	14.89±0.62
2	0	0.60±0.07	8.15±0.87	6.75±0.79	8.00±0.03	16.09±1.11
2	1	0.59±0.07	7.56±0.48	7.12±0.07	8.00±0.03	15.39±0.52
2	2	0.58±0.06	7.36±0.51	7.12±0.06	8.00±0.03	15.17±0.56
2	3	0.57±0.06	7.23±0.52	7.13±0.04	7.99±0.03	15.01±0.59
3	0	0.65±0.07	8.48±0.87	6.33±1.36	8.03±0.04	16.47±1.27
3	1	0.64±0.07	7.73±0.35	7.10±0.12	8.03±0.03	15.56±0.32
3	2	0.63±0.07	7.53±0.39	7.12±0.07	8.02±0.03	15.35±0.36
3	3	0.62±0.07	7.38±0.44	7.12±0.07	8.02±0.03	15.21±0.41
5	0	0.76±0.10	8.91±1.01	5.54±1.99	8.09±0.05	17.13±1.72
5	1	0.74±0.09	7.86±0.29	7.05±0.21	8.08±0.05	15.70±0.28
5	2	0.72±0.09	7.70±0.30	7.10±0.09	8.07±0.05	15.53±0.30
5	3	0.70±0.09	7.59±0.33	7.11±0.07	8.06±0.04	15.40±0.33
10	0	0.95±0.12	9.55±1.18	3.62±2.59	8.18±0.06	18.85±2.52
10	1	0.91±0.11	8.01±0.38	6.89±0.54	8.16±0.06	15.84±0.17
10	2	0.87±0.11	7.84±0.17	7.05±0.15	8.14±0.05	15.63±0.19
10	3	0.84±0.11	7.75±0.22	7.07±0.14	8.13±0.05	15.53±0.22
20	0	1.27±0.13	10.61±1.46	0.65±2.54	8.34±0.07	21.29±2.17
20	1	1.18±0.13	8.45±1.15	6.20±1.51	8.31±0.07	15.92±0.15
20	2	1.11±0.13	7.96±0.12	6.96±0.23	8.28±0.06	15.75±0.18
20	3	1.05±0.13	7.88±0.16	7.01±0.17	8.25±0.06	15.64±0.19
		Scenario (iv) Byzantine				
f	h	σ^{avg}	$\sigma^{\max}$	$\hat{\sigma}^{\min}$	$\hat{\sigma}^{\text{avg}}$	$\hat{\sigma}^{\max}$
1	0	1.50±0.24	8.51±1.44	2.42±1.73	8.40±0.10	16.44±1.65
1	1	1.50±0.24	8.27±0.88	2.65±1.05	8.40±0.10	15.89±0.13
1	2	1.49±0.24	8.15±0.10	2.72±0.96	8.40±0.10	15.89±0.13
1	3	1.49±0.24	8.14±0.02	2.75±0.93	8.40±0.10	15.89±0.13
2	0	1.58±0.31	9.30±2.78	1.89±2.64	8.44±0.14	17.11±2.28
2	1	1.57±0.31	8.53±1.61	2.49±1.33	8.43±0.14	15.91±0.12
2	2	1.56±0.31	8.17±0.34	2.69±1.08	8.43±0.14	15.89±0.12
2	3	1.56±0.31	8.14±0.04	2.75±1.02	8.43±0.14	15.89±0.12
3	0	1.62±0.28	10.03±3.52	1.13±3.32	8.45±0.13	17.86±2.74
3	1	1.61±0.28	8.90±2.19	2.19±1.50	8.45±0.12	15.90±0.13
3	2	1.60±0.28	8.21±0.40	2.51±1.15	8.44±0.12	15.89±0.13
3	3	1.59±0.28	8.14±0.05	2.59±1.09	8.44±0.13	15.89±0.13
5	0	1.78±0.52	10.82±3.92	0.24±3.75	8.53±0.22	18.75±2.93
5	1	1.76±0.52	9.25±2.63	1.87±1.72	8.52±0.22	15.92±0.13
5	2	1.75±0.53	8.25±0.54	2.37±1.24	8.51±0.22	15.89±0.12
5	3	1.74±0.54	8.14±0.03	2.52±1.18	8.51±0.23	15.88±0.12
10	0	2.12±0.71	12.21±4.67	-1.36±4.18	8.67±0.29	21.09±2.61
10	1	2.09±0.72	10.06±3.31	1.09±2.11	8.66±0.29	15.93±0.13
10	2	2.06±0.73	8.39±0.90	1.89±1.55	8.65±0.30	15.89±0.13
10	3	2.04±0.76	8.14±0.11	2.12±1.53	8.64±0.31	15.88±0.15
20	0	2.59±0.98	16.20±7.91	-4.17±4.33	8.87±0.41	22.73±2.56
20	1	2.52±1.00	12.53±5.91	-0.14±2.25	8.85±0.41	16.16±1.93
20	2	2.48±1.04	9.20±4.99	1.20±1.66	8.83±0.43	16.01±1.74
20	3	2.44±1.07	8.68±4.84	1.63±1.71	8.82±0.46	15.92±1.20

VI. FAST CLOCKS

HEX provides each node with a local clock signal that is well-synchronized even in a system with multiple persistent

TABLE IV. AVERAGE ± STANDARD DEVIATION OF σ^{avg}, $\sigma^{\max}$, $\hat{\sigma}^{\min}$, $\hat{\sigma}^{\text{avg}}$, AND $\hat{\sigma}^{\max}$, EXCLUDING ALL NODES WITH DIRECTED DISTANCE $\leq h$ FROM f ISOLATED FAIL-SILENT NODES, OVER 300 SIMULATION RUNS OF SCENARIO (III) [TOP] AND (IV) [BOTTOM].

		Scenario (iii) fail silent				
f	h	σ^{avg}	$\sigma^{\max}$	$\hat{\sigma}^{\min}$	$\hat{\sigma}^{\text{avg}}$	$\hat{\sigma}^{\max}$
1	0	0.57±0.04	7.83±0.26	7.09±0.11	7.99±0.02	15.90±0.65
1	1	0.56±0.04	7.58±0.32	7.13±0.04	7.99±0.02	15.44±0.32
1	2	0.56±0.04	7.40±0.39	7.13±0.03	7.99±0.02	15.24±0.38
1	3	0.55±0.04	7.26±0.44	7.13±0.02	7.98±0.02	15.11±0.43
5	0	0.85±0.09	8.13±0.43	6.87±0.46	8.14±0.04	17.42±1.95
5	1	0.82±0.09	7.92±0.14	7.04±0.15	8.12±0.04	15.79±0.19
5	2	0.80±0.09	7.81±0.19	7.08±0.13	8.11±0.04	15.61±0.21
5	3	0.77±0.08	7.71±0.24	7.09±0.12	8.10±0.04	15.51±0.23
10	0	1.12±0.12	9.09±1.96	5.95±1.86	8.28±0.06	19.62±2.44
10	1	1.07±0.12	8.04±0.30	6.91±0.33	8.26±0.06	15.91±0.16
10	2	1.01±0.11	7.93±0.13	6.99±0.17	8.23±0.06	15.74±0.17
10	3	0.97±0.11	7.87±0.16	7.03±0.16	8.21±0.06	15.64±0.20
20	0	1.54±0.14	11.60±3.50	3.49±3.43	8.51±0.07	22.22±1.38
20	1	1.42±0.14	8.42±1.11	6.48±1.07	8.46±0.07	15.99±0.12
20	2	1.33±0.14	8.03±0.10	6.87±0.21	8.41±0.07	15.82±0.15
20	3	1.24±0.14	7.97±0.12	6.93±0.20	8.37±0.07	15.73±0.15
		Scenario (iv) fail-silent				
f	h	σ^{avg}	$\sigma^{\max}$	$\hat{\sigma}^{\min}$	$\hat{\sigma}^{\text{avg}}$	$\hat{\sigma}^{\max}$
1	0	1.50±0.05	9.49±3.96	1.95±2.91	8.40±0.02	16.71±1.98
1	1	1.50±0.05	8.69±1.87	2.62±1.18	8.40±0.02	15.90±0.13
1	2	1.49±0.05	8.17±0.26	2.86±0.83	8.40±0.03	15.89±0.13
1	3	1.49±0.05	8.14±0.02	2.87±0.83	8.40±0.03	15.89±0.13
5	0	1.77±0.09	14.46±6.75	-1.25±5.08	8.54±0.05	19.98±2.95
5	1	1.74±0.09	10.74±3.56	1.66±2.14	8.53±0.05	15.94±0.12
5	2	1.71±0.10	8.42±0.92	2.88±0.85	8.51±0.05	15.90±0.12
5	3	1.69±0.10	8.14±0.02	2.92±0.85	8.50±0.05	15.89±0.12
10	0	2.02±0.13	18.38±6.47	-4.03±5.21	8.67±0.06	22.04±2.09
10	1	1.96±0.13	12.68±3.92	0.59±2.61	8.64±0.07	15.97±0.12
10	2	1.91±0.14	8.74±1.33	2.92±1.00	8.62±0.07	15.91±0.11
10	3	1.87±0.15	8.15±0.22	3.03±0.94	8.60±0.08	15.89±0.11
20	0	2.40±0.16	22.58±4.20	-7.39±3.60	8.87±0.08	23.19±1.16
20	1	2.29±0.16	15.26±3.03	-1.08±2.66	8.83±0.08	16.03±0.11
20	2	2.20±0.17	9.49±1.85	2.81±0.97	8.78±0.09	15.92±0.11
20	3	2.12±0.19	8.15±0.15	3.11±0.91	8.74±0.10	15.89±0.12

faults. Furthermore, HEX is able to recover from an unbounded number of concurrent transient faults. It should not come as a surprise that these impressive fault-tolerance properties come at a cost: The HEX clock signal has a fairly low and unstable frequency and a large jitter. This makes HEX unsuitable for applications that need to determine real-time durations very accurately. One such negative example is clock multiplication based on PLLs, which sustain only moderate input jitter. Fortunately, as exemplified by our communication subsystem in Section III, many applications do not have such stringent requirements.

A. Trade-offs between Quality and Dependability

To understand that the undesirable properties of HEX are inherent to our approach, if not even inevitable under the given design goals, recall that Byzantine or fail-silent nodes locally affect the triggering times by (i) cutting off or delaying the clock distribution signal on a (shortest) path to some node or (ii) triggering a pulse early. Because of (ii), a node cannot locally trigger a pulse just based on one of its lower neighbors; because of (i), a faulty lower-left or lower-right neighbor entails that the node must be triggered with the help of a node in the same layer, thereby increasing the length of a causal chain [24] involved in triggering the node compared to the fault-free case. Consequently, the time when the node is triggered may be affected notably, and this effect may accumulate over several layers (in contrast to the skew, which is the *local* difference of triggering times).

From these observations, we can conclude that simultaneously guaranteeing tolerance of Byzantine faults *and* a stable clock frequency would entail a stronger connectivity of the grid and thus larger node degrees. In particular, nodes would need to receive clock signals from at least 3 nodes from the previous layer, as well as forwarding them to at least 3 nodes in the subsequent layer. Higher degrees, however, increase the complexity of nodes—and thus the likelihood that an individual node fails—as well as the probability that two neighbors are faulty (even for a fixed probability of failure). We hence conclude that there is a trade-off between frequency stability and resilience to failures.

Besides the inevitable switching delays of the components making up the HEX nodes, the low frequency of the generated HEX clock is also caused by our self-stabilization requirement: Ensuring that the nodes become ready for the next, well-synchronized pulse, a conservative pulse separation time must be granted to "flush out" spurious pulses from the system; otherwise, we might observe a phenomenon similar to "ventricular fibrillation".

B. Local Clocks

While increasing the frequency stability of the HEX clock signal would require a more dense topology, there is an obvious solution to the low clock frequency issue: frequency multiplication. By equipping each node with a high-frequency oscillator that is synchronized to the HEX clock, one can generate well-synchronized high-frequency clocks (termed *fast clocks* in the sequel).

However, synchronizing a fast clock to an unstable HEX clock involves a trade-off:

(a) If a stable fast clock frequency is desired, the HEX clock's jitter must be amortized over $m > 1$ pulses. For example, this could be implemented by first dividing the HEX clock by m and then using a PLL clock multiplier to generate the desired fast clock; m must be chosen appropriately to guarantee an acceptable PLL input clock jitter. Unfortunately, this approach may increase the skew of the fast clock considerably if the high-frequency oscillators driving the fast clocks have a large drift.

(b) If a fast clock with minimal skew is desired, which is our major objective, the HEX clock jitter must be amortized within a single pulse. Unfortunately, using solutions like [25] are complex and inherently lead to considerable frequency fluctuations of the fast clock.

With this in mind, we propose a simple, robust approach for (b) that achieves a high and reasonably stable fast clock frequency with good skews, at the expense of *burstiness*. For each HEX pulse, the fast clock generates a fixed number B of fast clock pulses [26] as shown in Figure 7. This is accomplished by means of a free-running start/stoppable ring oscillator, which is started by a HEX pulse and stops when it has generated B pulses; in fact, we may make use of the same oscillator design already used for the timeouts T_{fire} and T_{sleep} of the HEX state machine (Figure 4). Note that this can be implemented in a way that entirely avoids metastability.

 ISBN: 978-1-61208-301-8

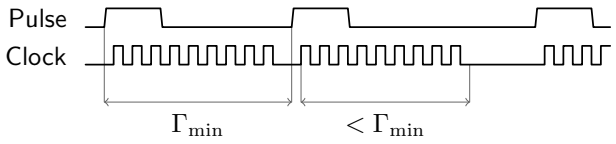


Fig. 7. A pulse generated by HEX will result in a fixed number of clock cycles being generated. The number of clock cycles and their frequency has to be chosen so that the required time span for them is less than the minimal time between pulses at correct nodes.

C. Analysis

First, we determine conservative timing constraints under the assumption that the system is fault-free. In this setting, the feasible number of clock cycles in each burst can be precisely expressed in terms of the *minimal pulse separation time*

$$\Gamma_{\min} := \inf_{\substack{i\in[W],\ell\in[L+1],k\in\mathbb{N}\\(\ell,i)\text{ correct}}} \{t_{\ell,i}^{(k)} - t_{\ell,i}^{(k-1)}\}. \tag{6}$$

Assuming that the fast clock sources are guaranteed to run at least with frequency $f_{\min}$, the number B of generated ticks per pulse must satisfy

$$B \leq f_{\min}\Gamma_{\min}. \tag{7}$$

Note that it is sufficient here if the frequency bound holds amortized over $\Gamma_{\min}$ time. To understand the resulting performance, assume for simplicity that this bound is an integer (i.e., we neglect that we may lose a "fractional tick") and choose $B = f_{\min}\Gamma_{\min}$. First, let us check the amortized frequency of the system. Clearly, we cannot guarantee a larger amortized frequency than $f_{\min}$. Of course, in addition $\Gamma_{\min}$ could be smaller than the long-term average time between pulses

$$\Gamma_{\text{avg}} := \lim_{k\to\infty} \frac{t_{0,0}^{(k)} - t_{0,0}^{(0)}}{k}. \tag{8}$$

Note that choosing the reference node to be $(0,0)$ in this definition is arbitrary, as using every other node must lead to the same result: Since the skew in the fault-free case is bounded, the influence of the difference of the triggering times vanishes in the limit (of course we assume here that the clock generation algorithm employed on layer 0 has bounded skew in the absence of faults). With this definition, the amortized frequency of the generated high-frequency clock of node (ℓ,i) can be expressed as

$$\lim_{k\to\infty} \frac{Bk}{t_{\ell,i}^{(k)} - t_{\ell,i}^{(0)}} = B \lim_{k\to\infty} \frac{k}{t_{0,0}^{(k)} - t_{0,0}^{(0)}} = f_{\min}\frac{\Gamma_{\min}}{\Gamma_{\text{avg}}}, \tag{9}$$

which equals $f_{\min}$ exactly if $t_{\ell,i}^{(k+2)} - t_{\ell,i}^{(k+1)} = t_{\ell,i}^{(k+1)} - t_{\ell,i}^{(k)}$ for all i, ℓ, and k, i.e., the HEX clock is perfectly stable. The "frequency loss" is determined by three factors:

1) The frequency fluctuation of the clock generation algorithm at layer 0.
2) The skew between the layer 0 nodes.
3) The variance in the speed at which pulses propagate through the grid.

The first two factors are determined by the clock generation algorithm and thus not to attribute to HEX. We remark that since we need to keep pulses well-separated, the first factor is likely to dominate the second. Similarly, large pulse separation times mitigate the influence of the third factor. If we increase the pulse separation time and take the limit, the resulting frequency of HEX will reflect the frequency provided by the clock generation algorithm at layer 0 (since the influence of skews vanishes in the limit). Hence, the frequency of the fast clocks will be $f_{\min}$ multiplied by the ratio between the frequency lower bound of the pulse generation algorithm—neglecting any additive variations that do not depend on the pulse separation time, in particular the skew between layer 0 nodes—and its average frequency.

These are good news, showing that a large pulse separation time does not hurt in terms of the overall frequency at which the system will be clocked. On the contrary, large pulse separation times essentially ensure the maximum frequency we can hope for! Even with fairly small pulse separation times, the system will run at a constant fraction of the frequency $f_{\min}$. However, there is no free lunch, as we will establish now by analyzing the fast clock skew of adjacent nodes.

Since pulses are anonymous, we interpret the generated local fast clock $L_{\ell,i}$ of node (ℓ,i) as a clock modulo B with the initial value being 0. For simplicity, we assume that the clock is continuous, i.e., it is real-valued from $[0,B)$, increasing modulo B at (possibly varying) rates from $[f_{\min}, f_{\max}]$, where $f_{\max}$ is its maximal frequency. The actual discrete clock reading at time t then is simply $\lfloor L_{\ell,i}(t)\rfloor$. With these definitions, the clock value at time $t \in [t_{\ell,i}^{(k)}, t_{\ell,i}^{(k+1)}]$ satisfies

$$\min\{f_{\min}(t - t_{\ell,i}^{(k)}), B\} \leq L_{\ell,i}(t) \leq \min\{f_{\max}(t - t_{\ell,i}^{(k)}), B\}, \tag{10}$$

since the clock is halted until the next pulse once it reaches value $B \equiv 0$.

Observe that at times where two adjacent nodes both locally triggered pulse k, but not pulse $k+1$, the worst possible clock skew is attained if (i) the difference in the triggering times is maximal, (ii) the clock of the node that triggered first runs at frequency $f_{\max}$ until its clock halts, and (iii) the other node's clock runs at frequency $f_{\min}$. For the skew between two adjacent nodes in layer ℓ, we thus get at times $t \in [t_{\ell,i}^{(k)}, t_{\ell,i}^{(k+1)}] \cap [t_{\ell,i+1}^{(k)}, t_{\ell,i+1}^{(k+1)}]$ that

$$\begin{aligned} &|L_{\ell,i}(t) - L_{\ell,i+1}(t)| \\ &\leq\ B - f_{\min}\left(\frac{B}{f_{\max}} - |t_{\ell,i}^{(k)} - t_{\ell,i+1}^{(k)}|\right) \\ &\leq\ \frac{f_{\max} - f_{\min}}{f_{\max}} \cdot B + f_{\min}\sigma_{\ell}^{\max} = \varrho B + f_{\min}\sigma_{\ell}^{\max}, \end{aligned} \tag{11}$$

where $\varrho := (f_{\max} - f_{\min})/f_{\max}$ is the *relative drift* of the high-speed clocks.

Now consider a time $t \in [t_{\ell,i}^{(k+1)}, t_{\ell,i}^{(k+2)}] \cap [t_{\ell,i+1}^{(k)}, t_{\ell,i+1}^{(k+1)}]$, i.e., node (ℓ,i) already triggered pulse $k+1$, but node $(\ell,i+1)$ has not done so yet. Since node i starts its clock—which in the worst case runs fast while the clock of node $i+1$ runs slow—at time $t_{\ell,i}^{(k+1)}$ again, a worst-case bound on the clock skew (modulo B) is obtained by comparing the clocks just before time $t_{\ell,i+1}^{(k+1)}$. However, this bound is subsumed by the previous one, as it covers the case $t = t_{\ell,i+1}^{(k+1)}$. Finally, the case $t \in [t_{\ell,i}^{(k)}, t_{\ell,i}^{(k+1)}] \cap [t_{\ell,i+1}^{(k+1)}, t_{\ell,i+1}^{(k+2)}]$ is symmetrical. This covers

 ISBN: 978-1-61208-301-8

all cases and therefore provides a worst-case bound for the intra-layer skew of the constructed fast clocks; replacing $\sigma_\ell^{\max}$ by σ_ℓ^{avg} in this result yields an average-case bound.

To also derive good bounds for the inter-layer skew, more care is necessary. Of course, we could simply replace $\sigma_\ell^{\max}$ by $\hat{\sigma}_\ell^{\max}$ and repeat the same analysis, but this would provide overly conservative results as it fails to leverage the known bias of the inter-layer skew. We will hence use a refined analysis for slightly modified clocks to improve the results.

On average, nodes at layer ℓ trigger roughly $\hat{\sigma}_\ell^{\text{avg}}$ after those at layer $\ell-1$ (recall that $\hat{\sigma}_\ell^{\text{avg}}$ respects the sign). Therefore, we can compensate the resulting bias in the inter-layer skew of the fast clocks by *shifting* the fast clocks of the nodes at layer $\ell > 1$ accordingly with respect to layer $\ell-1$. Similarly, to obtain a worst-case bound, we can consider $\hat{\sigma}_\ell^{\min}$ and $\hat{\sigma}_\ell^{\max}$ and apply a shift corresponding to $\bar{\sigma}_\ell := (\hat{\sigma}_\ell^{\max} + \hat{\sigma}_\ell^{\min})/2$.

Let us examine the latter case here; the former can be treated similarly. In $\bar{\sigma}_\ell$ time, a running fast clock makes progress of at least $f_{\min}\bar{\sigma}_\ell$. Therefore, we map the clock values $L_{\ell,i}(t)$ for all i and t to $L'_{\ell,i}(t) := (L_{\ell,i}(t) + f_{\min}\bar{\sigma}_\ell) \bmod B$. Analogous to the intra-layer skew, we now can bound

$$\begin{aligned} & L_{\ell-1,i}(t) - L'_{\ell,i}(t) \\ \leq\ & B - f_{\min}\left(\frac{B}{f_{\max}} - (t_{\ell,i}^{(k)} - t_{\ell-1,i}^{(k)}) + \bar{\sigma}_\ell\right) \\ \leq\ & \frac{f_{\max}-f_{\min}}{f_{\max}} \cdot B + f_{\min}\left(\hat{\sigma}_\ell^{\max} - \bar{\sigma}_\ell\right) \\ =\ & \varrho B + f_{\min} \cdot \frac{\hat{\sigma}_\ell^{\max} - \hat{\sigma}_\ell^{\min}}{2} = \varrho B + f_{\min}\tau_\ell, \quad (12) \end{aligned}$$

where $\tau_\ell := (\hat{\sigma}_\ell^{\max} - \hat{\sigma}_\ell^{\min})/2$ corresponds to $\sigma_\ell^{\max}$. The latter can be seen by noting that the the signed variant of the (symmetrical, i.e., absolute value) inter-layer skew is just $-\sigma_\ell$. Likewise,

$$\begin{aligned} & L'_{\ell,i}(t) - L_{\ell-1,i}(t) \\ \leq\ & B - f_{\min}\left(\frac{B}{f_{\max}} + (t_{\ell,i}^{(k)} - t_{\ell-1,i}^{(k)}) - \bar{\sigma}_\ell\right) \\ \leq\ & \frac{f_{\max}-f_{\min}}{f_{\max}} B + f_{\min}\left(\bar{\sigma}_\ell - \hat{\sigma}_\ell^{\min}\right) = \varrho B + f_{\min}\tau_\ell, \quad (13) \end{aligned}$$

which together with the bound on $L_{\ell-1,i}(t) - L'_{\ell,i}(t)$ implies that $|L'_{\ell,i}(t) - L_{\ell-1,i}(t)| \leq \varrho B + f_{\min}\tau_\ell$. An analogous reasoning shows that also $|L'_{\ell,i}(t) - L_{\ell-1,i+1}(t)| \leq \varrho B + f_{\min}\tau_\ell$.

Obviously, shifting all clocks in a layer by the same value will not affect the intra-layer clock skews. Applying the clock shifts inductively, i.e., shifting the clocks in each layer $\ell \geq 1$ by $\sum_{\ell'=0}^{\ell-1} \bar{\sigma}_{\ell'}$, we can bound the inter-layer skews on each pair of consecutive layers as above. We thus can conclude with the worst-case bounds

$$\varrho B + f_{\min}\sigma_\ell^{\max} \leq f_{\min}(\varrho\Gamma_{\min} + \sigma_\ell^{\max}) \text{ and} \quad (14)$$

$$\varrho B + f_{\min}\tau_\ell \leq f_{\min}(\varrho\Gamma_{\min} + \tau_\ell). \quad (15)$$

on the intra- and inter-layer skews of the (shifted) fast clocks. Note that they are tight for $B = f_{\min}\Gamma_{\min}$, which maximizes the amortized frequency of the fast clocks.

It can be seen that even if ϱ is fairly large (say, 10%) and $\Gamma_{\min}$ is significantly larger than $\sigma_\ell^{\max}$, say, up to factor 10, the maximal clock skew is still dominated by the term $f_{\min}\sigma_\ell^{\max}$. If one is willing to invest in more stable fast clock sources (e.g., $\varrho \approx 1\%$), large pulse separation times are feasible without incurring a significant impact on the skew. We stress that $f_{\min}$ and $f_{\max}$ in our bounds are—unless the clocks are extremely unstable—the amortized frequency upper and lower bounds over B fast clock pulses; large skews require a consistent difference in frequency for roughly $B/f_{\min}$ time. Moreover, ϱ captures the *relative drift* of the clocks. Any frequency change that applies to adjacent nodes in roughly the same way (i.e., a system-wide change in temperature or supply voltage) will not have noticeable effects on the skews.

D. Faults, Self-stabilization, and $\Gamma_{\min}$

So far, we neglected two key obstacles in our approach for constructing fast clocks:

- $\Gamma_{\min}$ is not known, and therefore we do not know an appropriate choice for B.
- Due to persistent or transient faults, $\Gamma_{\min}$ may be very small or not even well-defined.

Concerning the first issue, there are basically three options: We can rely on analytical bounds, simulation results, or experimental data. All approaches have pros and cons; for a final system, experimental data is the most valuable, but it is also the most difficult and expensive to create, as one needs a chip and a suitable apparatus for measurements first. From [17] and bounds for the layer 0 clock generation algorithm employed, analytical worst-case bounds can be derived. The results from Section V provide insight into the skew distributions for an average-case setting under some fairly conservative assumptions on the parameters.

Regarding the second issue, we clearly must exclude faulty nodes as well as triggering time differences from the self-stabilization period when estimating (an equivalent to) $\Gamma_{\min}$ in the general setting. Note that there is a trade-off: We can consider a non-faulty node that experiences large skews as incorrect, permitting the use of a less conservative bound on $\Gamma_{\min}$; in turn, we are losing the guarantee that this node will be able to complete each clock burst before the next pulse arrives, even after stabilization of the HEX pulse generation. The results from Section V give an idea on the behavior of the system under faults. It should be noted, though, that the specific values are highly dependent on the implementation of the HEX nodes, the layer 0 clock generation algorithm, and the used technology.

Once a suitable estimate $\tilde{\Gamma}_{\min}$ taking the role of $\Gamma_{\min}$ has been determined, one can choose B according to $B \leq f_{\min}\tilde{\Gamma}_{\min}$. Note that using a smaller value for B may be desirable in some situations, as a smaller value of B also leads to a smaller skew. On the downside, $B < f_{\min}\tilde{\Gamma}_{\min}$ is equivalent to using a smaller value of $\tilde{\Gamma}_{\min}$, which leads to a lower amortized frequency of the fast clocks. The resulting decrease in the amortized frequency can of course be mitigated by increasing the frequency of the layer 0 pulse generation algorithm, which reduces $\tilde{\Gamma}_{\min}$, Γ_{avg} and the gap $f_{\min}\tilde{\Gamma}_{\min} - B$.

VII. Conclusion and Future Work

In the work presented in this paper, we (i) implemented the HEX clock distribution grid [17] using the UMC 90 nm

standard cell library, (ii) performed extensive ModelSim-based simulations to gain insight on the average-case performance of the approach, and (iii) constructed small-skew high-frequency fast clocks and analyzed their performance. As argued in Section III, such clocks are a suitable basis for an efficient point-to-point communication subsystem. While it is still a long road to a fully-engineered solution, our results demonstrate the potential of the envisioned clocking scheme to serve in highly dependable, multi-synchronous GALS architectures.

Future Work. There are several vital improvements to the presented solutions that are subject to our ongoing research.

- The communication system proposed in Section III is not self-stabilizing, but merely *pseudo-stabilizing* [27]. That is, once transient faults cease, the system is guaranteed to eventually operate correctly, but there is no bound on the time the recovery process takes to actually complete. Based on the self-stabilizing fast clocks, it seems reasonable to assume that also fully self-stabilizing communication primitives can be devised.
- The skews offered by HEX compare unfavorably to those of less resilient solutions; in particular the gap to the performance of standard clock distribution trees (which cannot withstand any faults) is large. Because of the simplicity of the HEX state machine (Figure 4), we are positive that a HEX node can be implemented as a transistor-level custom standard cell. This would certainly decrease the resulting skews and pulse separation times further, and thus result in an improved HEX performance.
- The cylindrical hexagonal grid topology assumed in this work and [17] is convenient for analysis, but suffers from two drawbacks: (i) providing a synchronized clock to the entire layer 0 is difficult unless the grid is very narrow (i.e., W is small) and (ii) "folding" a cylindrical HEX grid onto an actual chip would result in two clock layers with physically close-by nodes that are far from each other in the grid. In [17], we proposed an alternative topology resolving these issues, and the analysis in the paper provides strong intuition that the resulting skews will not be larger. Generalizing the analysis from [17] to the new topology and performing corresponding simulations is thus of high interest.

REFERENCES

[1] D. M. Chapiro, "Globally-Asynchronous Locally-Synchronous Systems," Ph.D. dissertation, Stanford University, 1984.

[2] Y. Semiat and R. Ginosar, "Timing Measurements of Synchronization Circuits," in Proc. 9th Symposium on Asynchronous Circuits and Systems (ASYNC), 2003.

[3] P. Teehan, M. Greenstreet, and G. Lemieux, "A Survey and Taxonomy of GALS Design Styles," IEEE Design and Test of Computers, vol. 24, no. 5, 2007, pp. 418–428.

[4] D. G. Messerschmitt, "Synchronization in Digital System Design," IEEE Journal on Selected Areas in Communications, vol. 8, no. 8, 1990, pp. 1404–1419.

[5] A. Hansson, M. Subburaman, and K. G. W. Goossens, "Aelite: A Flit-Synchronous Network on Chip with Composable and Predictable Services," in Proceedings Design, Automation & Test in Europe Conference and Exhibition (DATE 2009), Nice, France. Los Alamitos: IEEE Computer Society, April 2009, pp. 250–255.

[6] T. Polzer, T. Handl, and A. Steininger, "A Metastability-Free Multi-Synchronous Communication Scheme for SoCs," in Proc. 11th Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS), 2009, pp. 578–592.

[7] W. S. Coates and R. J. Drost, "Congestion and Starvation Detection in Ripple FIFOs," in Proceedings of the 9th International Symposium on Asynchronous Circuits and Systems, ser. ASYNC '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 36–45.

[8] M. S. Maza and M. L. Aranda, "Interconnected Rings and Oscillators as Gigahertz Clock Distribution Nets," in Proc. 13th Great Lakes Symposium on VLSI (GLSVLSI), 2003, pp. 41–44.

[9] S. Fairbanks, "Method and Apparatus for a Distributed Clock Generator," 2004, uS patent no. US2004108876 [retrieved: 06, 2013]. [Online]. Available: http://v3.espacenet.com/textdoc?DB=EPODOC&IDX=US2004108876

[10] S. Fairbanks and S. Moore, "Self-Timed Circuitry for Global Clocking," in Proc. 11th Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC), 2005, pp. 86–96.

[11] V. Gutnik and A. Chandrakasan, "Active GHz Clock Network Using Distributed PLLs," IEEE Journal of Solid-State Circuits, vol. 35, no. 11, 2000, pp. 1553–1560.

[12] A. Korniienko et al., "A Clock Network of Distributed ADPLLs Using an Asymmetric Comparison Strategy," in Proc. 2010 Symposium on Circuits and Systems (ISCAS), 2010, pp. 3212–3215.

[13] M. Saint-Laurent and M. Swaminathan, "A Multi-PLL Clock Distribution Architecture for Gigascale Integration," in Proc. 2001 IEEE Computer Society Workshop on VLSI (WVLSI), 2001, pp. 30–35.

[14] M. Függer, A. Dielacher, and U. Schmid, "How to Speed-Up Fault-Tolerant Clock Generation in VLSI Systems-on-Chip via Pipelining," in Proc. 8th European Dependable Computing Conference (EDCC), 2010, pp. 230–239.

[15] M. Függer and U. Schmid, "Reconciling Fault-Tolerant Distributed Computing and Systems-on-Chip," Distributed Computing, vol. 24, no. 6, 2012, pp. 323–355.

[16] D. Dolev, M. Függer, C. Lenzen, and U. Schmid, "Fault-Tolerant Algorithms for Tick-Generation in Asynchronous Logic: Robust Pulse Generation - [Extended Abstract]," in Proc. 13th Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS), 2011, pp. 163–177.

[17] D. Dolev, M. Függer, C. Lenzen, M. Perner, and U. Schmid, "HEX: Scaling Honeycombs is Easier than Scaling Clock Trees," in Proc. 25th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), 2013, in press.

[18] E. W. Dijkstra, "Self-Stabilizing Systems in Spite of Distributed Control," Communications of the ACM, vol. 17, no. 11, 1974, pp. 643–644.

[19] I. Miro Panades and A. Greiner, "Bi-Synchronous FIFO for Synchronous Circuit Communication Well Suited for Network-on-Chip in GALS Architectures," in Networks-on-Chip, 2007. NOCS 2007. First International Symposium on, May 2007, pp. 83–94.

[20] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, Logic Synthesis for Asynchronous Controllers and Interfaces. Springer, 2002.

[21] UMC 90 nm, [retrieved: 06, 2013]. [Online]. Available: http://www.umc.com/English/process/g.asp

[22] I. E. Sutherland, "Micropipelines," Communications of the ACM, Turing Award, vol. 32, no. 6, 1989, pp. 720–738.

[23] G. Fuchs and A. Steininger, "VLSI Implementation of a Distributed Algorithm for Fault-Tolerant Clock Generation," Journal of Electrical and Computer Engineering, vol. 2011, no. 936712, 2011.

[24] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," Commun. ACM, vol. 21, no. 7, 1978, pp. 558–565.

[25] T. Olsson and P. Nilsson, "Portable Digital Clock Generator for Digital Signal Processing Applications," Electronics Letters, vol. 39, no. 19, sept. 2003, pp. 1372 – 1374.

[26] T. Olsson, P. Nilsson, T. Meincke, A. Hemam, and M. Torkelson, "A Digitally Controlled Low-power Clock Multiplier for Globally Asynchronous Locally Synchronous Designs," in Proc. 2000 Symposium on Circuits and Systems (ISCAS), vol. 3, 2000, pp. 13–16.

[27] J. E. Burns, M. G. Gouda, and R. E. Miller, "Stabilization and Pseudo-Stabilization," Distributed Computing, vol. 7, no. 1, 1993, pp. 35–42.

 ISBN: 978-1-61208-301-8

Reliable and Secure Memories Based on Algebraic Manipulation Detection Codes and Robust Error Correction

Shizun Ge
ECE, Boston University
Boston, MA, USA
Email: shizunge@gmail.com

Zhen Wang
Mediatek Wireless, Inc
Boston, MA, USA
Email: wang.zhen.mtk@gmail.com

Mark Karpovsky
ECE, Boston University
Boston, MA, USA
Email: markkar@bu.edu

Pei Luo
ECE, Boston University
Boston, MA, USA
Email: luopei@bu.edu

***Abstract*—The reliability and security of memories are crucial considerations in the modern digital system design. Traditional codes concentrate on detecting and correcting errors of certain types, e.g., errors with small multiplicities or byte errors, and cannot detect or correct unanticipated errors. Thereby, they may not be sufficient to protect memories against malicious attackers with strong fault injection capabilities and cannot correct unexpected errors with high multiplicities. The contribution of this paper is that we construct a new reliable and secure memory architecture based on robust Algebraic Manipulation Correction (AMC) codes. These codes can be used for correction of random errors and for detection of fault injection attacks. These codes can provide a guaranteed error detection probability for all errors even if both the user defined messages (data stored in the memory) and the error patterns are controllable by an attacker. The presented code can correct all single-bit errors in the information bits of the code. Moreover, these codes can be used to correct double-bit errors with high probabilities. The construction and the error correction procedures for the code will be described. The probability that an error can be successfully detected and/or corrected and the hardware overheads for the proposed memory architecture will be estimated. The presented approach is efficient for protecting security/reliability critical memories used to store the important information on the chip (e.g., a secret key in a cryptographic device).**

Keywords-Algebraic Manipulation Detection Code, Error Correction, Fault Injection Attack, Hardware Security

I. INTRODUCTION

Memories are critical elements in today's digital systems. Various types of memories are widely used in many different reliable and secure applications and appear in nearly all digital devices. SRAMs, for example, are often used as caches and internal memories in embedded systems. Non-volatile memories like EEPROM and Flashes are often used in cryptographic devices to store secret informations such as the encryption keys and passwords.

The reliability of memory is a crucial consideration for today's digital devices. For some designs as much as 70% of the chip area is taken by the embedded memory [1], [2]. This large area of the chip is especially vulnerable to single-event-upsets (SEUs) caused by single, energetic particles like high-energy neutrons and alpha particles. SEU temporarily alters the state of the devices and results in soft errors. These errors are nondestructive and appear as unwanted bit flips in memory cells and registers. Continuing scaling of device features and performance increases the likelihood of errors, which makes the error models more unpredictable. As the speed of the devices becomes higher the relative size of the clock transition timing window increases and this makes devices more sensitive to SEU [3]. Similarly, decreased voltage levels for modern technologies make bit inversions more likely to occur [4].

The dangers of possible errors in memories resulting from SEUs are often mitigated with the use of linear single-error-correcting, double-error-detecting codes (SEC-DED). These codes have minimum Hamming distance four and are able to correct all single bit errors and detect all double bit errors. In the presence of multi-bit errors, however, the reliability of systems utilizing error protection architectures based on these codes may be questionable. For any linear SEC-DED codes with k information bits, the number of undetectable multi-bit errors is 2k. In addition to this, a huge number of multi-bit errors will be miscorrected. In the case where SEU results in multi-bit distortions with high probability, these codes may not be sufficient to provide a high reliability. Anomalies of systems caused by multi-bit upsets (MBU) have already been reported [5], [6].

The increase of the MBU rate in deep submicron technologies deteriorates the situation even further. In 65nm triple-well SRAMs with a thin cell architecture, the rate of multi-bit errors caused by neutron induced SEU increases by a factor of ten compared to that in 90 nm technologies nearly 55% of the errors due to neutron radiation were multi-bit errors [7]. Although there are mechanisms like bit interleaving [8] that can be used to minimize the error rate contribution of multi-bit errors, whether it is enough under such high MBU rate is still unknown. Moreover, the advantage of bit interleaving comes at a price of more layout constraints, which may result in larger power consumptions and longer access times. Thereby, memory protection architectures which can provide better protection against multi-bit errors than that based on classical linear codes are in demand.

Memories used in secure cryptographic devices not only suffer from random errors but are also vulnerable to errors injected by malicious fault injection attacks. It has been shown that the attacker can derive the secret key of the cryptographic devices thus break the security of the whole systems by injecting faults during encryption or decryption operations to force the devices working abnormally [9], [10].

Most of the existing reliable and secure memory architectures are based on linear codes, which concentrate their error detection and correction capabilities against certain types of errors, e.g., errors with small multiplicities [11], [12], [13]. The reliability and security of systems protected by linear codes largely depend on the accuracy of the expected error model. For memories used in cryptographic devices, the error model and the number of distorted bits introduced by the faults injected by the attacker are generally unpredictable due to the adaptive nature and the advanced fault injection methods available to an attacker. Thereby, the security of memories protected by linear codes cannot be guaranteed assuming the strongest attacker model.

As an alternative to linear codes, **robust codes** and their variants based on **nonlinear encoding functions** were proposed in [14], [15]. Different from linear codes, robust codes can provide nearly equal protection against all error patterns and are more suitable for applications where multi-bit errors are more probable or the error model is hard to predict. One limitation of robust codes is that these codes assume the information bits of messages or outputs of the device-to-be-protected are uniformly distributed and are not controllable by external forces, e.g., by an attacker during error

 ISBN: 978-1-61208-301-8

injection attacks on devices. The reliability and the security of the communication or computation channels protected by robust codes will be largely compromised if both information bits of the messages and the non-zero error patterns can be controlled by the attacker.

Intuitively, the limitation of robust code described above can be efficiently eliminated by introducing randomness into the encoding procedure of the code. Due to the fact that the random data are independent of the user information y, they can always be uniformly distributed. As a result, the assumption for robust codes that y is uniformly distributed is no longer required. Moreover, since the user has zero knowledge and no control of the random bits generated for each encoding operation, no matter how the attacker selects y and e, the probability that e is masked will be upper bounded by a number determined by the size of the set of possible random numbers. A coding technique based on adding to k information bits m random bits and r redundant bits, which overcomes the limitation of robust codes, is called strongly secure **algebraic manipulation detection (AMD)** code [16].

However, the constructions presented in [16] usually generate codes with a Hamming distance of 1, which cannot be used for error correction. While the resulting AMD codes are suitable for protecting the secure devices against fault injection attacks, in certain circumstances they may not be able to provide enough resistance against random transient errors introduced by the mother nature. The contribution of this paper is as following. In this paper, we propose new constructions of Algebraic Manipulation Correction (AMC) codes and efficient algorithms foe decoding for these codes. Comparing with our previous works [17], [18], [16], this code have a Hamming distance 4 and can correct all single-bit errors. Moreover, we describe a new error correction algorithm for the code which can also correct all double-bit errors with high probabilities. The proposed codes can provide a guaranteed level of security as well as a high level of reliability to the protected device, although the proposed scheme will require more redundancy, area and power. The problem of separating between single bit random errors and attacks for AMC codes is discussed in [17].

The rest part of the paper is organized as follows. The definitions of AMD codes and AMC codes are shown in Section II. In Section III-A, we describe the construction of the proposed AMC code and present the error correction algorithm for correcting random single and double errors. The hardware design and the analysis and comparison of overheads for the proposed codes are shown in Section IV.

II. Definitions

AMD codes are designed to provide a guaranteed level of security even if the attacker can control both the error patterns and the input (thus the fault-free output) of a device. Different from regular error control codes, a codeword of an AMD code contains three parts: k-bit user defined information y, m-bit random data x and r-bit redundancy $f(y,x)$.

Throughout the paper we denote by $\oplus$ the addition in $GF(q), q = 2^r$. All the results presented in the paper can be easily generalized to the case where $q = p^r$ (p is a prime). An AMD code V with codewords $(y, x, f(y,x))$, where $y \in GF(2^k), x \in GF(2^m)$ and $f(y,x) \in GF(2^r)$, will be referred to as a (k,m,r) code.

Definition 2.1: (**Security Kernel**) [16] For any (k,m,r) error detecting code V with the encoding function $f(y,x)$, where $y \in GF(2^k), x \in GF(2^m)$ and $f(y,x) \in GF(2^r)$, the **security kernel** K_S is the set of errors $e = (e_y, e_x, e_f), e_y \in GF(2^k), e_x \in GF(2^m), e_f \in GF(2^r)$, for which there exists y such that $f(y \oplus e_y, x \oplus e_x) \oplus f(y,x) = e_f$ is satisfied for all x.

$$K_S = \{e | \exists y, f(y \oplus e_y, x \oplus e_x) \oplus f(y,x) \oplus e_f = \mathbf{0}, \forall x\}. \quad (1)$$

For cryptographic devices and secure applications, non-zero errors e in the security kernel can be used by an advanced attacker to bypass the protection based on the error detecting code V. For any error $e^* = (e_y^*, e_x^*, e_f^*) \in K_S, e^* \neq \mathbf{0}$, where $\mathbf{0}$ is all zero vector, there exists y^* (the protected information at the output of the device) such that for this y^* the error e^* is not detected for any choice of the random variable x (the probability of not detecting e^* for the information y^* is equal to 1). Thus to conduct a successful attack, it is sufficient for the attacker to inject $e^* \in K_S$ when the expected output is in the format of $(y^*, x, f(y^*, x))$. An AMD code should have no errors in the security kernel except for the all zero vector in $GF(2^n)$, where $n = k + m + r$.

Definition 2.2: [19] A (k,m,r) error detecting code is called Algebraic Manipulation Detection (AMD) code iff $K_S = \{\mathbf{0}\}$, where $\mathbf{0}$ is the all zero vector in $GF(2^n)$, $n = k + m + r$.

There are no undetectable errors (errors that are undetected with a probability of 1) for AMD codes. For any y and any e, the error masking probability for an AMD code V can be computed as

$$\begin{aligned} Q_V(y,e) \quad = \quad & 2^{-m} | \{x \,|\, (y, x, f(y,x)) \in V, \\ & (y \oplus e_y, x \oplus e_x, f(y,x) \oplus e_f) \in V\} |, \end{aligned} \quad (2)$$

which is the fraction of random m-bit vectors x that will mask a fixed error e for a given y. The security level of the system protected by AMD code can be characterized by the worst case error masking probability $Q_V = \max_y \max_{e \neq 0} Q_V(y,e)$.

The general architecture of a computational channel (device) protected by a (k,m,r) AMD code is shown in Figure 1, where RNG is a random number generator (either in software or hardware) and EDN is the error detection network.

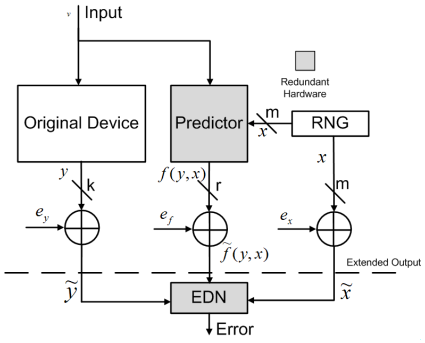


Fig. 1. Computation channel protected by a systematic (k,m,r) AMD code (original device and the predictor may be under attack).

The definition of AMD code can be found in [20], [16]. The existing AMD codes [20], [16] are designed for error detection and have Hamming distances less than 3. In this paper, we will construct Algebraic Manipulation Correction (AMC) codes that can be used not only for error detection but also for error correction. These codes can be used for design of reliable and secure memories where error correction is indispensable for restoring data distorted by natural effects such as soft errors. The formal definition of AMC codes is as follows.

Definition 2.3: An AMD code with Hamming distance at least 3 is called an Algebraic Manipulation Correction (AMC) code.

III. Proposed Single-Error-Correcting AMC Codes

A. Construction of the proposed single-error-correcting AMC codes

In this Section, we will present the general construction of AMC codes and two error correction algorithms. The first algorithm can correct all single-bit errors and detect all double-bit errors. The second algorithm can correct not only single-bit errors but can also correct double-bit errors with high probabilities at the cost of higher hardware overhead.

Theorem 3.1: Suppose

1) (x, xP) is a codeword of $(m + r_H, m, 3)$ binary linear Hamming code V_H with r_H redundant bits and distance 3, where $x \in GF(2^m)$, $xP \in GF(2^{r_H})$, P is a $r_H \times m$ encoding matrix, and
2) $f(y, x) \in GF(2^m)$ is a nonlinear encoding function $f(y,x) = y_1x \oplus y_2x^2 \oplus y_3x^3 \oplus \cdots \oplus y_bx^b \oplus x^{b+2}$ (b is odd, $b+2 < 2^m-1$). where $y = (y_1, y_2, \ldots, y_b)$; $y_i \in GF(2^m), (i = 1, 2, \ldots, b)$; $x \in GF(2^m)$; $f(y, x) \in GF(2^m)$ and all the operations are in $GF(2^m)$; $2^m - 1$ is a prime number;
3) $\pi y = y_1 \oplus y_2 \oplus y_3 \oplus \cdots \oplus y_b \in GF(2^m)$ is the byte-wise parity of y;

Then the code $V_{AMC} = \{(y, \pi y \oplus x, xP, f(y, x))\}$ is a $(k, m, m + r_H)$ SEC *Algebraic Manipulation Correction* (AMC) code, with $k = bm$ information bits, m random bits, $m + r_H$ redundant bits.

This code has secure kernel $K_{V_{AMC}} = \{\mathbf{0}\}$ with the maximum error masking probability $Q_{V_{AMC}} = (b + 1)(2^m - 2)^{-1}$ and Hamming distance 3.

We note that the proposed AMC code is a combination of $(y, x, f(y, x))$ AMD code which provides for secure kernel $\{\mathbf{0}\}$ and (x, Px) Hamming code which provides for distance 3. This construction is similar to Vasil'ev nonlinear perfect code [21].

Remark 3.1: We may use any AMD encoding functions $f(y, x)$ described in [20], [16]. In general case, $y \in GF(2^k)$, where $k = sr$, and $x \in GF(2^m)$, where $m = tr$. In this case, $x = (x_1, x_2, \ldots, x_t)$ and $f(y, x)$ is a polynomial of t variables $x_1, \ldots, x_t$. Thus, y will be divided into s/t parts, each of which contains tr bits, and then $\pi y \in GF(2^{tr})$ is the byte-wise parity. The padding zeros may be applied to y, when s is not dividable by t.

Example 3.1: Let $m = 7$, which is the number of random bits. Also let the encoding function be $f(y, x) = y_1x \oplus y_2x^2 \oplus y_3x^3 \oplus y_4x^4 \oplus y_5x^5 \oplus x^7$, where $y = (y_1, y_2, \ldots, y_5) \in GF(2^{35})$ is the information part, $y_i \in GF(2^7)$ for $i = 1, 2, \ldots, 5$, $x \in GF(2^7)$ is the random number.

Let $\{(x, xP)\}$ be the $(11, 7, 3)$ Hamming code, where P is the encoding matrix for the Hamming code. Since $2^7 - 1$ is a prime number, the code V_{AMC} defined by Theorem 3.1 is an AMC code with $d = 3$ and $Q_{V_{AMC}} = \frac{6}{2^7-2} = \frac{1}{21}$.

Comparing to the AMD Code with $k = br = 35$, $m = r = 7$ and nonlinear encoding function $f(y, x) = y_1x \oplus y_2x^2 \oplus y_3x^3 \oplus y_4x^4 \oplus y_5x^5 \oplus x^7$, the codeword of V_{AMC} contains 4 more redundant bits.

Remark 3.2: In a normal base Galois field [20], square operation can be achieved by the cyclic shift. As a result, $f(y, x)$ in Theorem 3.1 can be slightly modified to reduce its hardware complexity of computing $f(y, x)$ using the following encoding equation

$$f(y, x) = y_1x \oplus y_2x^2 \oplus y_3x^4 \oplus \cdots \oplus y_bx^{2^{(b-1)}} \oplus x^{2^b+1},$$

where $y = (y_1, y_2, y_3, \ldots, y_b)$ and $y_i \in GF(2^m)$ $(i = 1, 2, 3, \ldots b)$; $x \in GF(2^m)$; $x \neq \mathbf{0}, \mathbf{1}$, where $\mathbf{1}$ is all 1 vector; $f(y, x) \in GF(2^m)$; and $2^m - 1$ is a prime number and $b < m$.

This code reduces the computational complexity of decoding at the cost of higher error masking probability, which is going up to $Q_{V_{AMC}} = 2^b(2^m - 2)^{-1}$.

B. Algorithm for Single Error Correction and Estimation of Probabilities of Miscorrection for the Proposed Codes

A direct approach is to add codewords to an existing AMD code some additional redundant bits to provide for error correction, $(y, x, f(y, x), P)$ as an example. We will present another approach which can detect and correct the errors in the codewords but will requires less redundant bits.

1) Error correction algorithm for the proposed SEC-DED AMC code: There are four parts in every codeword of the AMC code constructed as in Theorem 3.1, namely y, $\pi y \oplus x$, xP, and $f(y, x)$. For a codeword $v = (v_1, v_2, v_3, v_4)$ of the AMC code V_{AMC} constructed in Theorem 3.1, there are

$v_1 = y = (y_1, y_2, y_3, \ldots, y_b)$; $y_i \in GF(2^m)$, $i = 1, 2, 3, \ldots, b$;
$v_2 = \pi y \oplus x$; $\pi y,\ x,\ v_2 \in GF(2^m)$;
$v_3 = xP$; $xP \in GF(2^{r_H})$;
$v_4 = f(y, x)$; $v_4 \in GF(2^m)$;

(x, xP) is a codeword of a linear Hamming code with distance 3 and the check matrix is $H = [P^T|I]$, where P^T is the transposed matrix of P and I is an identity matrix.

Denote the error vector by $e = (e_1, e_2, e_3, e_4)$ and the received message by $\tilde{v} = (\tilde{v_1}, \tilde{v_2}, \tilde{v_3}, \tilde{v_4})$, where $\tilde{v_i} = v_i \oplus e_i, i = 1, 2, 3, 4$ and $e_1, \tilde{v_1} \in GF(2^{bm})$; $e_2, \tilde{v_2} \in GF(2^m)$; $e_3, \tilde{v_3} \in GF(2^{r_H})$; $e_4, \tilde{v_4} \in GF(2^m)$. We assume that we only need to correct the errors in the information part $v_1 = y$. The decoding procedure can be divided into the following steps.

1) Calculate $(\tilde{u}, \tilde{v_3})$, where $\tilde{u} = \pi\tilde{v_1} \oplus \tilde{v_2}$
2) Calculate $S_H = H(\tilde{u}, \tilde{v_3})^T$, the syndrome for the Hamming code.
   Use S_H as the input to the Hamming decoder, then obtain the error locator ε, where $\varepsilon \in GF(2^m)$. Since ε is the output of the Hamming decoder, there should be only one bit in ε which is equal to one, and all other bits are zeros.
   Let $u = \tilde{u} \oplus \varepsilon = \pi\tilde{v_1} \oplus \tilde{v_2} \oplus \varepsilon$, where $u \in GF(2^m)$. If uncorrectable multi-bit errors are detected by the Hamming decoder, then no further steps need to be performed. Otherwise, go to the step 3.
3) Calculate S_{AMD} as follows

$$\begin{aligned} S_{AMD} &= f(\tilde{y}, u) \oplus \tilde{v_4} \\ &= f(y \oplus e_1, x \oplus \pi e_1 \oplus e_2 \oplus \varepsilon) \oplus f(y, x) \oplus e_4. \end{aligned} \quad (3)$$

   If both $S_H = \mathbf{0}$ and $S_{AMD} = \mathbf{0}$, then there are no errors. If only $S_{AMD} = \mathbf{0}$, there are multiple errors. Therefore, as long as $S_{AMD} = \mathbf{0}$, the correction procedure is completed. Otherwise go to the next step.
4) Compare S_{AMD} with εu^j, for all $j = 1, 2, 3, \ldots, b$.
   a) If $S_{AMD} = \varepsilon u^j$ for some $j \in \{1, 2, 3, \ldots, b\}$, then the j^{th} part $y_j \in GF(2^m)$ of information $\tilde{v}_1 \in GF(2^{bm})$ of the codeword is distorted and the error in that part is $\varepsilon \in GF(2^m)$, which means $\hat{y_j} = \tilde{y_j} \oplus \varepsilon$, where $\hat{y_j} \in GF(2^m)$ is the corrected message.
   b) Otherwise, there are multiple errors or the error is not in the information part v_1. No error correction will be attempted.

The decision table for the proposed single error correction algorithm is summarized in Table I

Example 3.2: (Single Error Correction)
Consider a proposed AMC code with $b = 2$, $m = 3$. V_H is a $(6, 3, 3)$

TABLE I
CORRECTION ALGORITHM DECISION TABLE FOR SEC-DED AMC

S_H	S_{AMD}	Decision
$S_H = \mathbf{0}$	$S_{AMD} = \mathbf{0}$	No Error
	$S_{AMD} \neq \mathbf{0}$	Double/Multiple Errors
$S_H \neq \mathbf{0}$ and $S_H \neq h_i$ [I] $(\forall i)$	$\forall S_{AMD}$	Double/Multiple Errors
$S_H = h_i$	$S_{AMD} \neq \varepsilon u^j$ or $S_{AMD} = \mathbf{0}$	Single Error in v_2, v_3, or v_4 Or Double/Multiple Errors
	$S_{AMD} = \varepsilon u^j$ $S_{AMD} \neq \mathbf{0}$	Single Error in v_1 (Correction)

[I] h_i $(1 \leq i \leq m)$ is the i^{th} column of the parity check matrix H of the Hamming code. This row is only valid for non-perfect Hamming code.

Hamming code with $P = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$.

The encoding function is $f(y,x) = y_1 x \oplus y_2 x^2 \oplus x^5$. The codeword is in the format of $v = ((y_1, y_2), \pi y \oplus x, xP, f(y,x))$, where y_1, y_2, x, $xP, f(y,x) \in GF(2^3)$. We select $z^3 \oplus z \oplus 1$ as the generating polynomial for $GF(2^3)$, with the rightmost bit being the least significant bit.

Suppose $y_1 = (001)$, $y_2 = (001)$, $x = (010)$. Then we have $\pi y \oplus x = (001) \oplus (001) \oplus (010) = (010)$, $(xP)^T = (101)$, and $f(y,x) = (001)(010) \oplus (001)(010)^2 \oplus (010)^5 = (010) \oplus (100) \oplus (111) = (001)$. Thus, the original codeword is $v = (v_1, v_2, v_3, v_4) = (001001, 010, 101, 001)$.

Suppose there is a single error $e = (000010, 000, 000, 000)$ in the received message. Therefore, the distorted message is $\tilde{v} = (001011, 010, 101, 001)$. We have $(\tilde{u}, \tilde{v_3}) = (\pi \tilde{v_1} \oplus \tilde{v_2}, \tilde{v_3}) = (000, 101)$.

$S_H = H(\tilde{u}, \tilde{v_3})^T = [P^T | I](\tilde{u}, \tilde{v_3})^T = (101)$. After decoding $(\tilde{u}, \tilde{v_3})$ using the Hamming decoder, we have $\varepsilon = (010)$. And $u = \pi \tilde{v_1} \oplus \tilde{u} \oplus \varepsilon = (000) \oplus (000) \oplus (010) = (010)$. Then syndrome $S_{AMD} = (001)(010) \oplus (011)(010)^2 \oplus (010)^5 = (011)$. Since $S_{AMD} = \varepsilon u^2 = (010)(010)^2 = 011$, the error $\varepsilon = (010)$ is located at second bit of $\tilde{y_2}$.

The error is successfully corrected.

2) Estimations on a probability for the miscorrection: Suppose $v = (v_1, v_2, v_3, v_4)$, where $v_1 = y$, $v_2 = \pi y \oplus x$, $v_3 = xP$, $v_4 = f(y,x)$ is a codeword for an AMC code V_{AMC} described in Theorem 3.1. Let $e = (e_1, e_2, e_3, e_4)$ be the error vector and $\tilde{v} = \{\tilde{v_1}, \tilde{v_2}, \tilde{v_3}, \tilde{v_4}\}$ be the received (distorted) message, where $\tilde{v_i} = v_i \oplus e_i, i = 1, 2, 3, 4$. Let $e_1 = (e_{11}, e_{12}, \ldots, e_{1b})$, where $e_{1i} \in GF(2^m)$ for $i \in \{1, 2, \ldots, b\}$. Denote the message after correction, i.e., the output of the decoder by $\hat{v} = (\hat{v_1}, \hat{v_2}, \hat{v_3}, \hat{v_4})$, where $e_1, \tilde{v_1}, \hat{v_1} \in GF(2^{bm})$, e_2, $\tilde{v_2}$, $\hat{v_2} \in GF(2^m)$; e_3, $\tilde{v_3}$, $\hat{v_3} \in GF(2^{r_H})$; e_4, $\tilde{v_4}$, $\hat{v_4} \in GF(2^m)$.

We say that the error is miscorrected if $c_1 \neq \hat{c_1}$. The miscorrection probability can be defined as

$$Q_{mc}(y,e) = |\{x | v_1 \neq \hat{v_1}, e \neq 0\}| 2^{-m}, \quad (4)$$

where 2^m is the number of possible values of x.

Theorem 3.2: Miscorrection Probability.

For the AMC code constructed by Theorem 3.1, the algorithm in Section III-B1 has a miscorrection probability $Q_{mc}(y,e)$, at most $b(b+1)(2^m-2)^{-1}$, $\max_{y;e\neq 0} Q_{mc}(y,e) \leq b(b+1)(2^m-2)^{-1}$.

The proof of this theorem is based on the analysis of the algorithm presented in Section III-B for the case $S_H \neq 0$, $S_H \neq h_i$ for each i (see Table I).

The AMC code for Theorem 3.1 can be extended to be a code with Hamming distance 4 by adding one more overall parity bit after which the code can correct single error and at the same time detect all double errors without miscorrection of double errors. The error detection and correction capabilities for the extended SEC-DED AMC code is summarized in Table II.

TABLE II
ERROR DETECTION AND CORRECTION CAPABILITIES FOR SEC-DED AMC CODE

Number of errors	Error in parity	Errors in v_1	Errors in v_2 and/or v_3	Errors in v_4 [I]
Single	Detected	Corrected	Detected [II]	Detected
Double	Detected. No miscorrection.			
Multiple even	Detected with a probability $1 - Q_{V_{AMC}}$. [III] No miscorrection.			
Multiple odd	Detected with a probability $1 - Q_{mc}$. [IV] Miscorrected with a probability Q_{mc}.			

[I] If errors are located only in the v_4, no errors in the other parts of codeword c, these errors will always be detected.

[II] Here if we assume there is only a single error, then when the error is not in v_1, it is in v_2 or v_3 and can be corrected.

[III] $Q_{V_{AMC}}$ is the maximum error masking probability. $Q_{V_{AMC}} = (b+1)2^{-m}$

[IV] Q_{mc} is the error miscorrection probability. $Q_{mc} \leq b(b+1)2^{-m}$

Remark 3.3: We note that the straightforward concatenation approach for construction of AMC codes with distance 3 based on adding redundant bits to AMD code requires more redundancy than codes constructed by Theorem 3.1.

For the straightforward concatenation approach $(y, x, f(y,x), P)$, the redundant parts are x, $f(y,x)$ and P, in which P is the Hamming redundant part for $(y, x, f(y,x))$ as the information part. Which means the redundant bits are $2m + \lceil log_2(mb + 2m + 1) \rceil$ for the straightforward concatenation approach. For the proposed architecture, the redundant parts are $\pi \oplus x$, xP and $f(y,x)$, the redundant bits number is $2m + \lceil log_2(m+1) \rceil$. For large b, the proposed architecture will save much more area than the straightforward concatenation approach.

C. Double Error Correction Algorithm and Estimations on the Probability of Miscorrection

1) Error correction algorithm: We note that the proposed AMC codes with overall linear parity bit can be used to correct double errors with a multiple-iteration algorithm. In this case the syndrome S_H for a Hamming code with distance 3 could be the sum of two columns h_{i_1} and h_{i_2} of the check matrix H, where i_1 and i_2 indicate the location of two errors.

For the proposed code, besides the syndrome for the Hamming code, we may use the syndrome for the AMD code to verify the location of the double error. After computing the syndrome, we can try all pairs (i_1, i_2) such that $h_{i_1} + h_{i_2} = S_H$ and if for one of them the corresponding AMD syndrome $S_{AMD} = 0$, it indicates that the double error is at the position i_1 and i_2.

We need to try all pairs of h_{i_1} and h_{i_2} with the same sum $S_H = h_{i_1} + h_{i_2}$. To achieve smaller number of iterations, we would like to make the number of pairs as small as possible. For a non-prefect Hamming code, we would like to select the check matrix H, such

 ISBN: 978-1-61208-301-8

that

$$\min_{H} \max_{S_H} |\{(i_1, i_2)|S_H = h_{i_1} \oplus h_{i_2}\}|, \quad (5)$$

where $\max_{S_H} |\{(i_1, i_2)|S_H = h_{i_1} \oplus h_{i_2}\}|$ indicates the maximum possible number of iterations.

We are able to pre-compute a lookup table for a fixed H for a Hamming code to simplify the locating of double errors for a given S_H.

Example 3.3: For a (11,7,3) shortened Hamming code with check matrix

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

we have $\max_{S_H} |\{(i_1, i_2)|S_H = h_{i_1} \oplus h_{i_2}\}| = 3$.

For another (11,7,3) shortened Hamming code with check matrix

$$H = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

we have $\max_{S_H} |\{(i_1, i_2)|S_H = h_{i_1} \oplus h_{i_2}\}| = 2$

Clearly, the second check matrix results in a smaller number of iterations to find the corrected error vector.

There are five parts (with the overall linear parity bit) in every codeword of the AMC code constructed as in Theorem 3.1, namely y, $\pi y \oplus x$, xP, $f(y, x)$ and the overall parity. For a codeword $v = (v_1, v_2, v_3, v_4, v_5)$ of the extended AMC code V_{AMC} constructed in Theorem 3.1, there are

$v_1 = y = (y_1, y_2, y_3, \ldots, y_b)$; $y_i \in GF(2^m)$, $i = 1, 2, 3, \ldots, b$;
$v_2 = \pi y \oplus x$; $\pi y,\ x,\ c_2 \in GF(2^m)$;
$v_3 = xP$; $xP \in GF(2^{r_H})$;
$v_4 = f(y, x)$; $v_4 \in GF(2^m)$;
$v_5 = \Pi(v_1, v_2, v_3, v_4)$

(x, xP) is a codeword of a linear Hamming code with distance 3 and the check matrix is $H = [P^T|I]$, where P^T is the transposed matrix of P and I is an identity matrix. v_5 is the overall linear parity bit.

Denote the error vector by $e = (e_1, e_2, e_3, e_4, e_5)$ and received message by $\tilde{v} = (\tilde{v_1}, \tilde{v_2}, \tilde{v_3}, \tilde{v_4}, \tilde{v_5})$, where $\tilde{v_i} = v_i \oplus e_i, i = 1, 2, 3, 4, 5$ and $e_1, \tilde{v_1} \in GF(2^{bm})$; $e_2, \tilde{v_2} \in GF(2^m)$; $e_3, \tilde{v_3} \in GF(2^{r_H})$; $e_4, \tilde{v_4} \in GF(2^m)$; $e_5, \tilde{v_5} \in GF(2)$. We assume that we only need to correct the errors in the information part $c_1 = y$. The decoding procedure can be divided into the following steps.

1) if the the overall parity indicates that there are errors with even multiplicities, go to step 2, otherwise it may be a single error.
2) calculate $(\tilde{u}, \tilde{v_3})$, where $\tilde{u} = \pi\tilde{v_1} \oplus \tilde{v_2}$
3) calculate $S_H = H(\tilde{u}, \tilde{v_3})^T$, the syndrome for the Hamming code.
4) use S_H as the input to a lookup table, then we can obtain some pairs of error locators ε_{i_1}, and $\varepsilon_{i_2}$where $\varepsilon_{i_1}, \varepsilon_{i_2} \in GF(2^m)$, for some i_1 and i_2, such that $S_H = h_{i_1} \oplus h_{i_2}$, where h_{i_1} and h_{i_2} are the i_1^{th} and i_2^{th} column respectively of the Hamming check matrix H.
   ε_{i_1} (as well as ε_{i_2}) is an m-bit vector with 1 in the only i_1^{th} (or i_2^{th}) bit and 0 in all other bits.
   Let $u = \tilde{u} \oplus \varepsilon_{i_1} \oplus \varepsilon_{i_2} = \pi\tilde{v_1} \oplus \tilde{v_2} \oplus \varepsilon_{i_1} \oplus \varepsilon_{i_2}$, where $u \in GF(2^m)$.
   If there are no such pairs in the look-up table, then no further steps need to be performed. The error is not correctable by this algorithm. Otherwise, go to the step 5.
5) calculate S_{AMD} as follows

$$\begin{aligned} S_{AMD} &= f(\tilde{y}, u) \oplus \tilde{v_4} \\ &= f(y \oplus e_1, x \oplus \pi e_1 \oplus e_2 \oplus \varepsilon_{i_1} \oplus \varepsilon_{i_2}) \oplus f(y, x) \oplus e_4. \end{aligned} \quad (6)$$

   If both $S_H = \mathbf{0}$ and $S_{AMD} = \mathbf{0}$, then there are no errors. Therefore, as long as $S_{AMD} = \mathbf{0}$, the correction procedure is completed. Otherwise go to the next step.
6) Compare S_{AMD} with $\varepsilon_{i_1} u^{j_1} \oplus \varepsilon_{i_2} u^{j_2}$, for all $j_1, j_2 \in \{1, 2, 3, \ldots, b\}$. If

$$S_{AMD} = \varepsilon_{i_1} u^{i_1} \oplus \varepsilon_{i_2} u^{j_2} \quad (7)$$

   for some $j_1, j_2 \in \{1, 2, 3, \ldots, b\}$, then the $y_{j_1}, y_{j_2} \in GF(2^m)$ of information $\tilde{v}_1 \in GF(2^{bm})$ of the codeword are distorted and the error in those parts are $\varepsilon_{i_1}, \varepsilon_{i_2} \in GF(2^m)$ respectively. That means $\hat{y}_{j_1} = \tilde{y}_{j_1} \oplus \varepsilon_{i_1}$, and $\hat{y}_{j_2} = \tilde{y}_{j_2} \oplus \varepsilon_{i_2}$,where $\hat{y}_{j_1}, \hat{y}_{j_2} \in GF(2^m)$ are the corrected messages.
7) If no j_1, j_2 are found, go back to step 4, try another pair $\varepsilon_{i_1}, \varepsilon_{i_2}$ for the same S_H.
8) If no $\varepsilon_{i_1}, \varepsilon_{i_2}$ satisfy (7), then the errors are not correctable for this algorithm.

Example 3.4: Consider an extended proposed AMC code with $b = 2$, $m = 7$. V_H is a $(11, 7, 3)$ Hamming code with

$$H = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The encoding function is $f(y, x) = y_1 x \oplus y_2 x^2 \oplus x^5$. The codeword is in the format of $v = ((y_1, y_2), \pi y \oplus x, xP, f(y, x), \pi_v)$, where y_1, y_2, x, $f(y, x) \in GF(2^7)$, $xP \in GF(2^4)$ and π_v is the overall parity. We select $z^7 \oplus z^3 \oplus 1$ as the generating polynomial for $GF(2^7)$, with the rightmost bit being the least significant bit.

Suppose $y_1 = (0000110)$, $y_2 = (0000011)$, $x = (0000010)$. Then we have $\pi y \oplus x = (0000110) \oplus (0000011) \oplus (0000010) = (0000111)$, $xP = (0110)$, and $f(y, x) = (0000110)(0000010) \oplus (0000011)(0000010)^2 \oplus (0000010)^5 = (0100000)$.

Thus, the original codeword is $v = (v_1, v_2, v_3, v_4, v_5) = ((0000110, 0000011), 0000111, 0110, 0100000, 1)$. Suppose there is a double error $e = ((0000001, 0001000), 0000000, 0000, 0000000)$ in the received message. Therefore, the distorted message is $\tilde{v} = ((0000111, 0001011), 0000111, 0110, 0100000, 1)$. We have $(\tilde{u}, \tilde{v_3}) = (\pi\tilde{v_1} \oplus \tilde{v_2}, \tilde{v_3}) = (0001011, 0110)$. $S_H = H(\tilde{u}, \tilde{v_3})^T = [P^T|I](\tilde{u}, \tilde{v_3})^T = (0101)^T$. The overall pairty shows that there is an error with a number of erroneous bits being even.

Since the rightmost bit is the least significant bit, we have the following lookup table for error locations.

S_H	i_1	i_2
0101	1	4
	2	7

Then let $i_1 = 1$ and $i_2 = 4$, then $\varepsilon_{i_1} = (0000001)$ and $\varepsilon_{i_2} = (0001000)$. We have $u = \tilde{u} \oplus \varepsilon_{i_1} \oplus \varepsilon_{i_2} = (0000010)$, and $S_{AMD} = \tilde{v_4} \oplus f(\tilde{v_1}, u) = (0100010)$. Let $j_1 = 1$ and $j_2 = 2$, we have $\varepsilon_{i_1} u^{j_1} \oplus \varepsilon_{i_2} u^{j_2} = (0000001)(0000010) \oplus (0001000)(0000100) = (0100010)$.

Therefore we know that the error ε_{i_1} locates at y_1 and ε_{i_2} locates at y_2. Actually, since $u = (0000010)$, if $i_1 = 1$ and $i_2 = 4$, then $S_{AMD} = (0100010) = \varepsilon_{i_1} u^{j_1} + \varepsilon_{i_1} u^{j_1}$ only if $j_1 = 1$ and $j_2 = 2$. If $i_1 = 2$, $i_2 = 7$, then $\varepsilon_{i_1} u^{j_1} + \varepsilon_{i_1} u^{j_1} = \varepsilon_2 u^{j_1} + \varepsilon_7 u^{j_2} \neq$

$S_{AMD} = (0100010)$ for any j_1, j_2. The corrected message is $\hat{v} = ((0000110, 0000011), 0000111, 0110, 0100000, 1)$.

In this example, we need at least 7 clock cycles to find the double error for the best case. Correspondingly, we will need $2+17*(14+3) = 291$ cycles to locate the double error for the worst case.

2) Estimations on the probability of miscorrection: In this section we estimate the upper bound for the probability that any error is miscorrected into a double error in the information part.

Theorem 3.3: For the algorithm of double error correction presented in Section III-C, the miscorrection probability Q_{mc} for any given pair e and y will be $Q_{mc} \leq (2^m - 2)^{-1} max(b^2 n_p(b+1),\ 0.5b(b-1)m(b+1))$.

The estimated Q_{mc} gives the upper bound for the miscorrection probability. Note that, a double error in the information part may be also miscorrected into another double error in the information part with the same miscorrection probability.

Example 3.5: (Miscorrection Probability) For an extended proposed AMC code with $b = 2$, $m = 7$, the check matrix for the $(11, 7, 3)$ Hamming code is the same as in Example 3.4. The encoding function is $f(y, x) = y_1 x \oplus y_2 x^2 \oplus x^5$, we have $n_p = 2$. Since $b^2 n_p(b+2) = 32 > 0.5b(b-1)m(b+2) = 28$, the maximum miscorrection probability is $Q_{mc} = (b^2 n_p(b+2))(2^m - 2)^{-1} = \frac{32}{126}$ for a given error e and a given information part y.

We conducted simulation experiments to estimate the probabilities of miscorrecting one double error into another one for $m = 2, b = 2$ and H as defined above with $n_p = 3$. All possible combinations of y and e are tested to see the number of random numbers x such that the errors are miscorrected.

The real miscorrection probability is $Q_{mc} = \frac{10}{126} \approx 0.08$, which is smaller than the estimated miscorrection probability is $Q_{mc} = \frac{32}{126}$.

Example 3.6: For an extended proposed AMC code with $b = 4$, $m = 17$, the check matrix for the $(22, 17, 3)$ Hamming code $H = [22, 21, \cdots, 2, 1]$, and encoding function $f(y, x) = y_1 x \oplus y_2 x^2 \oplus x^5$, we have $n_p = 6$.

Since $b^2 n_p(b+2) = 576 \leq 0.5b(b-1)m(b+2) = 612$, the maximum miscorrection probability is $Q_{mc} = (0.5b(b-1)m(b+2))(2^m - 2)^{-1} = \frac{612}{2^{17}-2}$ for a given error e and a given information part y.

A simulation was conducted to test the real miscorrection probability for a double error in the information part is miscorrected into another double error in the information part. $10,000$ random combinations of y and e have been tested to see the number of random numbers x such that the errors are miscorrected.

The evaluated miscorrection probability is $Q_{mc} = \frac{131}{2^{17}-2} \approx 0.001$, which is again much smaller than the estimated miscorrection probability $Q_{mc} = \frac{612}{2^{17}-2}$. We also note this miscorrection probability for double errors is small and converges to 0 very fast as the number of random bits m grows, thus the proposed code may be used for double-error-correction.

We note that compared with the direct construction $(y, x, f(y, x), P)$, the proposed architecture can protect the $f(y, x)$ at the same time while requires less redundant bits.

IV. Hardware Design of Reliable and Secure Memories Based on the Proposed AMC Codes

A. Hardware implementation

Figure 2 presents the architecture for memories protected by the proposed code in Theorem 3.1. In cryptographic applications, the m random digits can be generated by a random number generator (RNG) which is already integrated in most of the modern cryptographic devices. During a WRITE operation, the encoder loads the information bits and the random bits, then generates the redundant bits which are saved in the redundant memory block. The redundant bits are split into two parts, the linear part $(\pi y \oplus x, xP)$ which contains $m + r_H$ bits and the nonlinear part $f(y, x)$ which contains m bits. During a READ operation, the Error Correcting Network block computes the syndrome of the retrieved data and executes the error correction algorithm. If uncorrectable errors occur, ERR will be asserted and no correction will be attempted.

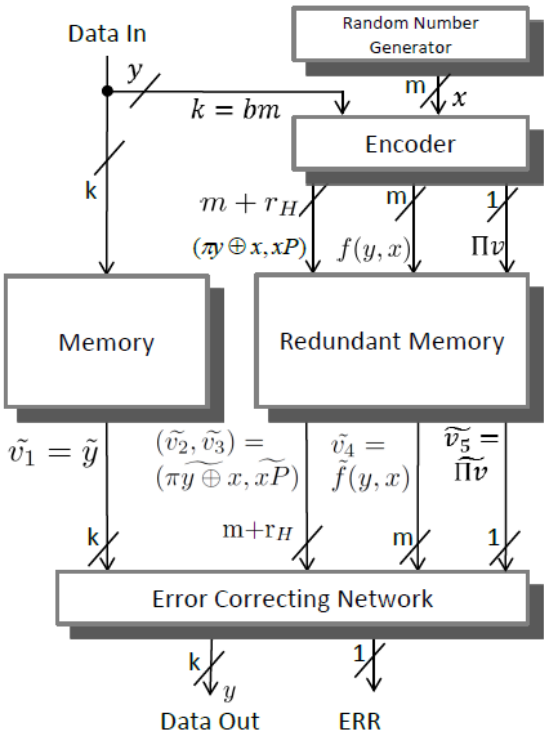


Fig. 2. Memory protected by the proposed code (the memory, redundant memory and the encoder may be under attack).

The encoder for the proposed code is presented in Figure 3. The Πv is the overall parity check of the system. The inputs to the encoder are the information bits y and the random number x. The encoder for the proposed AMC code contains the Hamming encoder and **the AMD encoder** to calculate $f(y, x)$. The AMD encoder requires b Galois field multipliers for $GF(2^m)$ to calculate the products of y and x^i and additional circuits to generate x^i.

The proposed code with distance 4 is a SEC-DED code, and only single error in the information part will be corrected. So if there are multiple errors or the single-bit error is not in the information part, the decoder sends the uncorrected messages to the output ports, and set the error flag (ERR). (ERR should be asserted iff there are uncorrectable errors.) The error flag signal will be used for the error handling, which is discussed in previous part of this paper. If there are even number errors, i.e., the overall parity bit c_5 is zero, no correction will be attempted.

To calculate the products between y_i and x^i in $f(y, x)$, for $i \in \{1, 2, \ldots, b\}$, b multipliers in $GF(2^m)$ are required. The computation of εu^i requires another b multipliers. We note that multipliers can be reused to reduce the area complexity at the cost of a larger decoding latency as more clock cycles are required. However, due to the reused hardware, a finite state machine should be introduced. If we do not reuse the multipliers, the architecture could be pipelined.

 ISBN: 978-1-61208-301-8

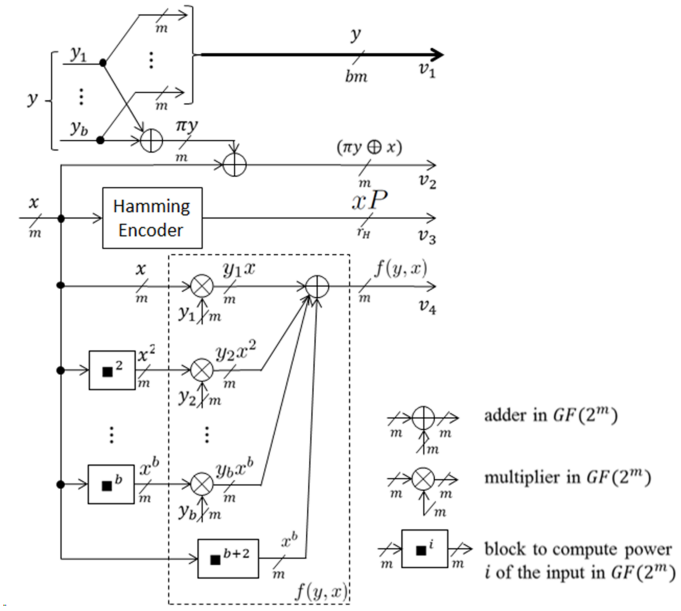


Fig. 3. Encoder without overall parity check bit for the proposed code

B. Comparison of the secure SEC-DED memories based on proposed codes with memories based on the known codes

In this section we compare the security levels in terms of the numbers of undetectable errors, sizes of security kernels and miscorrection probabilities, for the proposed codes V_{AMC} of distance 4 with additional overall check parity to the known nonlinear SEC-DED robust codes (extended Vasil'ev and extended Phelps) presented in [22]. The hardware overheads in terms of area, power and latency for encoders and decoders of different codes are also compared in this section. Transmission rates are also compared. All codes have distance 4. All the robust codes compared are $(39, 32, 4)$ codes [22].

For the proposed code, the number of information bits $k = bm = 35$, the number of random bits is $m = 7$, the degree of $f(y, x)$ is $b = 5$, the Hamming code used is a $(11, 7, 3)$ Hamming code. Overall parity is added to achieve Hamming distance four.

There are $35+7+7 = 49$ bits for the AMD part of the codes. The Hamming code used for error correction part is a $(55, 49, 3)$ code. Overall parity is also added to achieve Hamming distance four.

TABLE III
SIZES OF DETECTION KERNELS AND SECURITY KERNELS AND MISCORRECTION PROBABILITIES FOR DIFFERENT CODES

Codes	$\|K_D\|$ [I]	$\|K_V\|$ [II]	Q_{mc} [III]
Extended Hamming [IV]	2^{32}	$32(2^{32}-1)$	1
Extended Vasil'ev [IV]	2^{6}	$\approx 12(2^{32}-1)$	0.5
Extended Phelps [IV]	2^{27}	$32(2^{27}-1)$	1/16
Proposed code	0	0	30/126

[I] $|K_D|$ is the number of undetectable errors (size of detection kernel)
[II] $|K_V|$ is the size of security kernel
[III] Q_{mc} is the miscorrection probability
[IV] Data are obtained from [22]

Table III compares the sizes of detection kernels, i.e., the number of undetected errors, the sizes of security kernels and the miscorrection probability for the Hamming code, robust codes from [22], and the extended proposed code V_{AMC} with above parameters. It follows from the table that the proposed code is the best one providing good security for the strong attack model, as the code has zero size of the security kernel. The miscorrection probability for the proposed code is slightly worse than for the extended Phelps code, but it is still better than for the extended Vasil'ev code [22] and the Hamming code. Moreover, our code has no errors which are always undetected or miscorrected.

The encoder and the decoder for the proposed codes have been modelled in Verilog and synthesized in Cadence Encounter RTL Compiler with the Nangate 45nm Opencell library version v2009_07. The designs were placed and routed using Cadence Encounter. The latencies, the area overhead and the power consumptions of the encoders and the decoders were estimated using Concurrent Current Source (CCS) model under typical operation condition assuming a supply voltage of 1.1V and a temperature of 25 Celsius degree. The synthesis results for the encoder and decoder are shown in Table IV and Table V respectively. Those results were all obtained based on same simulation condition.

TABLE IV
SYNTHESIS RESULTS FOR ENCODERS

Architectures	Latency (ns)	Area (um^2)	Power (mW)
Extended Hamming [I]	0.290	282.2	0.2898
Extended Vasil'ev [I]	0.367	296.1	0.2916
Extended Phelps [I]	0.429	383.0	0.4728
Proposed code (smaller latency)	1.000	2246.6	3.444
Proposed code (lower power)	1.860	1573.9	1.044

[I] Data for the robust codes and the extended Hamming code are obtained from [22]

From Table IV, we can see that comparing to the extended Phelps code, the extended AMC code with distance four requires at least 161% increase in the latency. When optimizing the area and power, the latency increases to 433% of Extended Phelps Code encoder. The encoder of the extended proposed code also requires at least 310% more area and 121% more power than the encoder of the Extended Phelps code. This is the cost required to provide for the strong security.

We implemented the decoders for the proposed SEC-DED AMC codes with three profiles. One emphasizes the overall latency, another forces on the clock speed, and the third one requires the smallest area as well as power. To achieve the fastest clock speed, the decoder is pipelined. From Table V, we see that the smallest clock cycle of the decoder of our code is 1.096 ns, which is 64% more than the extended Phelps code, while 130% more area and 370% more power are used. On the other hand, sacrificing the clock speed, the decoder requires only 58% more area and 30% more power than the extended Phelps code. With 80% more area and 120% more power, we note that we can achieve the smallest overall latency, which is 1.971 ns.

From the results presented in this section one can see that the proposed AMC codes provide for better security (see Table III) but require larger overhead in latency, area and power (see Table III and Table II) than the known SEC codes (Hamming, Vasil'ev and Phelps).

C. Case studies for the proposed code

Figure 4 shows the transmission rate for different numbers of information bits for the extended Hamming code which is the same as the extended robust codes from [22] and the proposed SEC-DED

 ISBN: 978-1-61208-301-8

TABLE V
SYNTHESIS RESULTS FOR DECODERS

Architectures	Latency (ns)	Area (um^2)	Power (mW)
Extended Hamming [I]	0.538	620.3	0.7119
Extended Vasil'ev [I]	0.652	763.2	0.8340
Extended Phelps [I]	0.670	1799.8	1.774
Proposed code (smaller latency)	1.971	3272.9	4.056
Proposed code (faster clock)	1.096×2[II]	4215.6	8.49
Proposed code (lower power)	2.000×2[II]	2846.7	2.312
Double error correction for Proposed Code	0.917	11669.7	4.402

[I] Data for the robust codes and the extended Hamming code are obtained from [22]
[II] "×2" means there are two pipeline stages and the number in front indicates the clock speed.

AMC codes with distance 4 with three different parameter sets. One code uses $m = 7$ random bits, and (11,7,3) Hamming code. For another proposed code, we use $m = 17$ random bits and (22,17,3) Hamming code. The third proposed code use $m = 19$ random bits and (24,19,3) Hamming code. (We need that $2^m - 1$ will be a prime.) The degree b of nonlinear function of the proposed codes varies with the number of information bits.

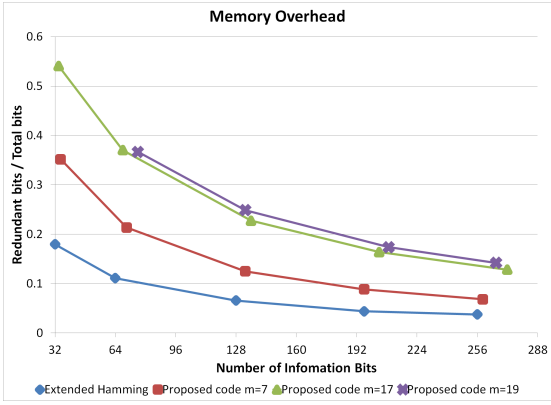


Fig. 4. Memory Overhead

The proposed codes require more redundant bits than the extended Hamming code, however as the number of information bits increases, the transmission rate of the proposed codes dramatically increases, since the number of redundant bits for the proposed codes depends on only m and does not depend on the number of information bits $k = bm$ as b increases. As k goes to infinity, the transmission rate of the proposed codes approaches one.

In order to decrease the probability for the missing, we can increase the number of random bits m. Moreover, in order to obtain a reasonable transmission rate, the degree b of the nonlinear function $f(y, x)$ should also be increased. Since $Q_{V_{AMC}} = (b + 1)(2^m - 2)^{-1}$, the denominator increases exponentially with m, while the numerator increase linearly with b.

In the remainder of this section we consider the problem of the optimal selection of parameters m and b for a given number k of information bits for the proposed codes.

With the same m, as b increases, additional multipliers are required for $f(y, x)$ and the complexity for the encoder and the decoder are increasing. Additionally, an increase in the length k of the information part will result in an increase of the probability of multiple-bit random errors, which diminishes utility of SEC-DED codes. Therefore when choosing the parameters of the code, we should balance the transmission rate and the encoder and decoder's hardware complexities, security and reliability demands.

We propose the codes balancing these secure and reliable demands, with parameters listed in Table VI.

TABLE VI
PARAMETERS FOR THE PROPOSED CODES

k	m	$m + r_H$	b	$Q_{V_{AMC}}$	Q_{mc}
68	17	22	4	$6(2^{17} - 2)^{-1}$	$30(2^{17} - 2)^{-1}$
136	17	22	8	$10(2^{17} - 2)^{-1}$	$90(2^{17} - 2)^{-1}$
204	17	22	12	$14(2^{17} - 2)^{-1}$	$182(2^{17} - 2)^{-1}$
272	17	22	16	$18(2^{17} - 2)^{-1}$	$306(2^{17} - 2)^{-1}$
76	19	24	4	$6(2^{19} - 2)^{-1}$	$30(2^{19} - 2)^{-1}$
133	19	24	7	$8(2^{19} - 2)^{-1}$	$56(2^{19} - 2)^{-1}$
209	19	24	11	$12(2^{19} - 2)^{-1}$	$132(2^{19} - 2)^{-1}$
266	19	24	14	$16(2^{19} - 2)^{-1}$	$240(2^{19} - 2)^{-1}$

The hardware overhead for the encoders and the decoders for those SEC-DED codes are shown in Table VII and Table VIII respectively, under the same simulation condition as in Section IV-B. Note that the architectures here are not pipelined. (The pipelined version may use faster clock.) And we do not use the Horner scheme to implement the AMD encoding functions in order to achieve a smaller overall latency. We may also sacrifice the area and power to achieve a smaller latency.

TABLE VII
ENCODER OVERHEAD FOR THE PROPOSED CODES

Parameters of the codes	Latency (ns)	Area (um^2)	Power (mW)
m=17, b=4	2.623	5495	1.509
m=17, b=8	4.375	11375.2	3.968
m=17, b=12	4.403	16709.6	7.145
m=17, b=16	4.799	23578.8	11.23
m=19, b=4	3.181	7013.6	2.522
m=19, b=7	4.914	13788.1	6.052
m=19, b=11	4.945	20253.5	10.898
m=19, b=14	5.419	28587.9	17.128

Additionally, we note that the data output from the memory can be directly forwarded to other parts of the system, before the decoder generates its outputs. When errors are detected or corrected by the decoder, the processor can be stalled and the data can be re-fetched from the decoder. In this case, when no errors occur, the decoder does not affect the performance of the system in term of the latency. We may take advantage of this when our code is used for a memory, when the random access is frequently required.

V. CONCLUSIONS

In this paper, we show a reliable and secure memory architecture based on robust algebraic manipulation correction (AMC) codes. We describe the constructions of robust AMC codes, estimate their parameters (error detection and/or correction probabilities, etc) and

 ISBN: 978-1-61208-301-8

TABLE VIII
DECODER OVERHEAD FOR THE PROPOSED CODES

Parameters of the codes	Latency (ns)	Area (um^2)	Power (mW)
m=17, b=4	5.421	9040	1.761
m=17, b=8	7.455	17376	4.411
m=17, b=12	7.594	26389	7.487
m=17, b=16	8.127	35446	11.25
m=19, b=4	6.051	10500	2.372
m=19, b=7	6.666	19609	5.072
m=19, b=11	7.565	30568	9.68
m=19, b=14	8.886	38363	12.18

present the robust error correction algorithm for these codes. For the presented codes any error for any user defined message can be detected with a probability exponentially converging to one. Any repeating error (error with a high laziness) can be corrected with a probability converging to one as P_R or the number of redundant bits r grows. The area, power consumption and the latency for the encoder and the syndrome computation circuit are studied. The described architecture is suitable for the protection of the most security/reliability critical part (which is usually a small portion of the system) of the memory in many applications, where the error model is unpredictable (e.g., memories in cryptographic devices that may suffer from fault injection attacks) or the multi-bit error rate is high or difficult to estimate (e.g., nano-scale memories).

ACKNOWLEDGEMENT

The work of the third author is sponsored by the NSF grant CNS 1012910.

REFERENCES

[1] T.R.Halfhil, "Z-ram shrinks embedded memory," Microprocessor Report, Tech. Rep., Oct 2005.

[2] S.K.Moore, "Masters of memory," *IEEE Spectrum*, vol. 44, no. 1, pp. 45–49, Jan 2007.

[3] A. H. Johnston, "Scaling and technology issues for soft error rates," ser. 4th Annual Research Conference on Reliability, 2000.

[4] A. Eto, M. Hidaka, Y. Okuyama, K. Kimura, and M. Hosono, "Impact of neutron flux on soft errors in mos memories," in *Electron Devices Meeting*, 1998.

[5] S. Satoh, Y. Tosaka, and S. A. Wender, "Geometric effect of multiple-bit soft errors induced by cosmic ray neutrons on drams," June 2000.

[6] G. M. Swift, "In-flight observations of multiple-bit upset in drams," *IEEE Trans. Nuclear Science*, vol. 47, 2001.

[7] G. Georgakos, P. Huber, M. Ostermayr, E. Amirante, and F. Ruckerbauer, "Investigation of increased multi-bit failure rate due to neutron induced seu in advanced embedded srams," in *Symposium on VLSI Circuits Digest of Technical Paper*, 2007.

[8] J. Maiz, S. Hareland, K. Zhang, and P. Armstrong, "Characterization of multi-bit soft error events in advanced srams," in *IEEE Int'l Electronic Device Meeting*, December 2003, pp. 519–522.

[9] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The sorcerers apprentice guide to fault attacks," 2002.

[10] S. Skorobogatov, "Optical fault masking attacks," *Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 23–29, 2010.

[11] Y. Emre and C. Chakrabarti, "Memory error compensation techniques for jpeg2000," in *Signal Processing Systems (SIPS), 2010 IEEE Workshop on*, 2010, pp. 36–41.

[12] R. T. Chien, "Memory error control: beyond parity," *Spectrum, IEEE*, vol. 10, no. 7, pp. 18–23, 1973.

[13] Y. Bentoutou, "Efficient memory error coding for space computer applications," in *Information and Communication Technologies, 2006. ICTTA '06. 2nd*, vol. 2, 2006, pp. 2347–2352.

[14] M. G. Karpovsky and A. Taubin, "New class of nonlinear systematic error detecting codes," *IEEE Transactions on Information Theory*, vol. 50, no. 8, pp. 1818–1820, 2004.

[15] Z. Wang, M. Karpovsky, and A. Joshi, "Reliable MLC NAND flash memories based on nonlinear t-error-correcting codes," in *Dependable Systems and Networks, IEEE/IFIP International Conference on*, 2010.

[16] Z. Wang and M. Karpovsky, "Algebraic manipulation detection codes and their applications for design of secure cryptographic devices," in *On-Line Testing Symposium (IOLTS), 2011 IEEE 17th International*, july 2011, pp. 234 –239.

[17] G. Shizun, W. Zhen, L. Pei, and K. Mark, "Secure memories resistant to both random errors and fault injection attacks using nonlinear error correction codes," in *Proc. Workshop on Hardware and Architectural Support for Security and Privacy, HASP 2013*, 2013.

[18] W. Zhen and M. Karpovsky, "Reliable and secure memories based on algebraic manipulation correction codes," in *On-Line Testing Symposium (IOLTS), 2012 IEEE 18th International*, 2012, pp. 146–149.

[19] R. Cramer, Y. Dodis, S. Fehr, C. Padr, and D. Wichs, "Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors," in *Advances in Cryptology C EUROCRYPT 2008*, ser. Lecture Notes in Computer Science, N. Smart, Ed. Springer Berlin / Heidelberg, 2008, vol. 4965, pp. 471–488.

[20] Z. Wang and M. Karpovsky, "Algebraic manipulation detection codes and their applications for design of secure cryptographic devices," in *IEEE 17th International On-Line Testing Symposium (IOLTS)*, 2011, pp. 234–239.

[21] J. L. Vasil'ev, "On nongroup close-packed codes," in *Probl.Kibernet.*, vol. 8, 1962, pp. 375–378.

[22] Z. Wang, M. Karpovsky, and K. Kulikowski, "Design of memories with concurrent error detection and correction by nonlinear SEC-DED codes," *Journal of Electronic Testing*, pp. 1–22, 2010.

 ISBN: 978-1-61208-301-8

Robustness of Security-Oriented Binary Codes Under Non-Uniform Distribution of Codewords

Igor Shumsky and Osnat Keren
Faculty of Engineering
Bar-Ilan University
Ramat-Gan, Israel 52900
Email: ig.shum@gmail.com, osnat.keren@biu.ac.il

Mark Karpovsky
Department of Electrical and Computer Engineering
Boston University
Boston, Massachusetts 02215
Email: markkar@bu.edu

***Abstract*—Robust and partially robust codes are used in cryptographic devices for detecting active side channel attacks on the hardware. The codes are usually designed for uniformly distributed codewords. In practice, however, there are codewords that are much more likely to appear than others. This paper addresses the question of how good are existing robust codes in this context. The worst case scenario is analyzed and a method that allows the designer to avoid this scenario with a relatively low cost is presented.**

***Index Terms*—Robust codes; security; undetected error probability; puncturing; fault analysis attacks; non-uniform distribution;**

I. INTRODUCTION

The security of cryptographic devices is threatened by fault injection attacks on the hardware. By injecting faults an adversary can obtain secret or private information that is stored in the device. Modern fault injection techniques allow an adversary to introduce faults at any physical point of the circuitry. A fault can flip bits, stuck a gate at a certain value, or change data on wires [2], [8], [10]. In turn, an attack can be mathematically modeled as an additive (i.e., symmetric) error that distorts the correct output of that circuit. Unlike random errors, i.e., errors caused by nature, an error induced by an adversary can be of any multiplicity.

Fault injection attacks can be detected with relatively high probability by security-oriented codes. It is convenient to classify fault injection attacks by their strength; In weak attacks the adversary *cannot* control which codeword will appear at the output of the circuitry, while in *strong attacks*, he can determine the outputs by choosing the inputs.A schematic architecture, which provides robustness against weak attacks is shown in Fig. 1; Its equivalent mathematical model is shown in Fig. 2.

Codes for detecting weak attacks, e.g., [1], [3]–[6], [11], are usually designed under the assumption that the codewords are equally likely to occur. However, when the source of the information is a computation channel, i.e., a combinatorial logic or a sequential machine, this assumption is almost always violated. Indeed, the distribution of vectors applied at run-time to the inputs of the combinational portion of a sequential machine is highly skewed due to the fact that some state transitions are more common than others and that some input combinations are invalid and hence cannot occur. The distribution of the outputs of arithmetic modules is also highly non-uniform. For example, it is more likely to have a '0' at the output of a multiplier than other values. In arithmetic modules and in sequential state machines, the probability of having a certain output can be easily computed. A judicious attacker can use this information to choose an error that is hardly (if ever) detected.

This paper addresses two questions: a) how good are the

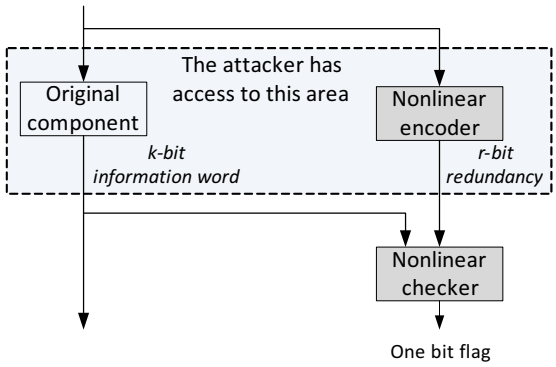


Fig. 1. A schematic architecture of a circuit component protected by a systematic security-oriented code. The shaded area is accessible to the attacker.

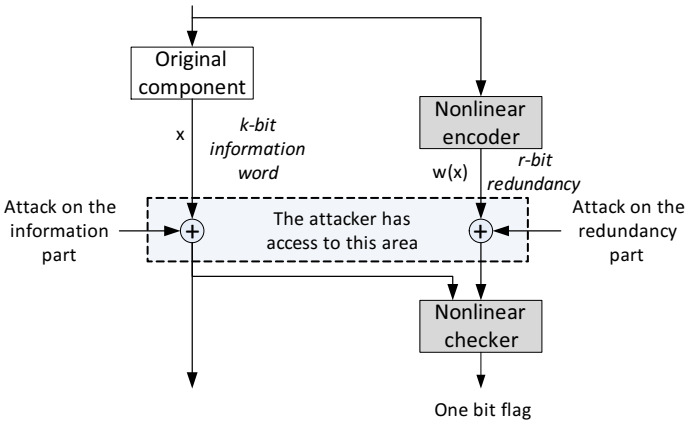


Fig. 2. Mathematical model of a circuit component protected by a systematic security-oriented code.

 ISBN: 978-1-61208-301-8

known robust codes, and in particular the Quadratic-Sum codes and codes derived from the cubic code, against an adversary that knows the distribution of the codewords, and b) is it possible to reduce the error masking probability of the code without adding more redundancy?

The remaining of the paper is organized as follows. Section II briefly describes security oriented codes and presents the Punctured-Cubic and the Quadratic-Sum codes. Section III analyzes the worst case scenario. Section IV introduces methods to avoid this scenario by mapping the set of most probable words to a predefined set. An upper bound on the error masking probability when using this mapping is also presented. Section V concludes the paper.

II. Preliminaries - Security oriented codes

A binary code $\mathcal{C}(n,k)$ is a subset of size 2^k of an n-dimensional binary vector space $\mathbb{F}_2^n$, ($\mathbb{F}_2 = GF(2)$). In conventional coding theory, codes are designed to provide reliability against *random errors*, i.e., errors of low multiplicity. The codes are therefore characterized by their rate (i.e., k/n), the minimal distance between the codewords, and the undetected (random) error probability. All these parameters are determined by the chosen code; They are indifferent to the encoding scheme.

In cases where the reliability of the system is the main concern, a *systematic code*, that is, a code in which the information word is embedded in the codeword in its original form, has an advantage over non-systematic codes since it simplifies the decoding procedure and usually has a lower implementation cost. However, in security oriented coding, the most important property of a code is its robustness, i.e its ability to provide immunity against weak attacks. As we show next, when some codewords are more probable to appear than others, the encoding (i.e., the mapping between an information word $m \in \mathbb{F}_2^k$ to a codeword $c \in \mathbb{F}_2^n$) plays a crucial role in determining the robustness of a code.

A. Definition of robustness

Let $\mathcal{C}$ be a code and denote by $p(c)$ is the probability that the codeword $c \in \mathcal{C}$ will be used. The robustness of $\mathcal{C}$ is measured in terms of its undetected error probability, which is also referred to as the *error masking probability*. The error masking probability is the probability, $Q(e)$, that a given error $e \in \mathbb{F}_2^n$ will map a codeword onto another codeword, i.e.,

$$Q(e) \equiv \sum_{c \in \mathcal{C}} p(c)\delta(c \oplus e) \tag{1}$$

where $\delta(z)$ is the characteristic function of the code, $\delta(z) = 1$ if $z \in \mathcal{C}$ and it equals 0 otherwise.

When the adversary induces an error e one of the following three scenarios may happen:

1) The error will always be detected ($Q(e) = 0$). The set of errors of this type is denoted by E_a.
2) The error will never be detected ($Q(e) = 1$). Errors that are never detected form a group. The group, denoted by K_d, is called the *Kernel* of the code.

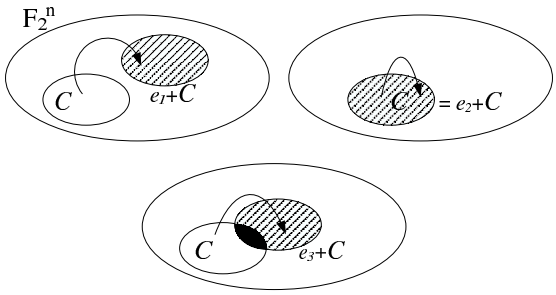


Fig. 3. The error $e_1 \in E_a$ is always detected since $\mathcal{C} \cap \{e_1 \oplus \mathcal{C}\} = \varnothing$. The error $e_2 \in K_d$ is never detected since $\mathcal{C} = \{e_2 \oplus \mathcal{C}\}$, and e_3 is detected with probability $Q(e_3) = |\mathcal{C} \cap \{e_3 \oplus \mathcal{C}\}|/|\mathcal{C}|$.

3) The error will be detected with probability $0 < 1 - Q(e) < 1$. That is, there exists at least one codeword that detects the error, and there exists at least one codewords that masks it.

The three scenarios are illustrated in Fig. 3.

Definition 1 (Robust and partially robust codes): Robust codes are codes for which the dimension of K_d equals zero, that is, no attack is masked. *Partially robust* codes are codes for which the dimension of K_d is greater than zero but less than k.

B. The error masking equation

Let $\mathcal{C}(n,k)$ be a binary systematic code of length $n = k + r$ and size 2^k. A codeword $c \in \mathcal{C}(n,k)$ has two parts: an information part denoted by x and a redundancy part w, which is a function of x. Each part can be referred to as an element of a finite field or as a vector over a finite field. For example, the information part x can be considered as a binary vector in k-dimensional space $\mathbb{F}_2^k$; It can be also referred to as an element of the finite field $\mathbb{F}_{2^k} = GF(2^k)$. For example, the expression Px^3 where P is a $r \times k$ matrix, has to be read as: refer to x as an element in $\mathbb{F}_{2^k}$ and compute x^3, then refer to the result as a vector in $\mathbb{F}_2^k$ and multiply it by the matrix P, the outcome of this operation is an element in $\mathbb{F}_{2^r}$.

Let $c = (x, w) \in \mathcal{C}$ be a codeword, where $w = w(x)$. Let $e = (e_x, e_w)$ be a nonzero error vector, $e_x \in \mathbb{F}_{2^k}, e_w \in \mathbb{F}_{2^r}$. An error is undetected (masked) by the codeword c if $c \oplus e \in \mathcal{C}$. Equivalently, e is masked by c if

$$w(x \oplus e_x) = w(x) \oplus e_w. \tag{2}$$

Equation (2) is called the *error masking equation* for systematic codes. The number of solutions (x's) to (2) and the probability of each determine $Q(e)$. Namely, let $X(e)$ be the set of x's that satisfy this equation,

$$X(e) = \{x | c(x) \oplus e \in \mathcal{C}\}. \tag{3}$$

Then,

$$Q(e) = \sum_{x \in X(e)} p(x), \tag{4}$$

where $p(x)$ is the probability of the codeword $c = (x, w)$, i.e., $p(x) = p(c)$.

 ISBN: 978-1-61208-301-8

The error masking probabilities of $\mathcal{C}$ and error masking probabilities of a coset of $\mathcal{C}$ are identical. Therefore, without loss of generality, we assume that $\mathbf{0} = (0,0) \in \mathcal{C}$. Consequently,

Property 1: If $\mathbf{0} \in X(e)$, then $e \in \mathcal{C}$.

The error masking probability for uniformly distributed codewords is lower bounded by [6],

$$Q(e) \geq \max(2/2^k, 2^k/2^n). \tag{5}$$

Codes that achieve this bound are called *optimum codes*.

C. The Punctured-Cubic code and the Quadratic-Sum code

In this paper, we analyze two robust codes, the Punctured-Cubic (PC) code derived from the cubic (x, x^3) code by deleting some redundancy bits, and the Quadratic-Sum (QS) code. Both codes are robust *systematic* codes of rate higher than one-half [1], [4], [7]. Moreover, both codes are optimum or close to optimum.

Construction 1 (Punctured-Cubic code [1]):
Let P be a binary $r \times k$ matrix of rank $r \leq k$. The code

$$\mathcal{C} = \{(x, w) : x \in \mathbb{F}_{2^k}, w = Px^3 \in \mathbb{F}_{2^r}\} \tag{6}$$

is called a Punctured Cubic $\mathcal{C}(k+r, k)$ code.

The error masking equation of the PC code is

$$P(x \oplus e_x)^3 = Px^3 \oplus e_w. \tag{7}$$

Construction 2 (Quadratic-Sum code [4]):
Let $k = 2sr$ and $x = (x_1, x_2, \cdots x_{2s})$, where $x_i \in \mathbb{F}_{2^r}$ for $1 \leq i \leq 2s$. The code

$$\mathcal{C} = \{(x, w) : x \in \mathbb{F}_{2^k}, w = x_1x_2 \oplus \cdots \oplus x_{2s-1}x_{2s} \in \mathbb{F}_{2^r}\} \tag{8}$$

is called a Quadratic-Sum $\mathcal{C}(k+r, k)$ code.

The error masking equation for the QS code is

$$\sum_{i=1}^{s}(x_{2i-1} \oplus e_{x,2i-1})(x_{2i} \oplus e_{x,2i}) = \sum_{i=1}^{s} x_{2i-1}x_{2i} \oplus e_w. \tag{9}$$

D. The robustness of the PC and QS codes under uniform distribution

If the codewords are uniformly distributed, then each codeword may appear on the output with probability of $1/|\mathcal{C}|$. The worst case error masking probability under uniform distribution of the codewords is denoted by Q_{mc}. The subscript mc stands for maximal correlation, since in this case

$$Q(e) = \frac{R(e)}{R(0)}, \tag{10}$$

and,

$$Q_{mc} = \frac{\max_{e \neq 0} R(e)}{R(0)}, \tag{11}$$

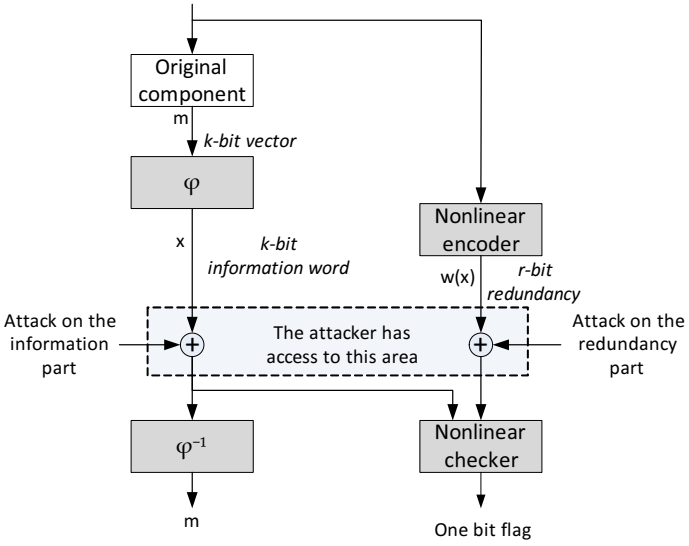


Fig. 4. A mathematical model of a computation channel protected by a one-to-one mapping φ followed by a systematic error detecting code $\mathcal{C}$.

where R is the autocorrelation function of δ, that is,

$$R(e) = \sum_{z \in \mathbb{F}_2^n} \delta(z)\delta(z \oplus e). \tag{12}$$

The error masking probabilities of the PC and QS codes are the following:

Theorem 1 ([7]): Let C be a PC code defined by a binary $r \times k$ matrix P of rank $r > 1$, Then the kernel of the code is of dimension 0. For odd values of k, $Q_{mc} = 2^{-r+1}$. For even values of k, there exist P matrices for which $Q_{mc} = 2^{-r}$.

Theorem 2 ([4]): Let C be a QS code. Then the kernel of the code is of dimension 0. For $k = 2sr$, the error masking probability is $Q_{mc} = 2^{-r}$.

III. The worst case scenario

Consider a computation channel that produces each cycle an output vector $m \in \mathbb{F}_2^k$. Let φ be a one to one mapping between m and an information word x, i.e., $x = \varphi(m)$. To provide immunity, each cycle a codeword $c = (x, w(x))$ is generated from the information word x (as shown in Fig. 4). The probability that a codeword $c(x) = c(\varphi(m))$ is used equals to the probability that the output m is produced, that is,

$$p(c) = p(x) = p(m). \tag{13}$$

Since for a given code, $X(e)$ is fixed, and

$$Q(e) = \sum_{x \in X(e)} p(x) = \sum_{m,\ \varphi(m) \in X(e)} p(m), \tag{14}$$

the error masking probability under non-uniform distribution of the outputs depends solely on φ.

The following lemma provides a lower bound on the error masking probability when the worst φ is used. In the next

section we show that if one uses a φ that maps the most probable vectors m to a *predefined* set S, s/he can reduce the error masking probabilities.

Without loss of generality assume that

$$1 \geq p(m_1) \geq p(m_2) \geq \cdots \geq p(m_{2^k}) \geq 0 \tag{15}$$

and

$$\sum_{i=1}^{2^k} p(m_i) = 1. \tag{16}$$

Consider the mapping $x_i = m_i$. For this mapping we have,

$$1 \geq p(x_1) \geq p(x_2) \geq \cdots \geq p(x_{2^k}) \geq 0. \tag{17}$$

Denote by $P(S)$ the accumulated probability $\sum_{x_i \in S} p(x_i)$ and assume that there is a set $S \subseteq \mathbb{F}_2^k$ for which $P(C \setminus S)$ is *negligible*. In the worst case scenario there exists an error e such that either $S \subseteq X(e)$ or $X(e) \subset S$. Namely,

Lemma 1: The worst case error masking probability, Q_{wc}, is lower bounded by

$$Q_{wc} \geq \begin{cases} P(S) & |S| \leq Q_{mc}2^k \\ \frac{Q_{mc}2^k}{|S|}P(S) & otherwise \end{cases} \tag{18}$$

Example 1: Let $k = 3$ and $r = 1$. The eight codewords of the corresponding PC code (represented by their integer values) are

$$(0,0),(1,0),(2,0),(3,1),(4,1),(5,1),(6,1),(7,0). \tag{19}$$

Table I shows the $X(e)$ of each error vector.

TABLE I
THE ERROR VECTORS AND THEIR MASKING CODEWORDS

e	$\lvert X(e)\rvert$	$X(e)$
(0,0)	8	all x's
(0,1)	0	-
(1,0)	4	0,1,4,5
(1,1)	4	2,3,6,7
(2,0)	4	0,2,4,6
(2,1)	4	1,3,5,7
(3,0)	4	1,2,5,6
(3,1)	4	0,3,4,7
(4,0)	0	-
(4,1)	8	all x's
(5,0)	4	2,3,6,7
(5,1)	4	0,1,4,5
(6,0)	4	1,3,5,7
(6,1)	4	0,2,4,6
(7,0)	4	0,3,4,7
(7,1)	4	1,2,5,6

The rows of the table are written in pairs. In each pair, one error vector is a codeword and the second is a non-codeword. By Prop. 1, an error vector whose $X(e)$ contains the all-zero word, is a codeword. It is clear from the table that the code is *partially robust* since the non-zero error $(4,1)$ is masked by all codewords. However, all the remaining error vectors are either always detected or they are masked by half of the codewords. Therefore, for uniformly distributed codewords, $Q_{mc}(e) = 0.5$. Although this paper deals with robust codes, to simplify the presentation, we assume that the adversary cannot induce the error $(4,1)$. This assumption allows us to use the $\mathcal{C}(4,3)$ partially robust PC code.

Assume now that the m's are not uniformly distributed,

$$p(m) = \begin{cases} (1-\epsilon)/5 & m \in \{2,3,4,6,7\} \\ \epsilon/3 & otherwise \end{cases} \tag{20}$$

If no mapping is used (i.e., $x_i = m_i$), then a judicious attacker would apply the error $(5,0)$ whose corresponding error masking probability is the maximal, $Q((5,0)) = \frac{4}{5}(1-\epsilon)$. However, a Gray code can reduce the worst case error masking probability. A Gray code maps $m = (m_{k-1}, \ldots m_0)$ to $x = (x_{k-1}, \ldots x_0)$ as follows: $x_i = m_{i+1} \oplus m_i$ for $i = 0, \ldots k-1$ where $m_k = 0$. In our case, the highly probable m's are mapped to the set $S = \{2,3,4,5,6\}$, and the worst case error masking probability becomes $\frac{3}{5}(1-\epsilon)$. As we show next, no better mapping can be found.

IV. CONSTRUCTIVE UPPER BOUNDS ON THE ERROR MASKING PROBABILITY

For uniformly distributed codewords, the error masking probability of the PC and the QS codes is upper bounded by Q_{mc}. Therefore, any error vector is masked by at most $2^k Q_{mc}$ codewords. Consequently, if the size of S, is greater than $2^k Q_{mc}$, then any error will be detected with probability of at least

$$1 - \frac{2^k Q_{mc}}{|S|}P(S) > 0. \tag{21}$$

Obviously, if the size of S is smaller than that, the probability that the error will be masked increases. In what follows we discuss the case where

$$|S| \leq \min_{e \neq 0} |X(e)|, \tag{22}$$

and present mappings for which any nonzero error will never be masked.

A. Sufficient conditions for $Q < 1$

In cases where $|S| = 2$, no mapping can help; An adversary who knows the two most probable outputs, say m_1 and m_2, and the mapping φ may choose an error

$$e = c(\varphi(m_1)) \oplus c(\varphi(m_2)), \tag{23}$$

for which $Q(e) \geq P(S) = 1 - \epsilon$.

The following theorem suggests a lower bound on the size of S for which there exists a mapping that can reduce $Q(e)$.

Theorem 3: Let $\mathcal{C}$ be a PC or a QS code. Then, there exists at least one set S of size s,

$$\frac{k+1}{-log_2(Q_{mc})} + 1 \leq s \leq \min(2^k Q_{mc}, 2^{k-2}), \tag{24}$$

such that $S \setminus X(e) \neq \phi$ for all non-zero e.

 ISBN: 978-1-61208-301-8

Example 2: Let $k = 16$ and $r = 4$. Assume that twenty vectors (out of the 2^{16}) may appear with probability $1 - \epsilon$ at the output of the device to be protected. Since there exists an error for which $\min(|X(e)|) = 2^{12}$, and $20 << 2^{12}$, in the worst case scenario the error will not be noticed. For a PC code we have,

$$\frac{16+1}{4-1} + 1 \leq |S| = 20 \leq min(2^{13}, 2^{16-2}). \quad (25)$$

therefore, by Theorem 3, there exist a subset S of twenty vectors such that any error is detected with probability of at least $\frac{1-\epsilon}{20}$.

Although Th. 3 states that it is possible to find a set that can detect any error, it does not provide an efficient way to do so. In the following sections we introduce two mappings, i.e., two sets, for which any non-zero error can be detected.

B. Generalized Hamming ball mapping

We define a generalized Hamming ball as follows:

Definition 2: Let $V = \{v_i\}_{i=1}^{u} \subset \mathbb{F}_2^k$ be an arbitrary set of u, $u \leq k$, linearly independent vectors. A generalized Hamming ball $B_{(u,w)} \subseteq \mathbb{F}_2^k$ is a set (or a coset of a set) that consists of the vectors

$$\left\{ \sum_{i=1}^{u} a_i v_i \mid a = (a_u, \dots a_1) \in \mathbb{F}_2^u, \ wt_H(a) \leq w \right\} \quad (26)$$

where $wt_H(a)$ stands for the Hamming weight of a.

Theorem 4: Let $\mathcal{C}$ be a PC or a QS code. Let $S \subseteq B_{(u,w)}$ where $u \geq k + log_2(Q_{mc}) + 1$ and w is the smallest integer such that $\sum_{j=0}^{w} \binom{u}{j} \geq |S|$. Then, the code $\mathcal{C}$ can detect all the nonzero errors with probability greater or equal to

$$\frac{|S| - \sum_{j=0}^{w} \binom{k+log_2(Q_{mc})}{j}}{|S|}. \quad (27)$$

The proof of Theorem 4 follows directly from the fact that the PC code and the QS code have the following property:

Theorem 5: Let $\mathcal{C}$ be a PC or a QS code. Then, $X(e)$ is a subspace iff e belongs to $\mathcal{C}$ and a coset otherwise.

Corollary 1: The minimal size of a set that can detect any non-zero error e with $Q(e) > 0$ is greater than two and less or equal to $k + log_2(Q_{mc}) + 2$.

Example 3: Let $k = 16$ and $r = 4$. Assume that 650 output vectors (out of the 2^{16} possible combinations) occur with probability of $1 - \epsilon$. Since for a PC code,

$$|X(e)| \geq 2^{k-r} = 2^{12} > 650, \quad (28)$$

in the worst case scenario, there may be an error that will be masked with probability greater than $1 - \epsilon$. However, for $w = 3$ and $u = 16$ we have

$$|B_{(16,3)}| = \sum_{j=0}^{3} \binom{16}{j} = 697. \quad (29)$$

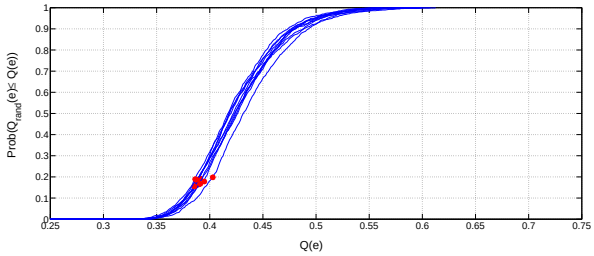


Fig. 5. The probability that a random mapping for a QS code with $k = 6$ and $r = 3$ will provide a maximal error masking probability smaller than $Q(e)$ for ten probability distributions having $|S| = 7$. The red dots denote $Q(e)$ achieved by Const. 3

Therefore, by mapping these 650 m's to a set $S \subset B_{(16,3)}$ that consists of binary vectors of Hamming weight less or equal to three, one can reduce the error masking probability to

$$Q(e) \leq \frac{|S \cap X(e)|}{|S|} \leq \frac{\sum_{j=0}^{3} \binom{k-r+1}{j}}{|S|} = 0.58. \quad (30)$$

Note that if $|S| > k$, then the size of $S \cap X(e)$ decreases as the number of linearly independent vectors u increases. Moreover, as $|S|$ increases, the required w increases. The following construction, presented in [9], is not optimal, however, since it uses binary vectors of weight one, it is simple to implement.

Construction 3 ([9]): Let $p(m_1) \geq p(m_2) \geq \cdots \geq p(m_{2^k})$ Assign to each m_i a binary vector x_i such that the Hamming weight of x_i is smaller or equal to the Hamming weight of x_j for all $i < j$.

Note that if, for example, $k = 6, r = 3$ and $|S| = 7$ with the probability distribution

$$p(m) = \begin{cases} \frac{1-\epsilon}{|S|} & 0 \leq m \leq 6 \\ \frac{\epsilon}{2^k - |S|} & m > 6 \end{cases}, \quad (31)$$

then there exist other mappings, which achieve smaller $Q(e)$'s than [9]:

$$\begin{array}{lll} S = \{0,1,2,4,8,16,32\} \text{ [9]} & \rightarrow & Q(e) \leq 0.5714, \\ S = \{0,10,21,27,50,55,62\} & \rightarrow & Q(e) \leq 0.4286. \end{array} \quad (32)$$

Although the mapping in [9] is not optimal, it is much better than a random mapping. Fig. 5 shows, for ten different probability distributions having $|S| = 7$, the probability that a random mapping will provide a maximal error masking probability smaller than $Q(e)$. The red dots in the figure denote the error masking probability $Q(e)$ achieved by the suggested mapping. Refer only to the x-coordinate of the dots. The y-coordinate has no meaning, the star is placed on the graph just for convenience. On average this mapping has $Q(e) = 0.39$. The probability that a random mapping will provide error masking probability smaller than that is 0.18.

 ISBN: 978-1-61208-301-8

C. Robust-code based mapping

The following theorem states that if the elements of S are the codewords of a robust code, then a nonzero error is never masked.

Theorem 6: Let $\mathcal{C}$ be a PC or a QS code of dimension k, r redundancy bits, and error masking probability Q_{mc}. Let S be a robust code of length $\hat{n} = k$, dimension $\hat{k} = u$ and error masking probability $\hat{Q}_{mc}$. Then, the error masking probability of $\mathcal{C}$ is

$$Q(e) \leq \sqrt{2\hat{Q}_{mc}Q_{mc}2^{k-u}}. \quad (33)$$

Corollary 2: Let $\mathcal{C}$ be a PC or a QS code of dimension k, r redundancy bits, and error masking probability Q_{mc}. Let S be a subset of a robust code of length k, dimension $u = \lceil log_2(|S|) \rceil$ and error masking probability $\hat{Q}_{mc}$. Then,

$$Q(e) \leq \frac{\sqrt{Q_{mc}2^k(\hat{Q}_{mc}2^u + 1)}}{|S|}. \quad (34)$$

Corollary 3: Let $\mathcal{C}$ be a PC or a QS code of dimension k, r redundancy bits. Let S be a QS code of dimension u and $k-u$ redundancy bits. Then we have,

$$Q(e) \leq \sqrt{2 \cdot 2^{-(k-u)} \cdot 2^{-r+1} \cdot 2^{k-u}} \leq 2^{\frac{-r+2}{2}}. \quad (35)$$

Example 4: As before, let $k = 16$, $r = 4$ and assume that 400 output vectors may appear with probability $1 - \epsilon$. Here again, in the worst case scenario we have $Q(e) \geq 1-\epsilon$. Define S to be a subset of a $\mathcal{C}(\hat{n} = k = 16, \hat{k} = u = 9, \hat{r} = k-u = 5)$ PC code with $\hat{Q}_{mc} = 2^{-5+1}$. Then,

$$Q(e) \leq \sqrt{2 \cdot 2^{-4} \cdot 2^{-3} \cdot 2^5} = 0.707. \quad (36)$$

Note that in this case, the construction suggested in Th. 4 provides $Q(e) \leq \frac{378}{400} = 0.945$.

The following example shows the relation between the three upper bounds on the error masking probability when a mapping is applied.

Example 5 (Concluding example): Consider a PC code of dimension $k = 16$ and $r = 4$ redundancy bits. Assume that the $|S|$ most probable words are mapped to a set S, and that

$$p(x) = \begin{cases} \frac{1-\epsilon}{|S|} & x \in S \\ \frac{\epsilon}{2^k - |S|} & x \notin S \end{cases}. \quad (37)$$

The efficiency of the mapping, i.e., the error masking probabilities that can be achieved by using the suggested mappings, is shown in Fig. 6.

The X-axis is the size of S and the Y-axis is $\max_{e \neq 0}(Q(e))$. The black line represents a lower bound on worst case scenario (Lemma 1). The other lines represent upper bounds on $Q(e)$. The red line is the bound presented in Theorem 4, the blue line is the bound presented in Theorem 3, and the green line is the bound in Corollary 2.

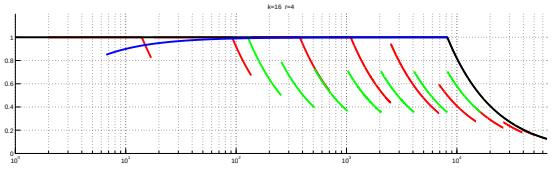


Fig. 6. Error masking probability of punctured cubic code with $k = 16$ and $r = 4$ as a function of $|S|$.

V. Conclusions

The Punctured-Cubic code and the Quadratic-Sum code are systematic robust codes designed for uniformly distributed codewords. The codes can detect any error with non-zero probability regardless its multiplicity. In cases where the codewords are not equally likely to appear, the performance of the codes degrades significantly and the robustness may vanish. The paper addresses this problem. It is shown that by mapping the most probable data patterns to a predefined set before the encoding, it is possible to significantly reduce the error masking probability and maintain the robustness of the codes.

Acknowledgment

The work of the first two authors was supported by the Israel Science Foundation (ISF) grant No. 1200/12. The work of the third author was supported by the NSF Grant CNS 1012910

References

[1] N. Admaty, S. Litsyn, and O. Keren, "Punctuating, Expurgating and Expanding the q-ary BCH Based Robust Codes", The 27-th IEEE Convention of Electrical and Electronics Engineers in Israel, 2012, pp.1–5.

[2] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The sorcerers apprentice guide to fault attacks", Vol.94, No.2, 2006, pp.370–382.

[3] S. Engelberg and O. Keren, "A Comment on the Karpovsky-Taubin Code," IEEE Trans. Info. Theory, Vol. 57, No. 12, 2011, pp. 8007–8010.

[4] M.G.Karpovsky, K. Kulikowski, and Z. Wang, "Robust Error Detection in Communication and Computation Channels" Keynote paper in the Int. Workshop on Spectral Techniques, 2007.

[5] M. G. Karpovsky and P. Nagvajara, "Optimal Codes for the Minimax Criterion on Error Detection," IEEE Trans. on Information Theory, Vol. 35, No. 6, November 1989, pp. 1299–1305.

[6] M.G.Karpovsky and A. Taubin, "A New Class of Nonlinear Systematic Error Detecting Codes," IEEE Trans. Info. Theory, Vol 50, No.8, 2004, pp.1818–1820.

[7] Y. Neumeier and O. Keren, "Punctured Karpovsky-Taubin Binary Robust Error Detecting Codes for Cryptographic Devices," IEEE International On-Line Testing Symposium, March 2012, pp.156–161.

[8] S. P. Skorobogatov, "Semi-Invasive Attacks - a New Approach to Hardware Security Analysis" Technical Report, University of Cambridge. Number 630.

[9] I. Shumsky and O. Keren, "Security-Oriented State Assignment", TRUDEVICE, 1'st Workshop on trustworthy manufacturing and utilization of secure devices, 2013.

[10] I. M.R. Verbauwhede(Ed.), Secure Integrated Circuits and Systems, Springer, 2010.

[11] Z. Wang, M. G. Karpovsky, and K. Kulikowski, "Design of Memories with Concurrent Error Detection and Correction by Non-Linear SEC-DED Codes" Journal of Electronic Testing, Vol. 26, No. 5, Oct 2010, pp.559–580.

 ISBN: 978-1-61208-301-8

Improving the Efficiency of Gossiping

Christian Esposito
Institute for High Performance Computing and Networking (ICAR), Napoli, 80131 - Italy
christian.esposito@na.icar.cnr.it

Roberto Beraldi
Dept. of Computer and Systems Engineering University of Roma La Sapienza, Roma, 00185 - Italy
beraldi@dis.uniroma.it

Marco Platania
Dept. of Computer Science Johns Hopkins University, Baltimore, MD 21218 - USA
platania@cs.jhu.edu

***Abstract*—In the current industrial practice, there is an increasing demand for an effective communication infrastructure to interconnect several heterogeneous systems. The effectiveness of the adopted communication infrastructure is defined in terms of the provided reliability and timeliness despite manifestations of failures within the network. Current approaches do not address both these aspects [7], but one aspect is generally assured at the expense of the other. Our driving idea is to take the best solution to achieve reliability and to improve its achievable performance so as to guarantee timely deliveries. Specifically, we propose to introduce determinism within the gossiping approach and to combine push and pull schemes. We have experimentally assessed such solutions through simulations, so as to find which is the one that best achieves both reliability and timeliness.**

***Index Terms*—Publish/Subscribe Middleware; Reliable Event Notification; Gossiping**

I. Introduction

Current software systems are characterized by a progressive increase in their scale and by a high demand for cooperation among their constituents, so as to adopt a "system of systems" perspective. The current literature is rich in practical examples of this novel generation of large-scale systems. The most demanding challenge that they have to face is to be able to interconnect several heterogeneous components in a large-scale setting, where the system is not limited within rack cabinets but spans over different distinct geographical sites and administrative domains. Therefore, the key component in these systems is the middleware solution adopted to enable the communication among their distributed components. Due to the required cooperation, the communication patterns that we can infer by analyzing the data flows within these systems do not consist of a naive request-respond communication style, where there is a client invoking a service upon a server and waiting for the result of such invocation. In most cases, we find a publish/subscribe communication style [1], where one, or even more, publisher asynchronously provides data to a set of interested subscribers.

The applications running on top of these systems present very strict non-functional requirements, *e.g.*, applications running on top of Grid or Cloud Computing typically can be assumed as business-critical, while the ones running on top of LCCI are mission-critical. Such application-level requirements are translated down to the middleware level in a set of proper constraints on the offered Quality-of-Service (QoS) in terms of reliability and timeliness. When considering large-scale systems, it is not practical to deploy a dedicated and proprietary network among interacting components. Therefore, communication can be only realized by means of the available IP-based network infrastructures, such as the Internet. However, such networks are typically affected by routing phenomena and failures that compromise the correctness of the packet delivery [2], which have a negative impact on the QoS experienced by users. Therefore, the adopted middleware has to be equipped with proper reliability enforcement methods to face such failures. In addition, the time to deliver information matters, since a message delivered too late can be useless or even dangerous for the system. Current approaches are not able to provide both reliability and timeliness, since recovering data dropped by the network typically implies some performance fluctuations. For a concrete example, a well-known approach called Gossiping [3] provides a high degree of reliability, while exhibiting a considerable worsening in performance. On the contrary, a distributed coding approach, called Network Coding [4], can present a more stable and predictable latency, while offering lower reliability guarantees.

Our driving idea is to select the best available solution to provide a high degree of reliability and to propose suitable methods to reduce its performance deficiency, so as to meet both reliability and timeliness. In a previous preliminary and theoretical work [5], we have shown that it is possible to combine the two mentioned approaches to obtain the best from both, *i.e.*, high reliability with no severe performance penalty. Such an intuition has been further proved by an experimental campaign in [6]. This previous work presents a significant flaw: the random nature of gossiping causes a high number of un-needed messages being exchanged among the nodes. This implies both a considerable traffic load on the network, which can cause congestion phenomena, and a non-optimal recovery time of lost data by wasting gossip messages sent towards nodes that do not require them. Our solution is to limit such random behaviour by forcing the protocol to prefer gossiping only with the nodes requiring a recovery action.

This paper is structured as follows. Section II provides a background on the current literature and describes our approach in combining coding and gossiping by highlighting open issues that we left untreated in our previous works. Section III presents our solution to introduce determinism, and Section IV proves the quality of our approach by means of simulations run in OMNET++. Last, we conclude with Section V, where we present the lessons learned with this work and its possible future evolution.

 ISBN: 978-1-61208-301-8

II. BACKGROUND

As analyzed in details in [7], there are several approaches available in the current literature that can be used to provide reliability by tolerating data losses. On one side, we have approaches based on temporal redundancy, which use retransmissions to recover dropped messages. While, on the other side we have the one based on spatial redundancy, which send additional information along with application data. This applied redundancy is used to reconstruct the lost information without requiring any retransmission.

The best known example of a reactive approach for multicasting in large-scale systems is *Gossiping* [3]: a node stores a received message in a buffer with a size b, and forwards it for a limited number of times f_{in} (called fanin) to a randomly-selected set of nodes of size f_{out} (called fanout). Many variants of gossiping algorithms exist, which can be categorized as follows. In the *Push Approaches*, when a node receives a new message, it is immediately retransmitted to the randomly-selected nodes (*i.e.*, fanin is constant and implicitly equal to 1). On the other hand, in *Pull Approaches*, after a proper timeout t expires, nodes periodically send a list of recently-received messages. If a lost message is detected by comparing the received list with the history of messages received by the given node, then a retransmission of the lost message is requested. As we have experimentally shown in [6], these reactive solutions achieve a high degree of reliability. However, this gain is obtained at the cost of a reduced performance and timeliness, since latency exhibits severe fluctuations due to the high number of retransmissions needed to recover from consecutive losses, quite frequent in the current Internet [2].

A concrete example of a proactive approach is represented by *Network Coding* [4], which allows the generation of redundant information from the content of the application packets as a set of the linearly independent combinations. The benefit of proactive approaches is to reduce the performance worsening caused by the use of a reliability enforcement method. However, they offer a lower reliability degree since, if the redundancy degree is not properly set, data can be irremediably lost.

In [6], we have taken the gossip protocol, and enhanced it by introducing coding in two precise points of its algorithm: when data is transmitted disseminated among the members of a given group, and when data is gossiped to the randomly-selected nodes, *i.e.*, coding generates new packets both at the push delivery and at the pull-based retransmission. We have proved that a proper combination of gossip and coding is able to realize an optimal trade-off between reliability and timeliness with limited overhead worsening. Since the nodes receiving a gossip message are randomly chosen, there is a non-negligible probability that gossip messages may reach nodes that do not need them. To demonstrate this we have defined an utility function, namely U, and indicate a gossip message as useful if it is able to detect and recover a data loss. For a push gossip, U is the number of push messages that have allowed to recover a loss over the total number of received push messages per each node. For a pull gossip, U is the number of push calls that have triggered a retransmission over the total number of received pull calls per each node. In Fig. 1, we see that U has a low value, which further decreases when f_{out} is increased. This implies that a considerable part of the traffic generated by a gossip scheme is unneeded and optimizable, leaving space for improvements.

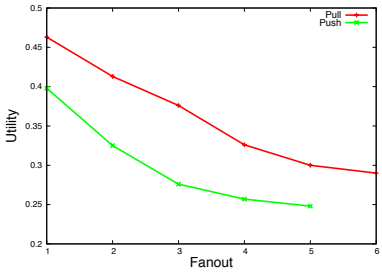


Fig. 1. Utility of the Gossip approaches from simulations in [6]

III. DETERMINISM IN GOSSIPING

We believe that the efficiency of gossip schemes in terms of performance and overhead can be improved by selecting nodes in a more deterministic manner. We refer to this approach as *Polarized Gossiping*, where node selection is not random, but based on a proper criterion. This solution is already present within the current literature, for example in [8] to reduce the overhead of gossiping in large-scale networks by allowing nodes to only gossip with other affine nodes, where node affinity is decided on the base of proximity or workload and information update frequency. However, our intention is different: we aim to speed up the recovery of lost data by preferring nodes with a high probability that the gossip message will be useful, so as to reduce the overall number of attempts to fully recover lost data.

We define as *optimal node selection* the approach to forward push messages only to those nodes for which such messages are useful. In the case of the push gossip, after the reception of a given event, a node will gossip with nodes still waiting for packets related to that event. Similarly, after the expiration of the timeout, a node will send a pull call message to the nodes that are still missing one, or even more, of the notifications contained in that pull call. Such a selection approach is not feasible in a large-scale system, since it requires complete knowledge of all the received notifications by each node within the system, which is not easily obtainable in most of the cases. Therefore, we propose proper heuristics for the node selection. Specifically, the driving idea is to assign a weight to each candidate to receive a gossip message, and those with the highest weights are selected for gossip. In particular, we describe two different methods for calculating such weights: in the first case, the weight is assigned based on the estimated overlay link status, expressed in terms of loss patterns; while in the second the weight assignment is based on the position of nodes within the tree. We conclude with several possible selection criteria based on the weights assigned to the nodes.

 ISBN: 978-1-61208-301-8

A. Labeling Nodes within the Multicast Tree

Our approaches for deterministic node selection require the knowledge of the topology within the multicast tree. This is typically a troublesome issue to be addressed in large-scale systems. However, we propose a method to handle it in a distributed and easy manner by assigning labels to each node. Specifically, each node has a label Φ of length l, where l is the level of that node in the tree. Each label is composed by l digits belonging to an m-ary alphabet, *i.e.*, a digit assumes a value in the range $[0; m-1]$. This label is assigned by the parent of the node: for each child in the tree, the parent adds a new progressive digit as the most significant one, while the root has $\Phi = null$. Labels are available to all the nodes thanks to a proper peer sampling service.

B. Weight Based on Loss Patterns

We assume that the links among nodes are not reliable, and exhibit a loss pattern characterized by the *Packet Loss Rate* (PLR), which is the probability of losing a packet. In particular, the adopted network model is the *Gilbert-Elliott* [9], one of the most-commonly applied in performance evaluation studies due to its analytical simplicity and the provided good results. PLR_x is continuously monitored by each node x over the overlay link that connects itself with its parent, *e.g.*, using the approach in [10], and disseminated towards the other nodes of the same group at the beginning, *e.g.*, after the node joins the group, and when there is a change in their value.

Such information on the link quality represents a valuable element for computing the weights for selecting nodes during a gossip round. Specifically, a given node x uses the PLR_x and the PLR estimated along the path between the root and its parent to compute the probability that it looses a packet along that path. We indicate this value with p_x, while $q_x = 1 - p_x$ is the probability of node x receiving a certain packet. Node x maintains information about all nodes along the path toward the root in a list called *ancestors* (easily computable in an iterative manner: when a node joins the tree, its parent passes to it its ancestor list). If the level of x in the tree is n, then the *ancestors* list contains n entries $z_0, z_1, ..., z_n$, where i indicates the level of a node in the tree and z_0 is the root and z_n is the node x itself. For each node z_i in this list, there is an associated packet loss rate PLR_{z_i} of the overlay link that connects z_i with its parent z_{i-1} (clearly, the packet loss rate associated to z_n is PLR_x, being $z_n = x$). Thus, the probability for node x of receiving a packet is the following one:

$$q_x = \prod_{i=1}^{n} (1 - PLR_{z_i}) \tag{1}$$

Such a value is continuously kept updated with the current value of PLR_{z_i}; if a node x communicates a change in its measured PLR, every node has to recompute its q_x, and relative weight, if the *ancestors* list contains the node x.

Based on these considerations, we define a formula that allows a node to assign weights to other nodes by considering the probability that those nodes have to lose packets. In the following, we describe how this formula is obtained for the previously introduced gossip strategies; then, we show how it can be easily modified for a given push/pull gossip strategy. The basic idea of this approach is that a node that receives a packet forwards it to the nodes that have a higher probability of having lost that packet. Thus, we denote with X the event *"node x received the packet"*, with Y the event *"node y received the packet"* and with $\overline{Y}$ the event *"node y did not receive the packet"*. Then, a node x that has to forward a received packet assigns to a node y a weight based on the following probability:

$$w_{xy} = Pr\{\overline{Y}|X\} \tag{2}$$

Such a value is influenced by the relative position of x and y in the tree. Let us consider x and y being on two completely different subtrees, *i.e.*, they share no common overlay links. Because we are assuming independent loss patterns, the probabilities of losing a packet at nodes x and y are totally uncorrelated. Thus, given two nodes x and y on two completely different branches of the tree, Equation 2 becomes:

$$Pr\{\overline{Y}|X\} = Pr\{\overline{Y}\} = p_y \tag{3}$$

If the node y is a predecessor of x, which is easily determined by comparing their labels, then obviously

$$Pr\{\overline{Y}|X\} = Pr\{\overline{Y}\} = 0 \tag{4}$$

Let us now consider nodes x and y in the same subtree. x has to find in the *ancestors* list the highest level ancestor h in common with y (it could be y itself if they are in the same branch of the tree. By indicating with n and l the level of y and h respectively, the probability of y having missed a packet knowing that x has received that packet depends on the probability of the packet having been lost during the path from h to y:

$$Pr\{\overline{Y}|X\} = \frac{Pr\{\overline{Y}, X\}}{Pr\{X\}} = 1 - \prod_{i=l+1}^{n} (1 - PLR_{z_i}) \tag{5}$$

It is easy to see that Equation 5 reduces to Equation 3 when the nodes x and y are on two completely different subtrees, the root of the tree being the highest level common ancestor (the root has $l = 0$). Indeed, in this case we have that $\prod_{i=l+1}^{n} PLR_i = p_x$. Thus, the final formula used by node x to assign a weight w_{xy} to node y includes the contributions 4 and 5:

$$w_{xy} = \begin{cases} 0 & \text{if y is predecessor of x} \\ 1 - \prod_{i=l+1}^{n} (1 - PLR_{z_i}) & \text{otherwise} \end{cases} \tag{6}$$

C. Weight Based on a Heuristic Approach

The previously described approach requires nodes to estimate the overlay link loss probability and to maintain additional information about the network conditions of their ancestors up to the root. In a large-scale system, it would generate a high network traffic, in addition to the scalability and consistency issues to maintain and update network status information. Thus, we also propose a heuristic approach to

 ISBN: 978-1-61208-301-8

assign weights that imposes no additional burden on system nodes since it is only based on the topological information extracted from the labels of the nodes. Specifically, this information takes into account the level of a node in the tree, referred to as the *horizontal cut*, and the subtree it belongs to, referred to as the *vertical cut*. strategies.

The assignment of w_{xy} is based on two considerations. The first one is the Horizontal cut: nodes in a lower level are more useful because they are closer to the source. In fact, nodes at the bottom of the tree are expected to experience a higher number of lost packets. The second one is the Vertical cut: nodes in different subtrees are more useful because, with a high probability, they experience a different loss pattern. The higher is the level of the root of the subtree that contains the two nodes, the more useful is their interaction for a gossip procedure. These considerations are orthogonal and can combined to assign a weight to all the nodes. Let us consider two nodes x and y, with labels Φ_x and Φ_y and label sizes s_{Φ_x} and s_{Φ_y} respectively (remember that they represent their level in the tree). In addition, let us define $\rho_{(x,y)}$ the length of the common suffix of Φ_x and Φ_y. The formula used by node x to assign a weight to node y based on the *Horizontal cut* is

$$w^h_{xy} = 1 - \frac{s_{\Phi_x}}{s_{\Phi_x} + s_{\Phi_y}}, \tag{7}$$

while the assignment based on the *Vertical cut* is as follows:

$$w^v_{xy} = 1 - \frac{\rho_{(x,y)}}{s_{\Phi_x}}. \tag{8}$$

The rationales behind these formulas are the following ones. With respect to the Horizontal cut, Equation 7 contains a formula in the form of $f(x) = 1 - c/(c+d)$, with c a constant. Given the two nodes x and y, with x that assigns a weight to y, $c = s_{\Phi_x}$ and $d = s_{\Phi_y}$, this formula is such that the value of $f(x)$ reduces when the variable x decreases. $w_h(x, y)$ assumes a lower value when both s_{Φ_x} is high and s_{Φ_y} is low, *i.e.*, when node x is further from the root and, on the contrary, node y is closer to the source of the information. In this way, node x has a benefit when it contacts node y for a recovery attempt. With respect to the Vertical cut, Equation 8 represents the fraction of overlay paths not in common between the two nodes x and y. The higher is the value, the higher is the level of their common ancestor. When $w_v(x, y) = 1$, the common ancestor is the root, *i.e.*, the nodes are on two completely different branches of the tree.

The total weight that node x assigns to node y is simply the product of the two previous equations:

$$w_{xy} = w^h_{xy} \cdot w^v_{xy}. \tag{9}$$

D. Polarized Gossip

Thanks to the introduced concepts of weighted selection, we can formulate two new gossiping schemes. *Weighted Deterministic Polarized Gossip* (WDPG) selects the nodes among the ones with the highest weights. *Weighted Random Polarized Gossip* (WRPG) specifies that the nodes with the highest weights have the highest probability of being selected. We have noticed that all the nodes with the highest weights are placed at the bottom of the tree, so nodes at the higher levels may less frequently receive gossip messages. This can reduce the loss-tolerant capability of the polarized gossiping; therefore, we have designed two other schemes for a better distribution of gossip messages. In *Polarized Gossip with Window* (PGW), all the nodes follow the same criterion of WDPG, while certain nodes at the bottom of the tree (whose percentage is an algorithm parameter called window) do not select the nodes with the highest weights, but with the weights closest to the highest one. With PGW, we allow certain nodes to send gossip messages to nodes in other parts of the tree and not only at the bottom. *Pull+Push Polarized Gossip* (3PG) combines push and pull gossip schemes by using a push-based WDPG joint-ly with a pull-based WDPG, where during the pull rounds nodes are selected by considering the lowest weights and not the highest ones. Such an approach has also a window, which indicates the nodes that are not able to commence a push round. This is meant to reduce the overhead that characterizes the push gossip.

IV. Simulation Study

The scope of this section is to present experimental results that (i) study the impact of different node selection criteria on the quality of gossiping, and (ii) investigate the effect on the polarized gossiping when coding is used. To achieve this aim, we implemented our solution by using the OMNET++ (www.omnetpp.org) simulator, and decided not to use any real wide-area networks, such as PlanetLab, due to the uncontrollable loss patterns that make the obtained results non reproducible [11]. The workload has been taken from the requirements of the SESAR project, representative of a real critical large-scale system. Specifically, the exchanged messages have a size of 23 KB, the publication rate is one message per second and the total number of nodes is 40 (*i.e.*, the number of ATM entities involved in the first phase of the project). The network behaviour has 50 ms as link delay, and 0.02 as PLR, based on a measurement campaign described in [11], with message losses not independent as proved by [2]. We have assumed that the coding and decoding time are respectively equal to 5ms and 10ms. We have also considered the block size equal to 1472 bytes, so that an event is fragmented in 16 blocks. We have published 1000 events per each experiment, executed each experiment three times and reported the average.

The metrics evaluated in our study are the following. First, the *Success rate* is the ratio between the number of the received events and the number of the published ones, and this is referred to as the reliability of the publish/subscribe service. If the success rate is 1 (*i.e.*, complete reliability), then all the published events have been correctly received by all the subscribers. Second, the *Performance* is expressed as the mean latency, which is a measure of how fast the given dissemination algorithm is able to deliver notifications, and the standard deviation of the latency, which indicates the possible performance fluctuations due to the applied fault-tolerance mechanisms, highlighting the timing penalties that can compromise the

 ISBN: 978-1-61208-301-8

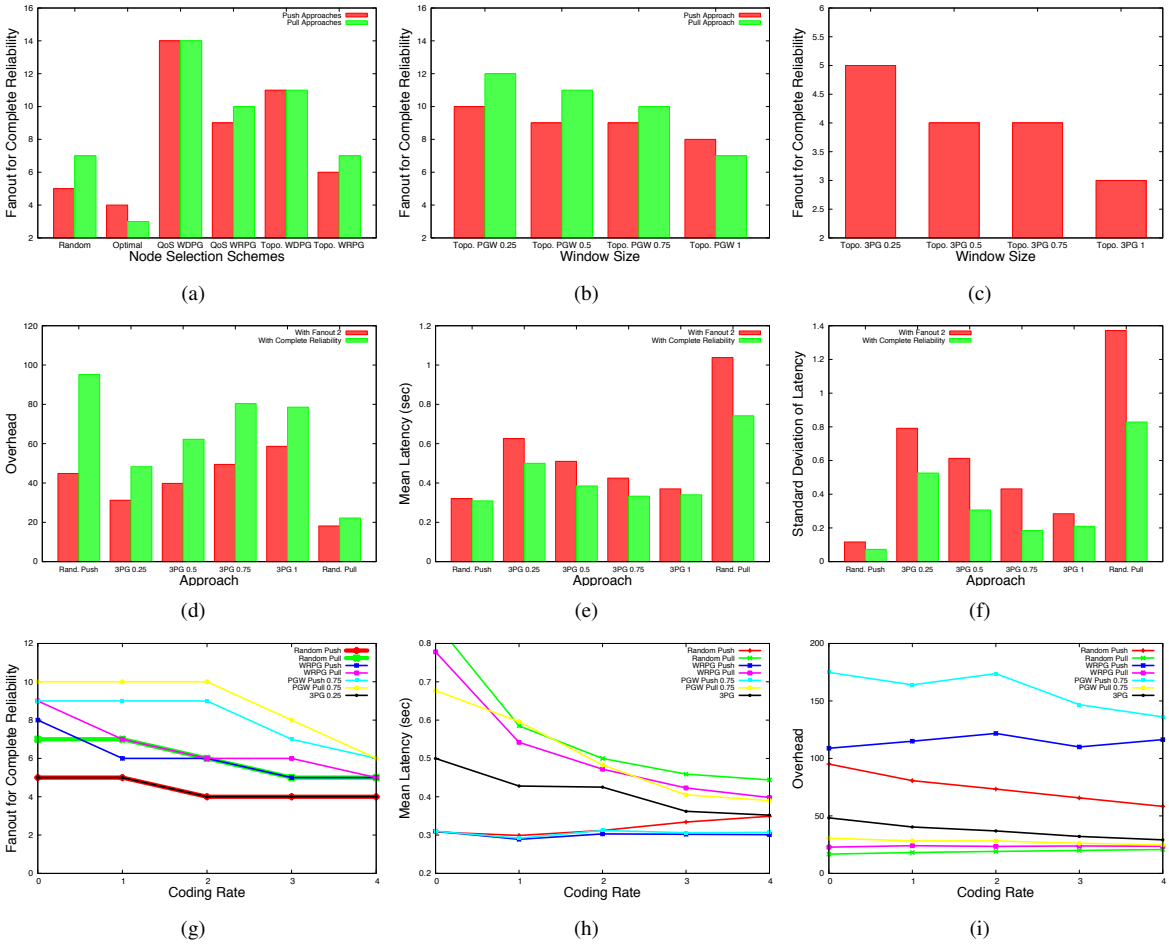


Fig. 2. Quality of the gossip scheme without and with coding.

timeliness requirement. Last, the *Overhead* is the ratio between the total number of datagrams exchanged during an experiment and the number of datagrams generated by the publisher. This is a measure of the traffic load that the dissemination strategy imposes on the network, and should be kept as low as possible, in order to avoid any congestion.

A. Polarized Gossiping

Let us compare the success rate achievable with random selection and the one with the introduced deterministic node selection schemes. A first consideration we can make is that the topology-based selection allows us to obtain a higher mean success rate than the QoS-based one. A second consideration is that polarized gossip, both WDPG and WRPG, allows us to have a lower mean success rate. The reason behind these two considerations is related to the fact that the topology-based heuristics and WDPG imply a strong focus on the nodes at the bottom of the tree, and losses in the higher levels of the tree are more troublesome to be recovered since they are selected less frequently. Such a focus on the nodes at the lower levels is less tight when applying a weighted random selection or if weights are computed with topology information, so that the gossip messages are more distributed, even if the nodes at the bottom still receive more gossip messages.

Let us consider the two schemes we have designed to deal with such issue. Fig. 2(b) shows that the windowing scheme in PGW is able to lower the required f_{out}; but it is not able to bring a considerable improvement than the cases with random selection. On the other hand, 3PG represents a better solution, thanks to its ability to forward pull calls towards the higher part of the multicast tree. From Fig. 2(c), we can notice that the f_{out} required by 3PG to obtain a complete reliability is lower than the one with a random selection. Moreover, lowering the window size implies a reduction of the obtainable success rate and a worsening of the convergence to a complete reliability.

Fig. 2(d) shows the experienced overhead for the gossip approaches with two bars: the first illustrates the overhead with a f_{out} equal to 2; while the second has the overhead when complete reliability is achieved. Generally speaking, there is a difference between these two bars, meaning that increasing f_{out} implies a growth of the experienced overhead. With the only exception being 3PG, the other node selection schemes do not present a lower overhead than the gossip with random selection, since their needed fanout is higher. As depicted in Fig. 2(d), lowering the window size allows us to have a lower overhead (even if the needed fanout is higher),

 ISBN: 978-1-61208-301-8

considerably below that experienced with random selection, as shown in Fig. 2(d): at complete reliability, the overhead of 3PG with a window size of 0.25 is about half of the one with a random push (but is double than the one of random pull).

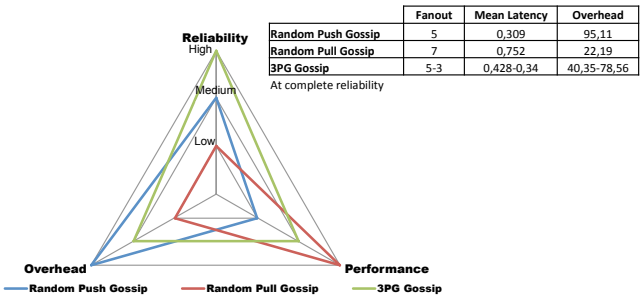
Let us consider the performance of the gossip schemes. Also, in this case we have two bars as in the previous one; however, in this case the second bar is always lower than the first, indicating that an increase of f_{out} brings a decrease in performance. Generally speaking, deterministic selection exhibits a performance below that achieved with random selection. However, such an improvement is nullified by the high number of f_{out} needed to have a complete reliability. Only 3PG is able to achieve a good performance. In particular, the increase of the window size in 3PG lowers the measured performance both in average and standard deviation (as depicted in Fig. 2(e) and Fig. 2(f)). So, with a window size of 1, we have that 3PG is very close to the performance of the random push gossip.

B. Polarized Gossiping with Coding

Coding can improve the quality of the polarized gossip approaches by exhibiting two kinds of benefits. First, when applied during dissemination it can help nodes closer to the root to recover lost events, which is not facilitated by the proposed deterministic selection modes. Second, when applied during gossiping it can speed up the event recovery by reducing the number of needed retransmissions. For these reasons, we have re-executed some of the experiments presented in the previous subsection by applying coding with a redundancy between 1 and 4 (we have considered only the topology-based weights, since coding has similar effects also on the QoS-based ones). For all the approaches, we have experienced a reduction of the value of f_{out} needed for complete reliability, as illustrated in Fig. 2(g). When random selection is applied, coding is able to reduce the needed f_{out} by about 25%. As expected, the improvement is more remarkable in the case of WRPG and PGW, since it is around 40%. 3PG has the same trend as random pull gossip. In Fig. 2(i), we can see a limited improvement of the overhead: the improvements for push gossip in terms of a lower f_{out} are nullified by the overhead introduced by coding. Fig. 2(h) shows an improvement of the experienced performance, even with the delay introduced by performing coding and decoding.

V. Conclusions and Future Work

The random selection of nodes in gossip schemes implies a low utility of the exchanged messages, causing some inefficiencies in terms of experienced latency and overhead. As depicted in Fig. 3, the random push has a higher loss-tolerant capability than random pull since it requires a lower f_{out} for complete reliability. This is achieved with a high overhead and a low performance costs, as opposite to random pull. We have proposed an improvement by presenting several schemes to realize what we called a polarized gossip. In particular, we have shown the pros and cons of such approaches. We have learnt that the best solution when determinism is introduced



	Fanout	Mean Latency	Overhead
Random Push Gossip	5	0,309	95,11
Random Pull Gossip	7	0,752	22,19
3PG Gossip	5-3	0,428-0,34	40,35-78,56

At complete reliability

Fig. 3. Schematic comparison.

in gossiping is to combine the push and pull approaches. As summarized in Fig. 3, such an approach has the highest loss-tolerant capability, and an optimal trade-off between overhead and performance. Deterministic approaches are further improved when coding is applied.

Acknowledgements

This work has been conducted when Christian Esposito was affiliated to the Department of Computer and Systems Engineering (DIS) at the University of Napoli Federico II, and when Marco Platania was affiliated to the Department of Computer and Systems Engineering (DIS) at the University of Roma La Sapienza. In addition, it has been partially supported by the Italian Project of National Research Interest (PRIN) DOTS-LCCI and by the BLEND Eurostar European Project.

References

[1] P.Th. Eugster et al. The many Faces of Publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.

[2] A. Markopoulou et al. Characterization of Failures in an Operational IP Backbone Network. *IEEE/ACM Transactions on Networking*, 16(4):749–762, August 2008.

[3] A.-M. Kermarrec, L-Massoulié, and A. J. Ganesh. Probabilistic Reliable Dissemination in Large-Scale Systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(2):1–11, February 2003.

[4] R. Ahlswede, S.-Y.R. Ning Cai Li, and R.W. Yeung. Network Information Flow. *IEEE Transactions on Information Theory (TIT)*, 46(4):298–313, July 2000.

[5] C. Esposito, S. Russo, R. Beraldi, and M. Platania. On the benefit of network coding for timely and reliable event dissemination in WAN. *Proceedings of the 1st International Workshop on Network Resilience*, October 2011.

[6] C. Esposito, S. Russo, R. Beraldi, M. Platania, and R. Baldoni. Achieving Reliable and Timely Event Dissemination over WAN. *Proceedings of the 13th ICDCN, - Lecture Notes in Computer Science*, 7129:265–280, January 2012.

[7] C. Esposito, D. Cotroneo, and S. Russo. On reliability in publish/subscribe services. *Computer Networks*, 57(5):1318–1343, 2013.

[8] I. Gupta, A.-M. Kermarrec, and A. J. Ganesh. Efficient and Adaptive Epidemic-Style Protocols for Reliable and Scalable Multicast. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 17(7):593–605, July 2006.

[9] G. Hasslinger and O. Hohlfeld. The Gilbert-Elliott Model for Packet Loss in Real Time Services on the Internet. *Proceedings of the 14th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems*, pages 1–15, 2008.

[10] C. Fragouli and A. Markopoulou. A network coding approach to overlay network monitoring. *Allerton*, 2005.

[11] C. Esposito. Data Distribution Service (DDS) Limitations for Data Dissemination w.r.t. Large-scale Complex Critical Infrastructures (LCCI). *Mobilab Technical Report*, 2011.

 ISBN: 978-1-61208-301-8

Self-Recovery Technology in Distributed Service-Oriented Mission Critical Systems for Fault Tolerance

Raymundo García-Gómez, Juan Sebastián Guadalupe Godínez-Borja, Pedro Josué Hernández-Torres, Carlos Pérez-Leguízamo
Central Bank of Mexico (Banco de México)
Av. 5de Mayo #2, Mexico City, 06059, MEXICO
E-mail: {rgarciag, jgodinez, pjhernan, cperez}@banxico.org.mx

***Abstract*—Mission Critical Systems (MCS) require continuous operation since a failure might cause economic or human losses. Autonomous Decentralized Service Oriented Architecture (ADSOA) is a proposal to design and develop MCS in which the system functionality is divided into service units in order to provide functional reliability and load balancing; on the other hand, it offers high availability through distributed replicas. A fault detection technology has been proposed for ADSOA. In this technology, an operational service level degradation can be detected autonomously by the service units at a point in which the continuity of the service may be compromised. However, this technology is limited because it requires human supervision for recovery. In this paper, we propose an autonomous recovering technology, which detects and instructs to service units to be gradually cloned in order to recover the operational service level. A prototype has been developed in order to verify the feasibility of this technology.**

Keywords-Service continuity; fault tolerance; service-oriented architecture; autonomous decentralized systems; fault detection; fault recovery

I. INTRODUCTION AND MOTIVATION

In the presence of a failure, most of the conventional systems implement reactive fault detection and recover mechanisms either automatically or manually. In both cases, the aim is to switch to a redundant or standby computer server upon the failure or abnormal termination of the previously active system. In some cases, the Mean Time to Recovery (MTTR) [15] of these technologies may represent a low risk for the service that the system offers. However, since a failure in MCS may provoke fatal consequences, it is important to reduce the MTTR to a value near to zero

In this paper, we briefly present ADSOA [4][5][6], which has been proposed as a service-oriented architecture for designing MCS, and it has been mainly utilized in financial sector applications. This architecture provides high functional reliability since it is possible to distribute and replicate the functionality of a system in specialized service units. One of the main technologies of ADSOA, called Loosely Coupled Delivery Transaction and Synchronization Technology [6], allows the system to detect when the provision of a service has reached a point in which the continuity of the service may be compromised and it sends a signal alarm to a monitor. This approach may represent a risk for a MCS since it depends on human intervention for taking the necessary actions to repair the system.

This has motivated this paper which presents a technology to autonomously detect and recover gradually all the unit services required for the operational service level in ADSOA. This technology is based on a cloning mechanism that is activated once the operational service level has been compromised due to some failed services units. We describe the protocol and algorithms that the healthy services units utilize in this cloning mechanism and show how they coordinate among them in order to avoid a massive creation of replicas. We developed a prototype in order to illustrate this approach.

The rest of this paper is organized as follows: In Section II, we show the related work. In Section III, we give a view of ADSOA concept and architecture. In Section IV, we present the proposed technology. In Section V, we show a prototype, and finally, in Section VI, the conclusion and future work.

II. RELATED WORK

Cloning technologies have been widely used in different technological areas for providing high reliability to the system in which it is applied. In Optical Burst Switching (OBS) Networks, burst cloning has been proposed as a proactive loss recovery mechanism that attempts to prevent burst loss by sending two copies of the same burst, if the first copy is lost, the second copy may still be able to reach the destination [9][10]. When designing cloning technologies one relevant issue that has to be considered is the resource utilization by the new clones. In this sense, in OBS Networks some technologies have been proposed for optimizing such resource utilization and maintaining a QoS [11][12]. In Multi Agents Systems (MAS), a frequently proposed solution to avoid performance bottlenecks due to insufficient resources is cloning an agent and migrate it to remote hosts [13][14].

Our approach is also comparable to the existing work on cloning technologies in terms of concept and objectives but applied to a novel service-oriented architecture for MCS. The main contribution of the proposed cloning technology are the protocol and algorithms that services units utilize to detect some failures in the service provision and the way they coordinate themselves to recover gradually the operation of that damaged part of the system.

 ISBN: 978-1-61208-301-8

III. AUTONOMOUS DECENTRALIZED SERVICE ORIENTED ARCHITECTURE

A. ADSOA Concept

A proposal used to implement MCS in financial sector is ADSOA [4][5][6], it provides load balancing and functionality, high availability and service-oriented modeling. ADSOA is based on Autonomous Decentralized Systems (ADS) [1][2][3] and Service Oriented Architecture (SOA)[7][8].

ADS is based in the concept of analogizing living things. The living body is built of numerous cells, each growing up independently, maintaining by body metabolism, and going on living. Depending on concept of perceiving the system that it is consisted by autonomous controllability and autonomous coordinability, ADS technology achieves fault tolerance, on-line maintenance and on-line expansion of the computer system. On the other hand, SOA offers a system modeling oriented to services and allows composition and reusability. ADSOA is the combination of SOA concept with ADS characteristics.

The ADSOA conceptual model, shown in Figure 1, is composed of autonomous entities that offers or requests services via messages. Each entity is formed by several instances fully independent. Each instance has the same functionality that its entity represents. A subsystem can be formed by a group of entities and in the same sense a business may be formed by a group of subsystems. This is similar to a living organism where an instance is like a cell, a subsystem could be an organ and the business is like a living organism.

In order to model a MCS using ADSOA, it is necessary to have a service-oriented thinking. At the beginning the system architect identifies the businesses involved in the process and then models the sub-systems in a business according to their responsibility. Finally, entities are modeled according to their atomic functionality. This modeling will allow to the system to grow, evolve, do composition and reuse the components. The next phase is to develop the services entities.

All the systems immersed in ADSOA are able to configure according to physical resources and criticality level. To offer high service availability, it is necessary to have a distributed environment and put on replicated entities. On the other hand, for load balancing it is necessary to divide the functionality in the entities, in such a way that the work be split without a coordinator. The challenge is to provide auto-coordination and auto-control to the system. In this sense, the Autonomous Processing Entity (APE) was proposed; it implements the communication protocol, manages the control instance messages and the services execution. Also, it is possible to define in each service (offered or requested) of the APE its criticality. All these elements form a technology denominated "Loosely Coupling Synchronization and Transactional Delivery Technology".

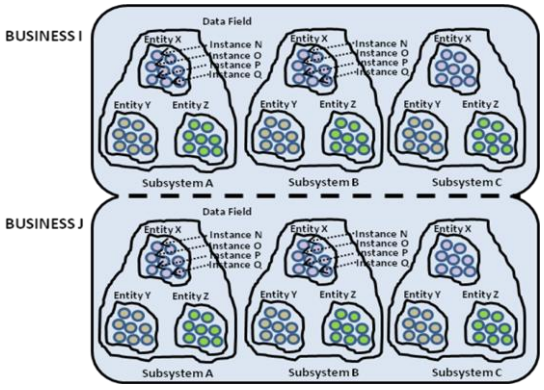


Figure 1. ADSOA Conceptual Model

B. Loosely Coupling Synchronization and Transactional Delivery Technology

In this technology, we define the concept of transaction in the scenario in which an entity requests a service to another and requires knowing if it has been received. The requesting entity must maintain this request in pending processing state until it receives an acknowledgement from receiving entity. Also, we define sequential order in the sense that the entity requester must receive a minimum number of acknowledgments from receiving entities in order to send the next service request, for example, a *X+1* request should not be sent until it receives the minimum number of acknowledgments of the *X* request.

The service request information structure should include the following elements: Content Code, Foliated Structure and Request Information.

The Content Code specifies the content and defines the requested service.

The foliated structure identifies the transaction. This structure is based on:

1. requester id,
2. specialized task id for that request (Pivot),
3. a sequence number,
4. a generated id based on the original request information (event number) and
5. a dynamic and unique id for the instance of the entity (instprintid).

With these elements the identification of acknowledgments received by the entity is guaranteed. We can also ensure the sequence of multiple requests, as shown in Figure 2.

If an instance receives a service request with a sequence number greater than expected, then by the principle of sequential order, knows that another instance of its entity will have the missing messages. In this case, the receiver instance asks to his entity the missed messages, that is, the other instances of the same entity. This idea is represented in Figure 3.

 ISBN: 978-1-61208-301-8

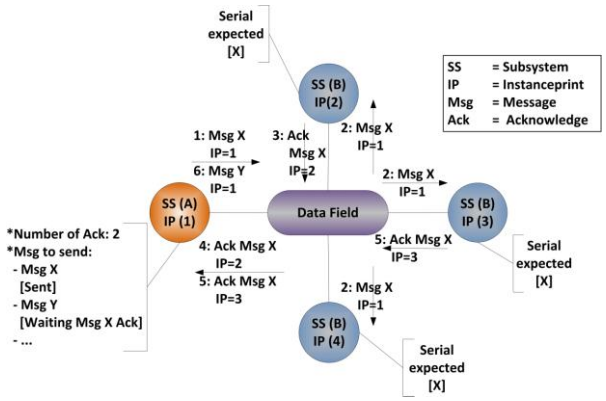


Figure 2. Sequentiality and Transactionality

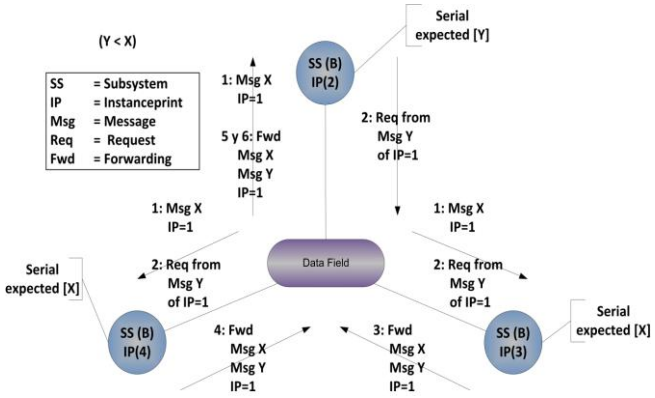


Figure 3. Synchronization with other Instances

On the other hand, if an entity receives several times the same service request, this can be distinguished by the instprintid if this request belongs to the same requester instance or from a different instance of the same entity. According to this, the receiver entity can determine whether requests received are in accordance with the minimum number of requests that the requester entity are required to send, as shown in Figure 4.

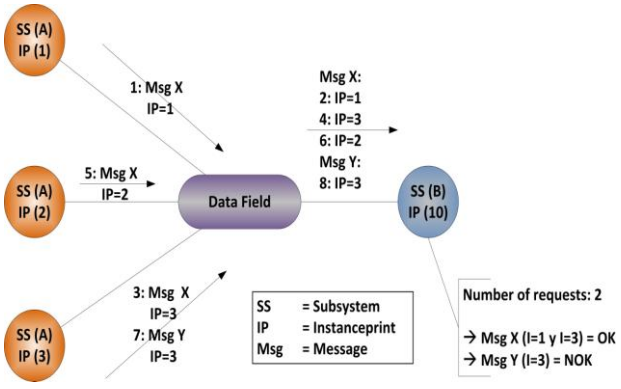


Figure 4. Receiving Multiple Requests from an Entity

In resume, the communication among the elements and its instances is based on an asynchronous and event-driven message protocol. This technology detects if an entity does not provide the service level required. It occurs when an instance sends a service request to an entity and each entity instance receives it and send back an acknowledge message, then sender registers how many acknowledges have received and evaluates if it covers the criticality level, if it is not proper, the sender repeats the sending process. E.g. consider a service with a criticality level equals to 3, its means that this business requires at least three distributed instances; when another instance requests a service to them, it expects at least three acknowledges to satisfy the criticality level, if it is not satisfied the entity will send the request of service again. When the sender detects that the maximum number of retries has reached, it triggers the alert process, which consists in sending an alert message that could be processed by a monitor element. This monitor alerts ADSOA infrastructure managers to perform the necessary activities and recover service continuity (creating new instances required to reach the criticality level). Unfortunately, this goes against MCS's principles since manual intervention is required thereby MTTR becomes dependent on operator's reaction.

In the next section, we present a technology that allows ADSOA subsystems to autonomously detect and recover for a failure in a replicated entity by cloning one by one an operational entity until the system reaches the criticality level required.

IV. SELF-RECOVERY TECHNOLOGY IN DISTRIBUTED SERVICE-ORIENTED MISSION CRITICAL SYSTEMS FOR FAULT-TOLERANCE

This technology is created to allow an MSC that uses an ADSOA infrastructure to self-recover automatically. This basic operation is to use the current self-monitoring scheme and instead of sending alerts to the operator when the service level is not appropriate, it instructs one entity of the degraded group to clone itself (functionality and state). An important challenge in the cloning process is to avoid the generation of multiple indiscriminate copies, which in a living organism would be a cancer. To ensure the healthy recovery, the entity selected to recover the system, generates a cloning-key with information of the times it has been cloned, its id, its instprintid and the requested entity id; this information is introduced into the algorithm to generate the cloning-key, that will be unique to only one cloning process between this entity and the requested id.

In this architecture, all the entities offer and request services, one of this services is the recovering by cloning an entity. In self-recovering technology at least two entities are involved; to explain the protocol let's imagine a group of entities ("A subsystem"), which request a service to other group of entities ("B subsystem"). In Figure 5, "A subsystem" is requesting a service to "B subsystem", the message exchange is carried out in compliance with ADSOA Loosely Coupling Synchronization and Transaction Delivery Technology, with the number of acknowledgments needed to ensure that the level of service is appropriate for. In this example, the "A subsystem", requires 3 acknowledgments by "B subsystem", and the "B subsystem" needs 2 service

 ISBN: 978-1-61208-301-8

requests by “A subsystem”; when the number of acknowledgments is complying, “B subsystem” attends “A subsystem”.

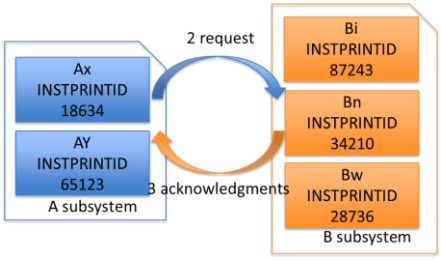


Figure 5. Normal Operation Cycle

If the number of the entities of the “B subsystem” is decreased because of a failure in the process or the server, the “A entities” detects that the service level is not complying within “B subsystem”, since the minimum number of acknowledgments, 3 for this example, cannot be reached within a specific period of time. Thus, the “A entities” starts the recovery mechanism instead of sending alerts.

Figure 6 shows the first steps in the recovery protocol. Firstly, the “A subsystem“ receives the acknowledgments from “B subsystem”. Secondly, based in the lowest “B”’s instprintid all the “A entities” select one healthy entity, which will be responsible for cloning itself. Thirdly, all the “A entities” request the "Auto-Cloning Service (reqidclon)", with the instprintid of the “B entity” selected for auto-cloning. In this example the “Bi entity” will be the responsible for cloning itself; although the “Bn entity” received the same request, only the “Bi entity” will clone. Fourthly, when the “Bi entity” receives the reqidclon request, it generates a cloning-key and sends both this cloning–key and its instprintid as a “Send the key (sendkey)” request service message. By sending its instprintid it can be ensured that the “A subsystem” will instruct to only the selected “B entity” to continue with the cloning process. Fifthly, when the “A subsystem” receives the sendkey service request, it takes the cloning-key in the message and sends it by the “Automatic recovery (autrecov)” request service message to the “B subsystem”, it also attaches to this message the instprintid selected in the second step of this protocol.

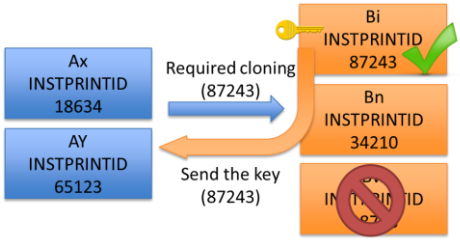


Figure 6. Start up the cloning mechanism

Figure 7 shows the final step of the protocol. When the “B subsystem” receives the autrecov request service message, as it occurs in the third step of this protocol, only the “Bi entity” will attend it, since its instprintid is in the received service message. “Bi entity” will validate if the cloning-key in the message is still valid and if so it will make a cloning of itself. During this process, “Bi entity” will close all the communication with outside and generate a new element in the same state like itself; once the cloning process is finished, it will open the communication again. Otherwise, if the cloning-key is not longer valid because the cloning process has already been completed, the message is ignored. It is important to notice, that the others “A entities” also send this final request service message, but only the first message which reaches “Bi entity” will be processed.

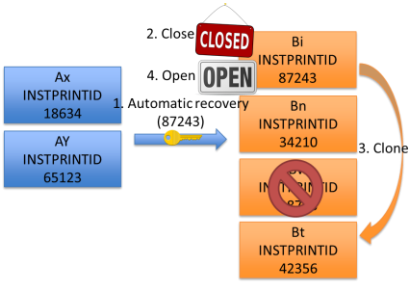


Figure 7. Cloning phase

In this sense, this technology will autonomously maintain the operational service level without human intervention.

V. PROTOTYPE

A prototype which implements this technology has been developed, as shown in Figure 8. This prototype consists of two subsystems with one entity each one, the Requester subsystem/entity, which is shown in blue color, and the Counter subsystem/entity, which is shown in orange color. The Requester demands a service to the Counter for providing a number which later it will be displayed in its screen. When the Counter receives this service request, it will increase by one the previous sent number and send it into a service request message to the Requester. For this example, the service level operation was set to 1 to the Requester and 3 to the Counter; it means that there will be only 1 instance of the Requester and 3 instances of the Counter. On the other hand, the Requester will send its next service request only if it receives from the Counter instances 3 acknowledges for the current request. In order to simulate a failure in a Counter instance, a PAUSE button has been implemented.

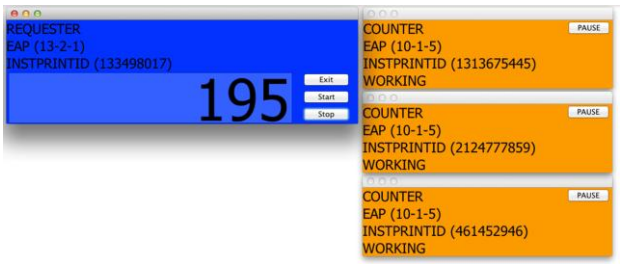


Figure 8. Normal Operation Cycle I (Prototype)

 ISBN: 978-1-61208-301-8

In Figure 9, it is shown that when the entity "2124777859" is stopped, the cloning-mechanism detects such failure and selects entity "1313675445" to repair the system. The reqidclon service request message is sent from the Requester to the Counter with instprintid "1313675445". This entity generates the cloning-key and then it sends the sendkey service request message to the Requester.

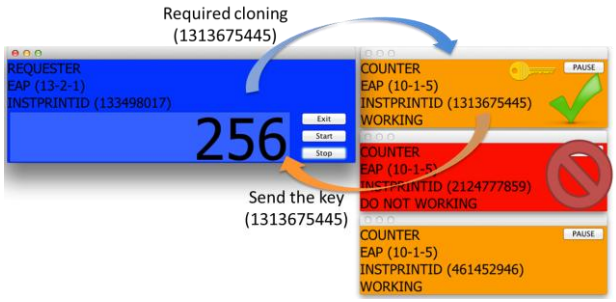


Figure 9. Start up the cloning mechanism (Prototype)

In Figure 10, the final part of the mechanism is shown. The Requester processes the sendkey service request message and sends to the selected entity the autrecov service request message. The "1313675445" entity starts the cloning process; firstly it closes the communication with outside, then it clones itself, and when the entity "191232582" is started, the "1313675445" entity finally opens the communication. In this sense, the system has repaired autonomously the damaged part and it can continue its normal operation.

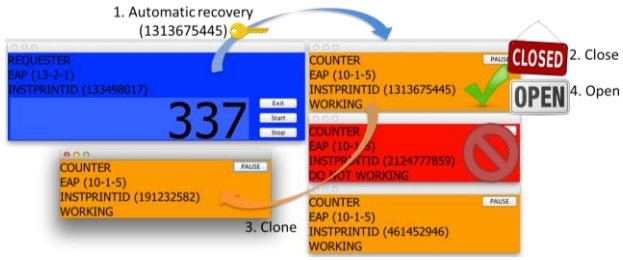


Figure 10. Normal Operation Cycle II (Prototype)

VI. Conclusion and Future Work

In this paper, we briefly presented ADSOA, which has been proposed as a service-oriented architecture for designing MCS, which has been mainly utilized in financial sector applications. We have proposed a cloning mechanism to recover quickly and efficiently the operational service level when a decrease on it is detected. We have built a prototype to verify the feasibility of this technology.

Besides the ongoing development efforts to complete the cloning prototype implementation, future work in this area focuses on get some metrics about resource utilization, network partition and multiple clones' coexistence. We will also compare the proposed technology with others such as those mentioned in Section II.

References

[1] K. Mori, S. Miyamoto, and H. Ihara, "Proposition of Autonomous Decentralized Concept", Journal of IEEE Japan, vol. 104, no. 12, pp. 303-310, 1994.

[2] H. Ihara and K. Mori, "Autonomous Decentralized Computer Control Systems", IEEE Computer, vol. 17, no. 8, pp. 57-66, 1984.

[3] K. Mori, "Autonomous Decentralized Computer Control Systems", First International Symposium on Autonomous Decentralized Systems (ISADS'93), Kawasaki, Japan, pp. 28-34, 1993.

[4] L.C. Coronado-García and C. Pérez-Leguízamo, "A Mission-Critical Certification Authority Architecture for High Reliability and Response Time", IJCCBS Special Issue on Autonomous Decentralized Systems in Web Computing, vol. 2, no. 1, pp. 6-24, 2011.

[5] L.C. Coronado-García, P.J. Hernández-Torres, and C. Pérez-Leguízamo, "An Autonomous Decentralized System Architecture using a Software-Based Secure Data Field", The 10th International Symposium on Autonomous Decentralized Systems (ISADS'11), Kobe, Japan, 2011.

[6] L.C. Coronado-García, P.J. Hernández-Torres, and C. Pérez-Leguízamo, "An Autonomous Decentralized Service Oriented Architecture for High Reliable Service Provision", The 10th International Symposium on Autonomous Decentralized Systems (ISADS'11), Kobe, Japan, 2011.

[7] Thomas Erl, 2005, "Service-Oriented Architecture (SOA): Concepts, Technology, and Design", Ed. Prentice Hall.

[8] Nicolai M. Josuttis, 2007, "SOA in Practice: The Art of Distributed System Design", Ed. O'Reilly Media.

[9] H. Xiaodong, V.M. Vokkarane, and J.P. Jue, "Burst cloning: a proactive scheme to reduce data loss in optical burst switched networks", IEEE International Conference on Communications (ICC'05), Seoul, Korea, 2005.

[10] S. Riadi and V-A. Mohammed, "A decision algorithm for efficient hybrid burst retransmission and burst cloning scheme over star OBS networks", Second International Conference on Innovating Computing Technology (INTECH'12), Casablanca, Morocco, 2012.

[11] L. Ji and K.L. Yeung, "Burst cloning with load balancing", Optical Fiber Communication Conference (OFC'06), Anaheim, California, 2006.

[12] S. Askar, G. Zervas, D.K. Hunter, and D. Simeonidou, "Classified cloning for QoS provisioning in OBS networks", The 36th European Conference and Exhibition on Optical Communication (ECOC'10), Turin, Italy, 2010.

[13] O. Shehory, K. Sycara, P. Chalasani, and S. Jha, "Agent cloning: an approach to agent mobility and resource allocation", IEEE Communications, vol. 36, no. 7, pp. 63-67, 1998.

[14] D. Ye, M. Zhang, and D. Sutanto, "Cloning, Resource Exchange and Relation Adaptation: An Integrative Self-Organisation Mechanism in a Distributed Agent Network", IEEE Transactions on Parallel and Distributed Systems, vol. PP, no. 99, pp. 1, 2013.

[15] P.A. Laplant and S.J. Ovaska, 2011, "Real Time Systems Design and Analysis", Ed. Wiley-IEEE Press.

 ISBN: 978-1-61208-301-8

Agents for Fault Detection and Fault Tolerance in Component-based Systems

Meriem Zaiter
Larbi Ben M'hidi University of Oum El-Bouaghi,
LIRE Laboratory at the University of Constantine 2,
Constantine, Algeria
meriem.zaiter@gmail.com

Salima Hacini and Zizette Boufaida
LIRE Laboratory at the University of Constantine 2,
Constantine, Algeria
{salimahacini, zboufaida}@gmail.com

***Abstract*— In recent years, the use of component-based systems has become increasingly large in the daily life such as domestic applications. In addition, the diversity and the dynamic components that can build them make this type of system very awareness. For this reason, the assurance of dependability and safety execution are required in order to propose a better system performance and the best user satisfaction by providing services continuity which consequently leads to reliability. That is really a challenge problem. Our goal in this paper is to propose an adaptation mechanism, based on the mirror services, to make such a system more efficient, and thus, more and more operational, even the existence of faults in it. To this end, the fault tolerance is a good solution. So, the contribution in this paper is based on a set of algorithms that will be employed by a set of local agents controllers and one global agent controller.**

Keywords-dependability; agent; fault detection; fault tolerance.

I. INTRODUCTION

Multi component system is composed of many different components, each of which is an individual system. The complete system has a set of fixed functionalities. Every component may have a varied composition and implementation. As an example, we find the domestic applications, which use a computer component in the home environments. In this space, the wireless communication between the components and the sensors devices are generally used. So, that system is very awareness and dysfunction error localization is a delicate task. Indeed, an abnormal execution in this kind of system can be caused by the failure of any component, which can imperatively causes a dysfunction of the overall system. So, it is indispensable to detect such situation before the crash of the overall system. One way to insure dependability of systems is to allow continuity of execution in the case of fault occurrence, which is the aim of this paper. A promising technique to do this; is the fault tolerance, which is defined as the ability of system to continue normal operation despite the presence of faults.

Our fault detection mechanism [1] focuses on the use of a global controller and a set of local controllers, which aims first to detect the fault whatever its nature then applies the algorithms to support it automatically, and adapt the execution to the suitable context. In this paper, our goal is to enhance the mechanism with the use of both the agents and the replication advantages. The replication is considered as one of the basic tools in a fault tolerance technique [2].

Moreover, the agent technology has been largely used and gives an interesting proposition to various problems such as e-commerce, distributed computing telecommunication networks services, monitoring and notification [3], etc. They provide several advantages, in the dependability area, the fact that an agent [4]:

- Has the ability to communicate
- Can migrate from a defective component to another in order to continue its execution, by the weak or the strong mobility characteristics.
- Can keep track of the execution follow.
- Can be duplicated and cloned as needed, or killed for example in a case of its failure
- Uses of low-cost and a low-power requirements when it is executed on an equipment
- Etc.

Also we find that the interaction agent-agent is exclusively via message-passing communication and the asynchronous message-passing have good scalability characteristics.

The remainder of the paper is devoted to the details of our agents based fault detection mechanism. Section 2 gives a state of the art of the fault detection techniques. Section3 presents an overview of faults' types that may affect the normal function. Section 4 details all the algorithms that handle the detected faults. Finally, a conclusion achieves this paper.

II. RELATED WORK

Fault tolerance is an indispensable characteristic required by different types of computer systems and specifically distributed systems. The latter can fail due to the failure of its components; why researchers are still trying to find a way to ensure dependability by fault tolerance. In some studies, [5, 6] the authors propose a service migration from one component to another to ensure a permanent presence of service despite it is being required; this can be insured through three mechanisms, the first is used to manage the context of interaction among the system components; the second is employed to specify the rules according to the current context and the changes that may occur, and the third one identifies the migrated service. This approach has some disadvantages:

- It is only applied in a context where all components of the system have the same architecture on which mobile service can migrate.

 ISBN: 978-1-61208-301-8

- Regardless to the state of the component the service should automatically migrate (even if the component was in a good state). So, unnecessary transmissions can be realized. In addition this technique is ineffective in case of a sudden fault (there is no fault detection) [5].

There are also some mathematical methods for fault detection. For example, in [7,8] the approach is rule-based. It requires first a definition of a fault model that categorizes the occurred fault as short, constant or noise, then based on the standard deviation between the later model and the current system state a fault can be detected. These methods [7][8] require a knowledge on the domain to identify the faults' type and a careful specification of the standard deviation threshold.

The method used in [9] also requires the definition of the types of faults that can occur in order to detect them on run-time. Some other software techniques [10][11][12] add a set of instructions to control the flow of execution, and thus detect the existence of fault by comparing for example a duplicated variables values with the variables themselves [12]. Generally, these techniques rely on external equipment to handle the fault that will certainly cause a significant perturbation treatment.

Some techniques incorporate exceptional behaviors during the entire development of fault tolerant distributed systems implemented within component [13]. Other model introduced in [14][15], specify the normal and exceptional behaviors of system components; so while exceptional responses, errors are detected.

In spite of the number of solutions to insure fault tolerance, the fault problems' detection and support persist and not treated definitely. On the other side, the requirement of safety running of systems and the availability of delivered services is very required.

III. THE FAULTS'S CATEGORIES

An error is the manifestation of a fault in the system, and a failure is the manifestation of an error on the provided service by the system [16]. The fault type plays a very important role if we want to get a fault tolerance. Moreover, faults can be categorized according to several criteria, like the degree of severity, degree of permanence and their nature.

The based component systems are considered as context-aware systems because a communication context varies from one moment to another. So, in order to classify faults, we exploit, in this section, the following definition of context [1] "*Context is any internal or external information, related to an entity, could be used and have an impact on the future state of the application. This information can be linked to one or more entities. The latter, regardless of their nature (hardware / software / human), can trigger events that affect the global state system. To this end, the occurrence of a fault causes certainly an immediate dysfunction to the global system*".

So, we suggest classify the faults on the base of the faults's sources, their manifestations and their persistence. One component in a component-based system may announce a dysfunction on behalf of another component, if the latter did not satisfy the needed request.

Also, failures can be a result of an improper use of the system by the user or due to:

- Software errors: that can be an arbitrary deviations related to the code,
- Materials errors: that can be the shutdown of a component or its internal constituents, or
- Transmission errors: such as the omission of sending or receiving messages or even to malicious attack (citing as an example an injection of a code into the system, by a malicious user, can cause a deviation of the normal execution flow).

In Table 1, the source of the fault is related to the element of context. The persistence of a fault means its duration; it may be transient or permanent. A permanent fault is a fault which requires a software maintenance or human intervention.

TABLE I. THE FAULTS' CATEGORIES

Elements of context		**Categories of faults**	
		external fault	***internal Fault***
Entity	Hardware / user	An error of interaction (such as error identification, or an input mistake ...)	/
		(transient / permanent) fault	
	Hardware	/	–Error referencing of an internal component (processor, memory…) –internal hardware failure
			permanent fault
	Software/ user	An entry outside the domain specification of the application	/
		(transient/ permanent) fault	
	Software	/	Design fault in the application itself
			permanent fault
Temporal aspects (date, time)		Fault in scheduling and synchronization of messages among system components	– the local Clock is not synchronized – physical error due to a transmission problem
		transient fault	(transient/ permanent) fault
Location		Localization problem of neighboring entities (effect of fog in the environment)	Fault in the physical controllers of component
		transient fault	permanent fault

 ISBN: 978-1-61208-301-8

After the presentation of the proposed faults' categories, the next section details our proposition that aims to describe how the presented kind of faults are detected and supported.

IV. THE DETAILS OF OUR PROPOSITION

Dependability of our system is associated with the dependability of its components. It can be provided by insuring the availability of the exchanged events and services among the system components (each component will be called entity "ei") (Figure 1).

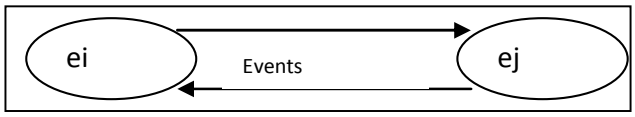


Figure 1. the dynamic exchange of events in the system.

To ensure a high dependability of the system and its entities, we focus, as it is explained in the introduction, on the advantages of agents as well as the replication. To this end, some tools (local and the global agents controllers (see the next sub section)) and the next constraints must be set:

- Each entity (ei) proposes a set of services OS (ei) that can be provided in the form of quadruplet:
  OS (ei)={(s0, child(s0), c0 , a0),(s1, child (s1), c1,a1),(s2, child (s2), c2,a2)…} where child (si) are all sub-services managed by the service "si" ," ci" and "ai" denote respectively the cost and the availability of the service (the latter takes the value available or busy). These quadruplets are sent to the global agent controller by the entity.
- For each connection of an entity (ei) to the system, two tasks will be performed:
  - An entity sends its request.
  - Upon receiving the entire answers, the entity defines all the functional dependencies D (ei) and sent them to the global agent controller. The Functional dependencies D (ei) are defined by a set of pairs, as an example (D (ei) = {(ej, sk) ...} where sk is the delivered service by ej to ei.
- Each exchanged message must be double signed (through an hash function) by both the entity and its agent controller that helps in the control task of the local agent controller operation itself. So, if an agent controller does not sign its message a "Raise not-signed" event will be reported.
- A duplicated global agent controller is set and updated periodically in order to take the control task if the principal global agent controller fails.
- Each agent is supported by an entity and it (the agent) will be killed if this entity fails or disconnects; except the global agent controller and its cloned agent (duplicated agent) which will migrate, if the entities where the agents are running fail.

A. *The global agent controller*

The global controller is seen as an agent. It provides a set of functionalities:

- Manages the faults' detection and,
- Takes charge the occurrence of faults.

So, switch the kind of the received event the agent performs the suitable action. To do this, the global agent controller (Gac) must has a set of information in database knowledge. That contains an entry for each entity composed of (see, table 2):

- the identity of the entity noted ei,
- its state as a set of pairs (ss, state) where each pair represents a name "ss" and the state (that can has the value "good" or "bad") of every its provided service.
- a description of the list of the offered services of the current entity and all its functional dependencies
- The execution state "ESi" which represents the status of the executing operation, on the entity, which is periodically updated.

TABLE II. THE REPRESENTATION OF THE INFORMATION THAT CHARACTERIZES AN ENTITY AT GAC.

The entity	state	offered Service	Dependence	Execution state
e1	(s1, good) (s2, bad)…	(s1, child (s1), c1, a1) …	(e2, s4) ……	ES1

1) *The global agent Controller as a manager of a fault detection*

The operations of the global agent controller are detailed through a set of algorithms (see Figure 2), some of them used to manage events flowing through the system, as the indication of a fault of an entity ei:

- By its local agent controller (see Figure 2, instruction 10) or,
- By another local agent controller (other than its own agent controller) (see Figure 2, instruction 15).

Upon receipt of the entire functional dependencies of a given entity (ei) the global agent controller exploits them to update the availability of services (instruction 9 in Figure.2) from "available" to "busy".

The other types of events are detailing in the next algorithm:

```
1 Input: event
2 Begin
3 Repeat
4       case (event) of:
5       nw_elt:
6       Creat and send the local agent controller Laci
7       Load _DB (ei)
8       D(ei):
9       Update availability (D (ei))
10      alert (Laci , ei,ss) :
11      P← takecharge(ei,ss)
12      Send fault (ei, Gac,ss) to every element of P
13      For every controller element of (P) do: Research (E,S)
14      For every controller element of (P) do: Send Mirror
        (Gac, em, em)
15      alert (Laci , ej,ss) or good_state(Lacj ,ej,ss) :
16      Check_state (ej,ss),
17      Raise not-signed (i): preparing and send a new Laci
18Until (false)
19End
```

Figure 2. the global agent Controller algorithm.

– **Terminology**

nw_elt: means that a new entity is connected to the overall system.

Research (E, S): allow proposing a set of mirror services (ms) to the entity affected by the fault.

Load _*DB (ei)*_: adds to the database an entry contains the information concerning the entity (ei) (state (ei), D (ei) ...) (see Table 2).

Extract_state (ej, ss): This function enables the recuperation of the service state ss (ej) from the database (see Table 2).

Raise not-signed (i): this event means that there is a dysfunction on the Laci, so a substitution of the defective Lac must be done. Therefore, the new agent takes the needed information like (the most recent value of the execution state ESi) and continues the control of its entity.

Check_state (ej,ss) is a verification function (see Figure 3) for testing the entity state.

All the other instructions will be carefully explained in their appropriate context.

```
1 Input : entity : ej, service ss
2 Output: state of ej
3 Begin
4          State ( ej,ss) ←Extract_state (ej,ss);
5   if (state (ej,ss) =bad) then
6          send fault (ej,Gac,ss) to Laci
7   else
8          send Rv (Gac,ej,ss) to Lacj
9          if (Rep (ej,ss)) then
10         send good_state (Gac, ej,ss) to  Laci
11         else
12         P← takecharge(ei,ss)
13         send fault (ej, Gac,ss) to every element of P
14         For every element of (P) do research (E,S)
15         For every element of (P) do Send Mirror(Gac, em, em)
16         End if
17  End if
18End
```

Figure 3. Check_state function.

Rv (Gac, ej, ss) is a verification request sent by the global agent controller to (ej) in order to test its state.

2) The agent global Controller as a responsible of the fault tolerance

Upon the confirmation of the detection of a fault the global agent controller performs the two following tasks to support the fault:

- Declares the entity (ei) as partially defective, in service s, by following the algorithm in Figure 4, which may indicate the fault of the service provided by an entity to all its dependencies (see Figure 2 instruction 11and 12; Figure 3 instruction 12 and 13):

```
1 Input : (ei, s)
2 Output : list of pair P of (entity e, service s)
3 Begin
4     For j=1 to NE do
5          repeat
6           Dt(ej)← D(ej)
7          (en,sn) ←  extract an element from  Dt(ej)
8          if (en = ei) and sn belong to {s} U {child (s)} then
9                    insert P(ej,sn)
10                   update_state (ei,s)
11         End if
12         until Dt(ej)= Φ
13    End For
14         return (p)
15 End
```

Figure 4. Algorithm of the takecharge function.

NE: indicates the number of entities in the system

The procedure update_state aims to update the operational state, by " bad", of the defective service "s" and its child(s).

- The second sub-task of the global controller is to make sure the continuity of system and the service deliverance by following the algorithm (in Figure 5) that aims to research the similar services (see Figure 2, Instruction 13; Figure 3, Instruction 14)

(es, ec, ae) : corresponds to the elected service

```
1 Input : (ei, s)
2 Output : list of pair P of (entity e, service s)
3 Begin
4          ae ←  busy// the availability of service
5          State e←  bad // the operational state of service
6          es←s
7          ss←es
8          ec←max (c)
9          elect←i
10  For j=1 to NE do
11         if ((ss = es) and (ae =available)
           and (sc <= ec) and ( state s = good))  then
12                   (es, ec, ae) ←  (ss, sc, ae)
13                   elect← j
14         End if
15  (ss, sc, ae) ←  extract an offered services from the table
16   State e ←  state (ss)
17  End For
18   if (State e = bad) then
19         return (Φ , Φ )
20   else
21         return (e elect , es)
22   End if
23 End
```

Figure 5. The function Research

B. The local agent controller

To insure our control a set of a local controllers have been used; these controllers are seen as a local agents noted (Lac), one for each entity. Our suggestion of implementing an agent controller is based on the idea of self-testing, which allows individual control of each entity. This local controller executes a set of tasks that allows it to control the operational state of the entity. Faults are declared if an entity deviates from this normal operation:

- the entity sends and saves a simple request (RS (ei, ej, ss), (save (RS (ei, ej, ss))), waiting for answers (waiting (RP (ej, ss))) (see Figure 6 instructions 7 thru 9), and a possible definition of functional dependencies (see Figure 6 instructions 7 thru 9),

So, the abnormal functional of an entity is represented as events sent by the agent. Such as sending an alert (alert (Laci, ej, ss)) if an entity ej has promised the entity ei to ensure the service ss and it has not responded, or if it returned a wrong result. Indeed, each controller provides the following tasks:

1) Keeping track of execution in order to capture the execution state, this will be used in a fault recovery (see Figure 6, instruction 42).

2) Receiving the external events coming into the entity ei. In this case an external event can be:

- a simple request from an entity ej (Figure 6, instruction 5);
- a negative feedback from the Lacj, the controller of an entity ej, that is resulting from an eventual previous interaction with the entity ei;
- An inquiry concerning the entity ei, or an information failure of an entity ej if ei depends functionally from the defective entity ej (this event is raised by the global agent controller) (Figure 6, instruction 10…);

3) Informing the failure of its entity ei (see Figure 6, instruction 18, 22,…); in the case of a no-response to the periodical test of inspection performed by the agent controller Laci itself,

The clarification of the terminology used in the next algorithms (in Figure 6 and Figure 7) is explained bellow:

RS (ei, ej,ss): it is a simple request send by ei to ej requesting the service ss.

RP(ei,ss/Gac): it is an answer for a request sent by the entity (ei) (or Gac).

Fault (ej, Gac,ss): it indicates a failure of an entity ej, at the service ss, reported by the global agent controller.

Rep (ei,ss): this is a Boolean function. It represents the answer or not of ei to the local test relating to the service ss, triggered by the local agent controller Laci .

Check_ local _state (ei,ss, t): it is a function that represents the local test triggered after a time t. This function has a value 0 if the service ss of the entity ei did not answer, and 1 otherwise (see Figure *9).*

Time: it represents the duration between two periodical tests.

Alert (Laci , ei,ss): It denotes a failure of a service ss of the entity ei reported by the agent controller Laci .

good_state (Laci , ei,ss): it is an event emanated from the local agent controller Laci. It shows that the service ss of it entity ei is in a good state.

good_state (Gac, ei,ss) it is an event emanated from the global controller. It shows that the entity ei is in correct state.

fd (ej,ss): denotes a promise from the entity ej to perform the service ss (functional dependency in the service ss).

Verify_Rq (ei, ej,ss): the role of this function is to check if a request emitted by an entity ei is being processed by an entity ej or not.

Save (RS(ei ,ej,ss)) : its role is to save the request (sends by the entity (ei)) that will be processed by the entity (ej)

Wait (RP(ej,ss)): it aims to start the control of the duration of the response of an entity ej to the ei request's.

Mirror (Gac, em, sm): indicating the elected service mirror " sm" and the identity of the entity that provides them. In order to ensure the continuity of operation of the overall system.

```
1 Input: event;
2 Begin
3 Repeat
4  case (event) of:
5   RS(ej ,ei,ss) :
6          save (RS(ei,ej,ss)) ;
7          Send fd((ei,ss) to Lacj
8          Treat  (RS(ei, ej,ss)) ;
9          send (RP(ei,ss)) ;
10 fault (ej ,Gac) OR alert(Lacj, ej,ss) :
11     If (Verify_Rq (ei, ej,ss)) then
12                 Cancel  (RS(ei,ej,ss))
13    else
14                 state (ej)← bad
15    End if
16 alert (Lacj, ei,ss):
17    if (state (ei,ss)=bad) then
18         send  alert (Laci ,ei,ss) to Gac
19         send  alert(Laci ,ei,ss) to Lacj
20    else
21    if ((check_ local _ state (ei,ss,0))=0) then
22         send  alert(Laci, ei , ss) to Gac
23         send  alert(Laci ,ei,ss) to Lacj
24    else
25         send good_state(Laci ,ei,ss) to Lacj
26         send good_state(Laci ,ei,ss) to Gac
27   End if
28   End if
29  fd (ej,ss):
30         update dependence D(ei)
31         Send D(ei) to Gac
32 good_state(Lacj , ej,ss) and not RP(ej,ss):
33         send RS(ei,ej,ss) to Lacj
34 good_state(Gac, ej,ss) and not RP(ej,ss):
35         send RS(ei,ej,ss) to Lacj
36         save (RS(ei ,ej,ss)) ;
37         wait (RP(ej,ss));
38 Mirror (Gac, em, em)
39          send RS(ei,em,sm) to Cm
40          save (RS(ei ,em,ss)) ;
41          wait (RP(mj,ss));
42         continue the execution from the current state
43 good_state(Lacj,ej,ss) and not RP(Gac):
44         send RS(ei ,ej,ss) to Lacj
45         raise degraded mode
46         activate the duplicated Gac
47 until (false)
48 End;
```

Figure 6. The agent local controller algorithm.

The periodical test is performed by the execution of the "check_ local _ state" function (see Figure 7) that will be performed by the local agent controller, every a given time t or in any other necessary moment like in the case of the instruction 21 in Figure 6.

```
1 Input: entity ei,ss, time t;
2 Begin
3       if (not (Rep (ei))) then
4               state (ei,ss)← bad
5               send alert(Ci ,ei,ss) to Gac
6               t←time
7               Return (0)
8       else
9               state (ei)← good
10              t←time
11              Return (1)
12      End if
13End
```

Figure 7. The function check_ local _state.

This last function (Figure 7) is used by the agent Lac(i) to control the operational state of entity ei and all its offered services.

V. CONCLUSION

This paper contains an effective contribution in fault tolerance area applied to component-based systems. First, some faults' categories have been established by giving an overview of the errors' types, then this paper describes our reflection to insure a high dependability by a fault tolerance, which is based on a global agent controller and a set of local agents controllers. Diverse situations (theoretical scenario) have been treated: (1) even a fault, insuring the continuity of delivering services, in a right way, by exploiting the agent ability of keeping tack to capture the recent execution context. (2) Insuring continuity of control even a dysfunction at the global agent controller itself or at one or more local agent controllers, through the use of the following features: the replication and the migration ability, etc. In order to validate the proposed mechanism, a simulation of a domestic application is on the way with the purpose of giving some statistics; and improving our theoretical. We have chosen an application for monitoring a patient at home, on which we have selected a set of adequate components, some components are strongly coupled and other are not, to inject faults and test how the system react, etc.

REFERENCES

[1] M. Zaiter, S. Hacini, and Z. Boufaida "Towards a Functional Control of a Context-Aware Systems", Information Studies: Online ISSN: 1911-8414, vol. 4, no.1 January 2012, pp, 9-17.

[2] C. Leangsuksun, T. Liu, L. Shen, and S. L. Scott. "Building high availability and performance clusters with ha-oscar toolkits". Proc. the High Availability and Performance Workshop, Santa Fe, NM, October, 2003.

[3] D. B. Lange and M. Oshima, "Seven Good Reasons for Mobile Agents". Communications of the ACM, vol. 42, no.3, March. 1999.

[4] H. Paulino "An Overview of Mobile Agent Systems'", Technical Report Series: DCC-02-1. February. 2002.

[5] O. Riva, J. Nzouonta, and C. Borcea. "Context-Aware Fault Tolerance in Migratory Services", MobiQuitous 2008, Dublin, Ireland, July 21- 25, 2008.

[6] R. Handorean, R. Sen, G. Hackmann, and G.-C. Roman. "Context Aware Session Management for Services in Ad Hoc Networks". Proc. IEEE International Conference on Services Computing (SCC'05), July, 2005, pp, 113–120.

[7] Ramanathan et al. "The Final Frontier: Embedding Networked Sensors in the Soil", Tech. Rep. 68, CENS. November. 2006.

[8] A. B. Sharma, L. Golubchik, and R. Govindan "Sensor faults: Detection methods and prevalence in real-world datasets" Journal ACM Trans. on Sensor Networks TOSN Vol. 6, Issue. 3, A23, ACM New York, NY, USA, June. 2010.

[9] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom,. "Declarative Support for Sensor Data Cleaning". Proc. 4th International Conference. Dublin, Ireland,, pervasive 2006.May 7-10. 2006, pp, 83-100.

[10] N. Oh, P.P. Shirvani, and E.J. McCluskey, "Control Flow Checking by Software Signature" IEEE Trans. on Reliability, vol. 51, no. 1, 2002, pp, 111–122.

[11] Cheynet, et al. "Experimentally Evaluating an Automatic Approach for Generating Safety-Critical Software with Respect to Transient Errors," IEEE Trans. on Nuclear Science, vol. 47, no. 6, 2000, pp, 2231–2236.

[12] M. Rebaudengo, M. Sonza reorda, and M. Violante "A New Approach to Software-Implemented Fault Tolerance". journal of electronic testing: Theory and Applications 20, Kluwer Academic Publishers. United States. 2004, pp, 433–437.

[13] C. M. F. Rubira, R. de Lemos, G. R. M. Ferreira, and F. C. Filho. " Exception handling in the development of dependable component-based systems". Softw. Pract. Exper., 35(3): , 2005,pp,195–236.

[14] P. Lee and T. Anderson. "Fault Tolerance: Principles and Practice", Second Edition. Prentice-Hall, 1990.

[15] A. Bucchiarone "Architecting Fault-tolerant Component-based Systems: from requirements to testing"; Electronic Notes in Theoretical Computer Science 168 Elsevie, 2007, pp, 77–90.

[16] J.P. Blanquart "Sûreté de Fonctionnement des systèmes embarqués critiques Les enjeux industriels (Domaine Spatial) (Astrium Satellites)" ETR'09 Ecole d'Eté Temps Réel, TELECOM ParisTech, Aug 31-Sept 4, 2009 (in French).

 ISBN: 978-1-61208-301-8

Modeling and Analysis of State/Event Fault Trees using ESSaRel

Kavyashree Jamboti*, Michael Roth*, Robin Brandstädter*, Peter Liggesmeyer†
*Technical University of Kaiserslautern, Dept. Software Engineering: Dependability
Building 32, Paul-Ehrlich-Straße, 67663 Kaiserslautern, Germany
{jamboti, michael.roth, r_brands}@cs.uni-kl.de
†Scientific Director
Fraunhofer, Institute for Experimental Software Engineering
Kaiserslautern, Germany
Peter.Liggesmeyer@iese.fraunhofer.de

***Abstract*—Fault Trees (FTs) have been a popular tool used in the industry and academia to model safety related failure scenarios of systems. However, since FTs are incapable of modeling certain type of scenarios involving stochastic dependency, timing and sequencing properties, they need to be extended or modified to handle such scenarios. A State/Event Fault Tree (SEFT) is one such tool for developing and analyzing systems with dynamic behavior involving sequencing, timing and priorities of events that cannot be modeled by ordinary fault trees. SEFTs encompass dynamic behavior in the form of state charts for constituent components of a system where failure propagation between components is made possible by outports and inports. Conceptually, SEFTs borrow the notion of components from Component Fault Trees (CFTs). CFTs are nothing but fault trees which encompass boolean logic related to failure within the corresponding component boundaries. The ESSaRel tool was initially built to model CFTs. In this paper, we describe our experiences with the implementation of an editor for SEFTs by extending the ESSaRel tool. We describe the concepts behind the design decisions of the tool and the challenges that were addressed in order to reduce the burden on the user to develop 'correct' SEFTs. We also give some insights and tips for engineers who would like to use SEFTs as modeling correct SEFTs requires a good understanding of the semantics of its modeling elements.**

Keywords*—*Fault trees, Reliability tool, Safety tool, State/Event Fault Tress, ESSaRel

I. INTRODUCTION

Fault Trees (FTs) are an established method for conducting safety analysis of systems due to their ability to provide both qualitative and quantitative analysis results. They are able to capture those combinations of events that lead to a compromise in safety of systems which might not have been captured by other safety techniques. With the advent of component-based development, it became important to be able to create models for individual components, which can be combined to create a system model. However FTs have no notion of components, a functional/structural change made to one component implies that the entire FT has to be reconstructed, or at least it has to be ensured that there is no need to make further changes to the FT after making changes to the corresponding part of the FT. This can be a time consuming task considering that FTs can run into several pages depending on the size of the system. Component Fault Trees (CFTs) [1] combat this problem very elegantly by introducing component boundaries around failures and gates where failure propagation between components is facilitated using outports and inports. This makes it very easy to identify where the changes in the CFT has to be made while the rest of the CFT remains unchanged. Although one can easily modify existing CFTs and combine CFTs for constituent components to obtain a CFT for the whole system, they are just an extension of FTs and still have the inherent shortcomings of the fault trees' inabilities to model timing and sequencing of failure events. Also, there is no difference between states which refer to a persistent condition of a component and an event which refers to an occurrence without a temporal expansion. State/Event Fault Trees(SEFTs) [2] go one step further by extending CFTs to include the above mentioned features by introducing state-charts in components. These state-charts capture failure related behavior of components. Events and states can influence and be influenced by the behavior of other components through ports. SEFTs offer a wide range of gates that render it a powerful tool to model a variety of safety scenarios. Furthermore, any other specialized gate can be modeled by the user as a component that can be reused whenever it is required. Please note that unlike gates in a FT, which are capable of only enclosing boolean logic, gates in SEFTs can be thought of as components with an internal state-chart of their own which are capable of modeling sequencing, timing and memory. ESSaRel (Embedded Systems Safety and Reliability Analyzer) is a tool that was initially designed to provide an editor to model and analyze CFTs. It was then later extended to enable modeling and analysis of SEFTs. SEFTs have to be transformed to extended Deterministic Stochastic Petri Nets (eDSPNs) in order to be analyzed. The extended version of ESSaRel provides an editor for modeling SEFTs and also implements the translation algorithm that converts an SEFT to an eDSPN in a format compatible with the TimeNET tool [3], which can be used to analyze the translated petri nets.

SEFTs, although powerful, require a thorough understanding for their proper usage. In this paper, we describe how ESSaRel assists a user in creating SEFTs without syntactic errors. We describe our experiences of extending ESSaRel for modeling SEFTs. We also provide pointers regarding how to use SEFTs in the real world in order to create meaningful and semantically correct models.

 ISBN: 978-1-61208-301-8

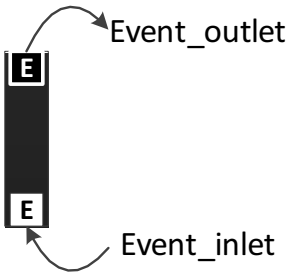


Fig. 1: Event Inlet and Outlet

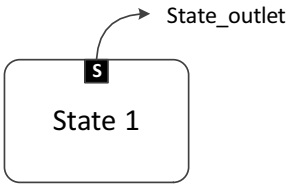


Fig. 2: State Outlet

II. Modeling elements of SEFTs

In this section, we describe the modeling elements of SEFTs, their semantics and constraints associated with them. We first list out the modeling elements defined in [2] for SEFTs and then describe the modeling elements that we added to SEFTs to be able to create them using a tool. The conceptual modeling elements are:

(a) Component which encompasses the behavior of the modeled component in the form of state charts.
(b) State charts that contain state changes facilitated via events. States and events are connected using temporal connections which are represented by arrows with unfilled arrowheads.
(c) Inports and Outports that allow propagation of failures in and out of components. These ports are further typed as 'state' or 'event' ports which are connected to other elements in the state chart or gates by causal connections represented by arrows with filled arrowheads.
(d) Gates are connected to one or more inputs leading to an output that combine states and events using boolean logic, priorities and may have parameters such as delays associated with them. As mentioned earlier, gates in SEFTs are not just boolean gates like the ones used in traditional FTs, but have internal state charts associated with them.
(e) Once an SEFT interface consisting of outports and inports has been created for a component type, it can be instantiated in other components of which it is a part. This can be done by dragging and dropping a component or by creating a proxy from the palette and then choosing the component type in the ESSaReL tool. Both methods result in creation of a component instance with its interface consisting of the inports and outports which we refer to as Proxy State Inlets/Outlets and Proxy Event Inlets/Outlets.

In addition to the above mentioned elements, we add a few modeling elements to the tool which are described below:

(a) Event inlet and Event outlet for triggered or triggering action of events (Fig. 1).
(b) State outlet for connecting outgoing causal edges from states (Fig.2).

A state does not have a state inlet as states cannot be triggered from other components, only events can be triggered which can cause the corresponding state transitions.

III. Design decisions for ESSaRel

As the use of Component Fault Trees increased in the industry and in research, it became clear that the first version of the tool [4] needed to be rewritten. The reason for this was that the existing version was not flexible enough. With the increasing use of CFTs in the academic/research field, new ideas were implemented in different tools and there was need to integrate ESSaRel with these other tools. For example, CFTs created in different front-ends such as Magicdraw [5] needed to be analyzed using ESSaRel. Also multiple back-ends such as Fault Tree+ and Zusim could be used for analysing CFTs. To address this issue, there is a common data model was introduced for both CFTs and SEFTs on which ESSaRel is based. This serves as an intermediate model between ESSaRel and other tools. In addition, for SEFTs, there is a common data model for DSPNs as well so that once an SEFT is translated to a DSPN, different backends capable of analysing DSPNs can be used. Fig. 3, 4 and 5 show a fire alarm system described in [4] modeled in the new version of ESSaRel. In the repository explorer of Fig. 3, we can see the three components: FireScenario (which represents the system under study), FireAlarmUnit and Watchdog. The FireAlarmUnit was modeled just once (Fig.3) but instantiated twice as FireAlarmUnit1 and FireAlarmUnit2 in the FireScenario component. Watchdog has been modeled (Fig.4) and instantiated once in FireScenario, its outport has been connected to inports of both instances of the FireAlarmUnit.

One can notice that there are no outports or inports in the palette in Fig.3. This is because the canvas that is visible is that of the realization of the FireAlarmUnit. There is another canvas for the interface (which has not been shown) whose palette contains the inports and outports. Once a port is added or deleted from the interface, ESSaRel automatically synchronizes the interface and the realization by making the corresponding change in the realization. This ensures consistency by making sure that the only way to add or remove ports in the realization is by modifying the interface. The reason for this kind of separation between the interface and realization is to allow a user to have the flexibility to choose between different realizations for a component. For example, the values at the outports of a component may come from an underlying SEFT or a MATLAB/Simulink model. The realizations for the Watchdog and the FireAlarmUnit are shown in Fig. 4 and 5 respectively.

In the remainder of this section, we describe the design features of the tool to make it easier for users to create syntactically correct SEFTs. Tables I and II below show the constraints associated with causal and temporal connections respectively. We have omitted those modeling elements that cannot be sources on the first column and those that cannot be targets on the first row in order to reduce the size of the tables. The entries with a check mark (✓) indicate the

 ISBN: 978-1-61208-301-8

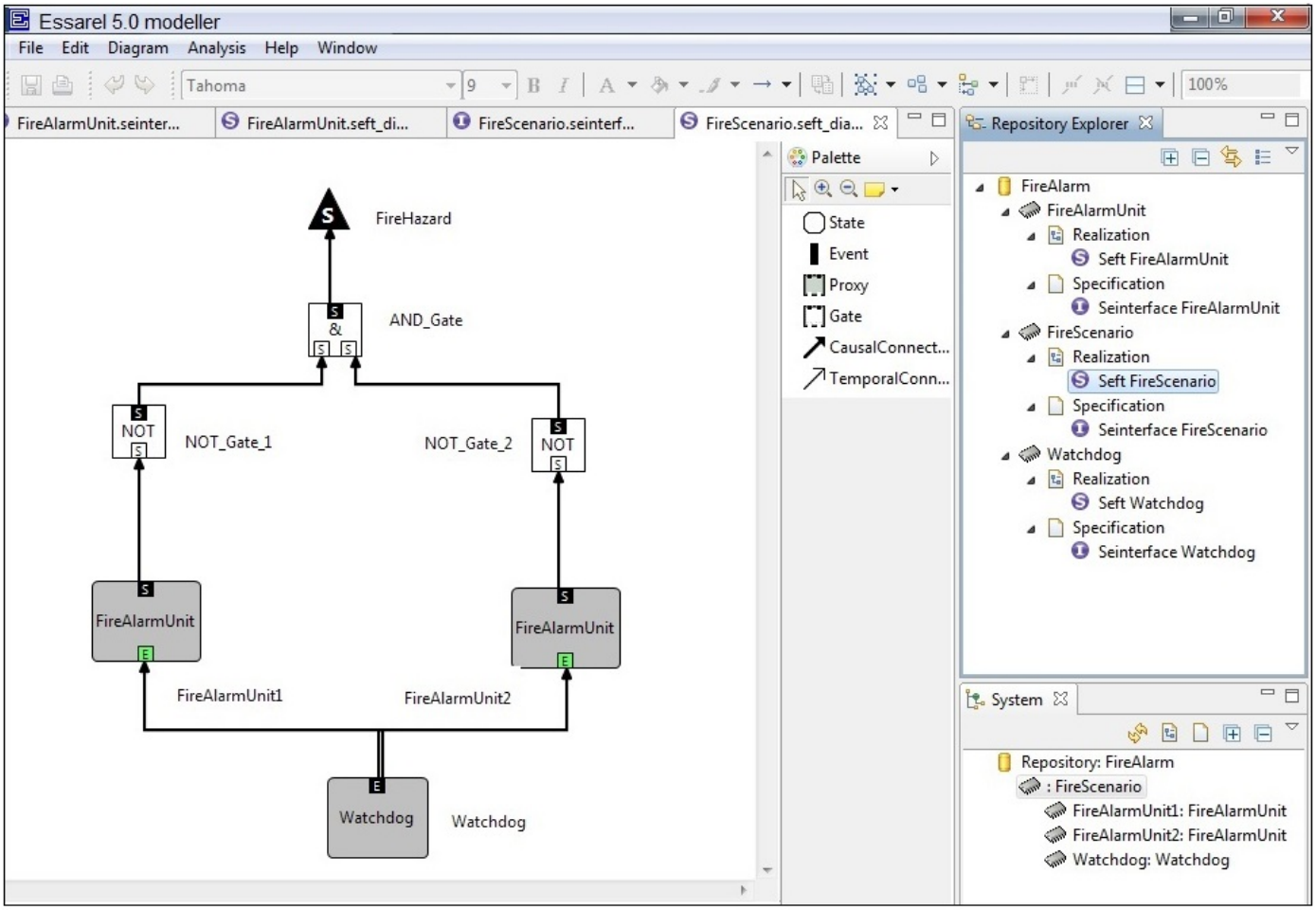


Fig. 3: Fire Scenario modeled in ESSaRel

valid connections and entries with a cross mark (✗) are prohibited. We can see that there are a large number of rules for connecting modeling elements which increase the chances for a user to unknowingly create wrong connections. ESSaRel prevents such errors by inhibiting the creation of prohibited connections.

It may be useful for users to note that only outgoing causal edges are allowed for states through state outlets, but states cannot have incoming causal edges. States can occur only by the triggering of a preceding event connected to the state through a temporal connection. This implies that states cannot be 'propagated'; only events can facilitate propagation by triggering events in other components. States, on the other hand can act as guards for an event so that the event can only be triggered when the guarding state is true. As mentioned in the previous section, this is the reason why a state does not have a state inlet.

From Table 2 for temporal connections, we can see that a state can only be connected to events and vice-versa, i.e., there is a strict alternation between states and events connected by temporal edges. A state can have two or more outgoing and incoming edges, but an event can have only one incoming and one outgoing edge.

Apart from the above constraints, the tool ensures that there is not more than one incoming causal edge for a target. This is necessary to ensure that there is no ambiguity in the cause of an event or state. The tool also ensures prevention of shallow cycles where a component references itself by preventing proxies of a component to be created in its own realization.

IV. Practical Modeling Tips for Users

Often, it can be confusing for users as to which is a good way to model a given scenario as the same scenario can be represented in multiple ways. For example, let us consider a situation where an intermediate failure can be represented as either the output of n-state-AND gate or as the output of the History-AND gate for events. In such a situation, it is important to understand well the difference between states and events. States have a persistent nature, but a component is also capable of changing its state. On the other hand, events do not have any temporal expansion and once they occur, the only way to record their occurrence is through the use of a History-AND gate. Therefore, in a situation where we want to model a scenario where two or more components are required to be present in a certain state for an event to occur, we can use the n-state-AND gate and when we want to model a scenario

 ISBN: 978-1-61208-301-8

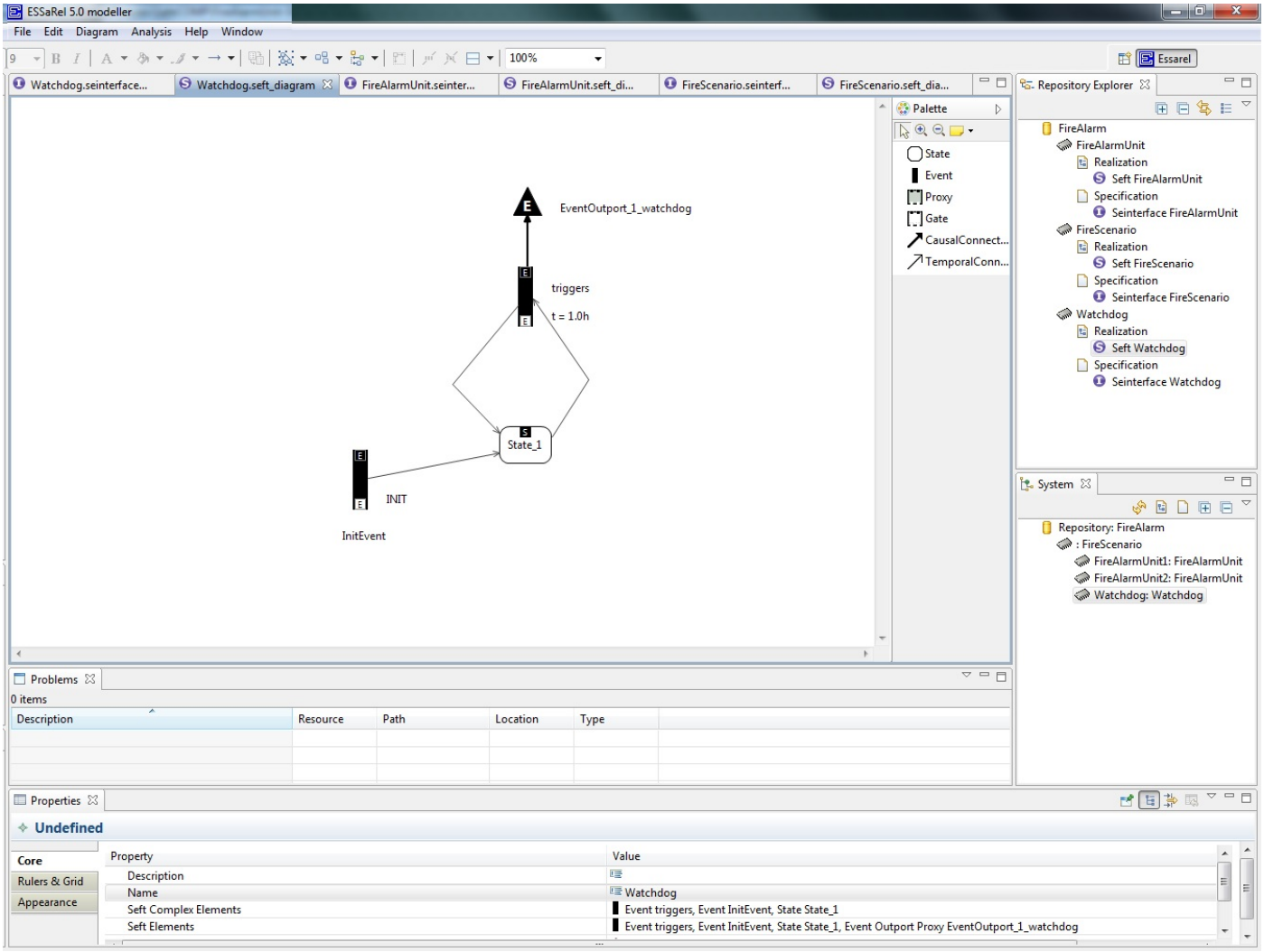


Fig. 4: Realization for Watchdog

where the occurrence of some events cause an intermediate event even when the components may no longer be in the states resulting from those events, we can use a History-AND gate.

As another example, let us consider the situation for modeling state chart interaction in SEFTs where there are two possibilities. The components can interact with each other via state ports or event ports. As input ports both possibilities have to be connected to events, either as a triggered event or as a guard function of a state. But we are of the opinion that the event based communication is better because triggered events are much more intuitive compared to guard functions. It is indeed possible to transform every state based communication into an event based one by using a Flip-Flop gate. To do this, the 'Set' port of a Flip-Flop gate has to be connected to all incoming transient arcs of the state and the Reset port with all outgoing arcs. But it is not advisable to do so unless necessary as the complexity of the transformed state-chart (into Petri Nets for analysis) increases and the advantages of a more intuitive SEFT are nullified. In this case a state-based communication is preferable.

A. *Nomenclature Scheme for SEFTs*

SEFTs, like fault trees, are constructed by humans who may give arbitrary names to failure ports. This may lead to miscalculations during quantitative or qualitative analysis. More information on consequences of wrong or ambiguous nomenclature has been documented in [6], [7], [8], [9]. Here, the authors recommend the use of two fields to construct FT event names:

(a) Component Name, which is the fully qualified name of the component given by the system decomposition hierarchy.
(b) Failure Mode, which describes the nature of the failure.

But since SEFTs are specialized FTs, the above two fields are not sufficient for unambiguous nomenclature of its events. Hence, to ensure unambiguity, we recommend to users to construct names with the following two additional fields along with the ones mentioned above:

(a) Environment Condition, which depends on the context in which the component is deployed.
(b) System Condition, which depends on the configuration

 ISBN: 978-1-61208-301-8

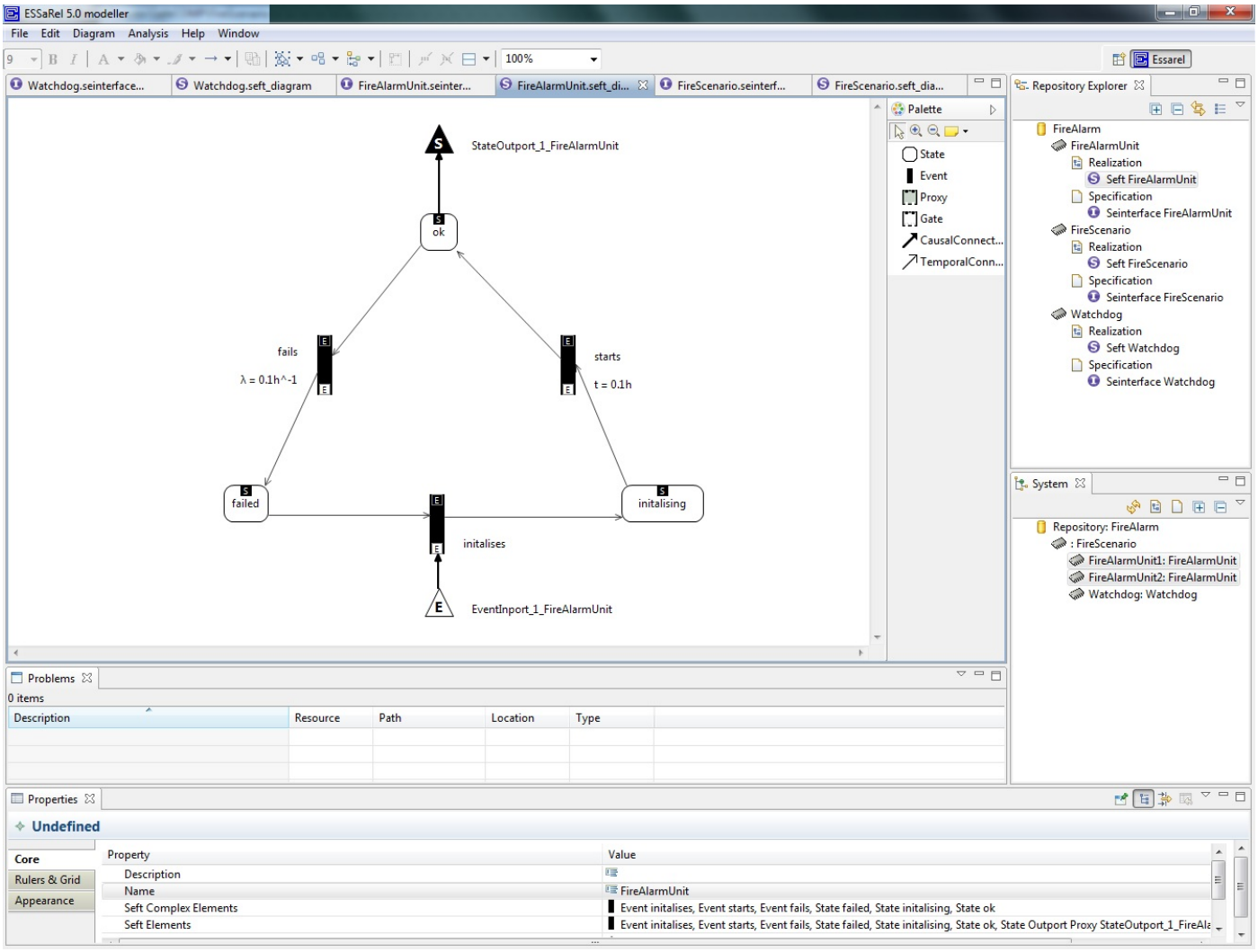


Fig. 5: Realization for Fire Alarm Unit

peculiar to the component itself. This field is useful to distinguish between two or more events that are sources for temporal edges leading to a common target state.

These fields help to distinguish names of failure modes of the same component the working conditions, which depends on the context of deployment and the configuration of the component. For more information on the description and usage of the above fields, users are suggested to read [10].

B. Analysis

This subsection explains the evaluation functions of TimeNET. SEFTs can be component-wise translated into eDSPNs. This operation transforms events into transitions and states into places where the initial state is expressed as the initial marking of the resulting net. Deterministic and exponentially distributed events can directly transfer into the equivalent Petri Net transitions. For triggered events, however, a pattern exists to connect different Petri Nets with each other. According to the gate dictionary [4], for every gate an equivalent eDSPN is available. By the usage of these transformation functions, a Petri Net can easily be built out of every SEFT. In addition to normal DSPNs, eDSPNs support probabilistic values as well as priorities to avoid conflicting situations. The probabilistic functions specify a value which stands for the likelihood that a transition fires after its activation. Further it provides textual elements such as the so called performance measures. These measures represent the asked questions such as "What is the probability for a certain marking of a specific place?" A special grammar has to be used for defining the measures which can be found in [3]. An example of such a performance measure for the probability that place P1 contains more than one token is P{#1>1}.

In TimeNET, there are different evaluation methods for the analyzing these performance measures. The categories are divided into (1) analysis and (2) simulation techniques. To run an evaluation, at least one performance measure has to be specified. The first evaluation category gives an exact numerical result by computation of the reachability graph. Therefore, the reachability graph has to be finite. In case of a transformed SEFT this precondition is always fulfilled because all converted Petri Nets are bounded. This type of

 ISBN: 978-1-61208-301-8

TABLE I: Constraints for Causal Connections

Source \ Target	Proxy State Outport	Proxy Event Outport	State Inport	Event Inport	Event Inlet	Gate State Inlet	Gate Event Inlet
Proxy State Inport	✓	✗	✓	✗	✓	✓	✗
Proxy Event Inport	✗	✓	✗	✓	✓	✗	✓
State Outport	✓	✗	✓	✗	✓	✓	✗
Event Outport	✗	✓	✗	✓	✓	✗	✓
Event Outlet	✗	✓	✗	✓	✗	✗	✗
Gate State Outlet	✓	✗	✓	✗	✗	✓	✗
Gate Event Outlet	✗	✓	✗	✓	✓	✗	✓

TABLE II: Constraints for Temporal Connections

Source \ Target	State	Event
State	✗	✓ (incoming edges = 1)
Event	✓ (outgoing edges = 1)	✗

evaluation technique can be further subdivided into transient and stationary analysis, also called steady-state analysis. The transient evaluation can analyze the net from the point of initial marking during a given time period. The result can be shown as a curve which represents the defined measure for the complete interval or as a point which represents the measure only at the end of the interval. Steady-state evaluation is able to find a result without specifying such a time period. It could be seen as a transient analysis with an infinite time period. For getting a steady-state result it is necessary that the eDSPN is designed without deadlocks. This means that the reachability graph may not have any nodes where no transition is enabled. An additional precondition for the analysis of eDSPNs in TimeNET is the existence of at most one deterministic transition. If this precondition is fulfilled then these evaluation methods can be used to calculate the result for the measures depending on the required time period (finite or infinite).

The second category of evaluation techniques is the simulation methods. These methods estimate the measures by the use of a modified Monte Carlo simulation and can also subdivided into a stationary and a transient method. Because of the inaccuracy, simulation should only be used if there is more than one deterministic transition in the eDSPN. Apart from that, simulation methods need to fulfill the same preconditions as analysis methods. More information can be found in [3].

V. Conclusion and Future Work

ESSaRel has been designed to be a user-friendly tool that aids engineers to create syntactically correct SEFT models. Based on our experience, we have also described some aspects of SEFTs to provide tips to a user to build semantically correct SEFT models. It is possible to perform quantitative analysis on SEFTs, but it is not possible to perform qualitative analysis on SEFTs as not much research has been carried out with respect to this aspect. We intend to integrate qualitative analysis such as minimal cut set generation. We would like to integrate mechanisms for qualitative analysis when they become available. In the future, we would like to extend the functionality of ESSaRel to be able to directly display analysis results for SEFTs directly in ESSaRel without having to manually run the TimeNET tool.

Acknowledgment

This work was funded by the VIERforES project supported by BMBF, Germany.

References

[1] B. Kaiser, P. Liggesmeyer, and O. Mäckel, "A new component concept for fault trees," in *SCS*, 2003, pp. 37–46.

[2] B. Kaiser, "State event fault trees: a safety and reliability analysis technique for software controlled systems," Ph.D. dissertation, Technische Universität Kaiserslautern, 2006.

[3] A. Zimmermann and M. Knoke, "TimeNET 4.0 user manual," Technische Universität Berlin, Faculty of EE&CS, Tech. Rep. 2007-13, 2007.

[4] B. Kaiser and C. Gramlich, "State-event-fault-trees - a safety analysis model for software controlled systems," in *SAFECOMP*, 2004, pp. 195–209.

[5] R. Adler, D. Domis, K. Höfig, S. Kemmann, T. Kuhn, J.-P. Schwinn, and M. Trapp, "Integration of component fault trees into the uml," in *MoDELS Workshops*, 2010, pp. 312–327.

[6] C. A. Ericson, "Fault tree analysis by design," in *Proceedings of The 16th International System Safety Conference*, 1998.

[7] A. Long, "Variants of classical cutset characterization," in *21st International System Safety Conference*, 2003.

[8] Y. Papadopoulos and U. Petersen, "Combining ship machinery system design and first principle safety analysis," in *IMDC 03, 8th Intl Marine Design Conf*, Athens, May 2003, pp. 1:415–426.

[9] M. Stamatelatos, W. Vesely, J. Dugan, J. Fragola, J. M. III, and J. Railsback, *Fault Tree Handbook with Aerospace Applications*. NASA, 2002.

[10] K. Jamboti, C. Gomez, O. Mackel, and P. Liggesmeyer, "Improved nomenclature schemes for component fault trees and state/event fault trees," in *Annual European Safety and Reliability (ESREL) Conference(In-Press)*, 2013.

 ISBN: 978-1-61208-301-8