DEPEND 2015

The Eighth International Conference on Dependability

ISBN: 978-1-61208-429-9

August 23 - 28, 2015

Venice, Italy

DEPEND 2015 Editors

Pascal Lorenz, University of Haute-Alsace, France

Petre Dini, Concordia University, Canada / China Space Agency, China

DEPEND 2015

Foreword

The Eighth International Conference on Dependability (DEPEND 2015), held between August 23-28, 2015 in Venice, Italy, 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 2015 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 2015. 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 2015 organizing committee for their help in handling the logistics and for their work to make this professional meeting a success.

We hope that DEPEND 2015 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 Venice provided a pleasant environment during the conference and everyone saved some time for exploring this beautiful city.

DEPEND 2015 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

DEPEND 2015 Industry Liaison Chairs
Piyi Yang, Wonders Information Co., Ltd., China
Timothy Tsai, Hitachi Global Storage Technologies, USA

DEPEND 2015 Research/Industry Chair
Michiaki Tatsubori, IBM Research Tokyo, Japan

DEPEND 2015 Special Area Chairs
Cross-layers dependability
Szu-Chi Wang, National Ilan University, Taiwan
Big Data and dependability
Cesario Di Sarno, University of Naples Parthenope, Italy
Empirical assessments
Marcello Cinque, University of Naples Federico II, Italy
Security and Trust
Syed Naqvi, CETIC, Belgium

DEPEND 2015

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

DEPEND 2015 Industry Liaison Chairs

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

DEPEND 2015 Research/Industry Chair

Michiaki Tatsubori, IBM Research Tokyo, Japan

DEPEND 2015 Special Area Chairs

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

Big Data and dependability
Cesario Di Sarno, University of Naples Parthenope, Italy

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

Security and Trust
Syed Naqvi, Birmingham City University, United Kingdom

DEPEND 2015 Technical Program Committee

Habtamu Abie, Norwegian Computing Centre, Norway
Don Adjeroh, West Virginia University, USA
Muhammad Afzaal, National University of Computer and Emerging Sciences, Pakistan
Joxe Inaxio Aizpurua Unanue, University of Strathclyde, UK
Murali Annavaram, University of Southern California, USA
Luciana Arantes, Université Pierre et Marie Curie (Paris 6), France
Afonso Araújo Neto, University of Coimbra, Portugal
José Enrique Armendáriz-Iñigo, Universidad Pública de Navarra, Spain
Radu F. Babiceanu, Embry-Riddle Aeronautical University, USA
Ian Bayley, Oxford Brookes University, U.K.
Siegfried Benkner, University of Vienna, Austria
Jorge Bernal Bernabé, University of Murcia, Spain

Il Yen, University of Texas at Dallas, U.S.A
Hee Yong Youn, Sungkyunkwan University, Korea

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

Measuring Application Server Availability on the NorNet Core

Sune Jakobsson
Department of Telematics
NTNU
Trondheim, Norway
Email: sune.jakobsson@telenor.com

***Abstract*— This paper investigates the availability of applications servers running on the NorNet Core test-bed. NorNet Core is the world's first, open, large-scale Internet test-bed for multi-homed systems and applications. Particularly, it is currently used for research on topics like multi-path transport and resilience. The NorNet Core test-bed provides access to worldwide distributed nodes, connected with multiple interfaces over a set of ISPs (Internet Service Providers), providing independent transport paths between them. Each node has a set of programmable nodes that can be used for network experiments. This paper describes a practical approach to assess how suitable this test-bed is for distributed computing, and application servers.**

Keywords- Test-bed; Java virtual machines; application servers; availability; tunnelling.

I. INTRODUCTION

This paper addresses the behaviour and availability of distributed computing resources in a "virtual" network built on top of academic and commercial networks, afterwards referred as the NorNet test-bed [13]. In a previous paper discussing the availability of web servers in commercial settings using providers (e.g. Amazon, Google and other providers) hosting the computing resources that use the Internet as the transport network [12]. In such setting, you as a customer have little or no control over the computing resources. It is hard to assess to what extent the computing resources are shared or virtualized, but one can assume that the Internet itself is a reasonably stable platform for transport. However, as a consumer of the computing resources one has little or no control of the instance of deployment. By this we mean that the commercial providers do not disclose any or very little information regarding their infrastructure. In the NorNet Core case, one has near complete control and information over the computing resources but limited control over the point-to-point tunnels running between the sites.

The objective here is to assess the behaviour and availability of web servers running in the NorNet Core network and the transport of packets between sites. It describes a series of simple experiments at application level, i.e., invocation of Web servers and how to capture their continuous operation and long term behaviour.

In Section II, we describe the infrastructure in detail and how the experiments were carried out. The goal of these measurements was to detect changes in the test-bed over periods of days or weeks, due to issues that can be traced back to the communication or the software (SW) running at the sites and how these issues affect application servers running on the test-bed. The NorNet Core test-bed was continually updated and upgraded and the packet route the tunnels use was entirely up to the ISP, so there was a number of factors that impacted the availability. Section III addresses the details of the monitoring, and Section IV highlights a subset of the results. Section V provides recommendations.

II. THE NORNET CORE TEST-BED

A. Test-bed structure

As of writing March 2015, the NorNet Core test-bed [1] is deployed on 19 sites physically distributed across the world and interconnected with tunnels over 14 different ISP's networks. The majority of the sites are at the major universities in Norway, at Simula A/S in Oslo and the rest are at universities in Sweden, Germany, China, Korea and USA. The 14 ISP's provide connectivity across the sites, so that the majority of the sites are connected using tunnels with more than 1 ISP involved.

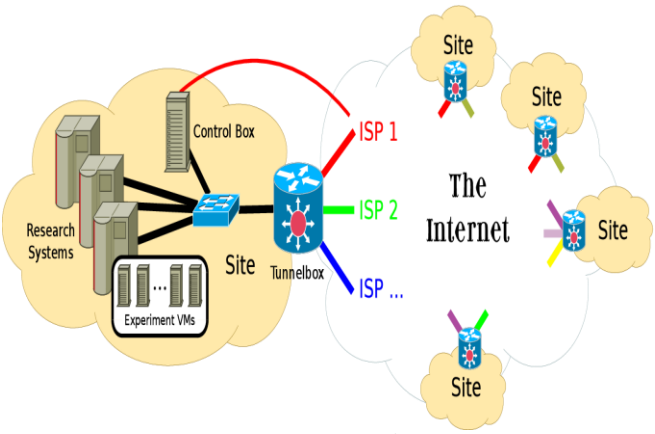


Figure 1. Overall structure of Nornet Core

Each site has a set of research systems running virtual machines for experiments, a control box for management functions, and a tunnel box that terminates the tunnel end-points between the sites. The NorNet Core runs its own domain name service (DNS), and the tunnels provide both IPv4 and IPv6 connectivity between the sites. The tunnels provide site to site connectivity over academic and commercial IP networks. The overall structure of the NorNet Core is illustrated in Figure 1.
The red line from the control box is the connection to the central management system in Oslo.

Each site contains a set of physical servers that host individual virtual machines running instances of Planet lab software [3] for managing the sites. The virtual machines

 ISBN: 978-1-61208-429-9

(VM) run the Fedora version 18 operating system [4], and connect to all available VPN tunnels at the site through the tunnel-box, and researchers can use them for multi-homed experiments as needed. Each site contains a number of VM instances, and they are all connected to all ISP's at the site. The experimenters are free to install SW on the VM's as needed. These VM instances are referred as slivers in the NorNet terminology. Please note that the term sliver in this paper refers to a running VM at a site. The term is also used by Fedora, but with a different meaning in their setting. The test-bed is configured so that the users get global access to all nodes, and they are able to do experiments on each node as needed, by accessing each virtual machine on an instance by instance basis. This allows individual users to get assigned VM's with private IP addresses, and do not need to consider sharing network interfaces with other users. There are some restrictions on what access rights a user is assigned to the operating systems on each site, and access to all operating system instances is done through the central site at Simula A/S in Oslo, Norway. These restrictions include tunnel configuration, and the underlying management of the research systems at a site. The entire NorNet Core infrastructure is managed from Simula Research Laboratory and their technical staff in Oslo, Norway.

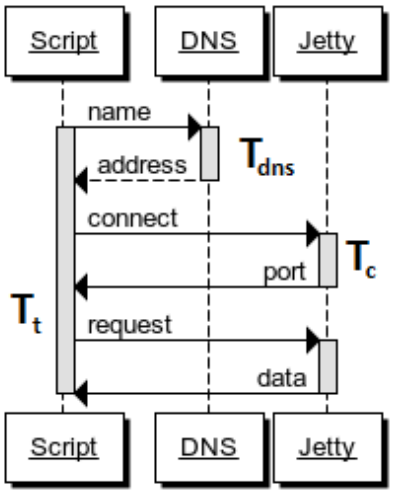


Figure 2. Invocation time sequence for the measurement script

B. The measurement setup

The NorNet Core test-bed at Simula A/S, has kindly provided a central node in Oslo for measurement purposes, which has direct access to the network interface, and is able to do packet capture on the wire, so that the behaviour of the network can be captured and studied in retrospect. The measurements are done on HTTP calls issued from the central node in Oslo, where the calls are issued at fixed intervals to a select set of sites, using all tunnels, and thereby using the infrastructure provided by all involved ISP's. Since this is also the node that manages all other tunnels and nodes, and has other usages within Simula A/S, it is fair to assume that any operational issues are observed and rectifier within reasonable time. This central measurement node runs the Ubuntu operating system and is directly connected to four ISP's. The HTTP calls from the measurement node are issued in a shell script using the `curl` command [7] and `crontab` [8] to schedule the commands every minute, and the results are captured in a log file, as shown if Figure 2. The results from a day without down-time and invocation errors on a particular ISP and site are shown in Figure 3, where the status (200 OK) is shown in a horizontal pink line and the black lines are the DNS lookup times, the blue lines are the connection setup times, and the green lines are the total invocation time. However some of the invocations might exceed the time constraints, but this depends on the actual application, and its requirements. The time shown in green shows all the time elements, DNS lookup, connection time and data transfer time added together.

C. Experiment VM at sites

For the availability experiment, an instance of an Embedded Jetty Server [5] runs and listening to HTTP requests on all interfaces. The HTTP requests are issued with the `curl` command, and scheduled with `crontab`. The invocations are scheduled at one minute intervals, and each ISP tunnel runs 1440 measurements per day. Since the network is virtualized each sliver has its own IP addresses on each of the ISP tunnels. The HTTP requests are tagged with time-stamps and also logged locally on each Web server.

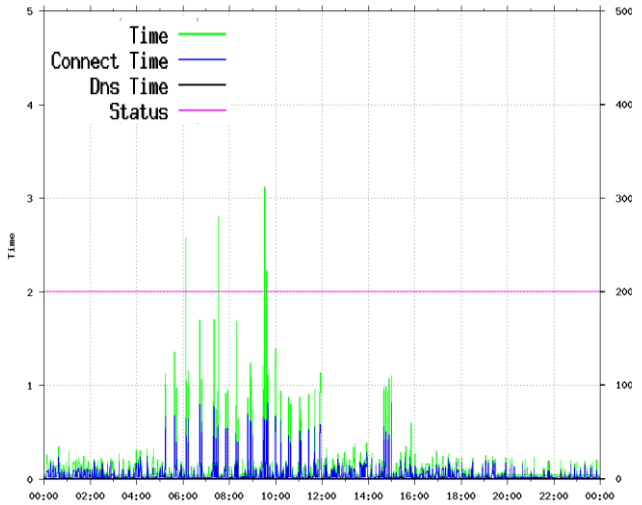


Figure 3. Invocation times (seconds) and status for 24 hours

The Web Server logs locally the incoming request and their unique invocation tags, and responds with a short response containing the amount of available memory on its Java instance. The triplet of IP source and destination addresses and time-stamp provides a unique identifier in the local logs. This also eases the identification of the packets captured on the wire between servers and clients, and makes it possible to observe the network behaviour at the packet level, with the packet sniffing tools.

 ISBN: 978-1-61208-429-9

III. MONITORING OF THE TEST-BED

There are multiple issues that one wants to observe in order to determine the stability of a test-bed. One is the nodes themselves, their ability to communicate, and what changes over time are observable. Given that the test-bed should be able to run Internet scale experiments, observing them from an application point of view will give a real life picture of its abilities. The setup of the NorNet Core needs a set of experiments carried out in parallel in order to pinpoint possible issues that can occur, whether they occur on a single sliver, on an entire site, or on NorNet Core as a whole.

The measuring node runs two distinct sets of measurements in parallel where the first set runs towards physically distributed nodes and the other set runs towards the slivers residing on one physical location. The reasoning behind this is to be able to detect internal issues on a node, i.e., if there are issues that can be traced back to the tunnel-box and the installation at that site vs. general operation issues in the test-bed as a whole. Since all requests are issued on all ISP's available for transport at that particular site, one can determine if an issue is related to a site or to transport. Each tunnel on each ISP can then be plotted every day as a graph shown in Figure 3, and one has a visual overview of the behaviour over time from day to day.

The "`curl`" command gives the connection time, DNS lookup time and connection time for each invocation. In addition each invocation is tagged with a time-stamp so that it is possible to explore the network behaviour at the packet level using tools like "`Wireshark`" [9] to do retro inspection of unusual or odd behaviour in the HTTP communication between the sites. Unfortunately the packet capture is only available at the monitoring node. In addition the local logs are available at each sliver that is invoked.

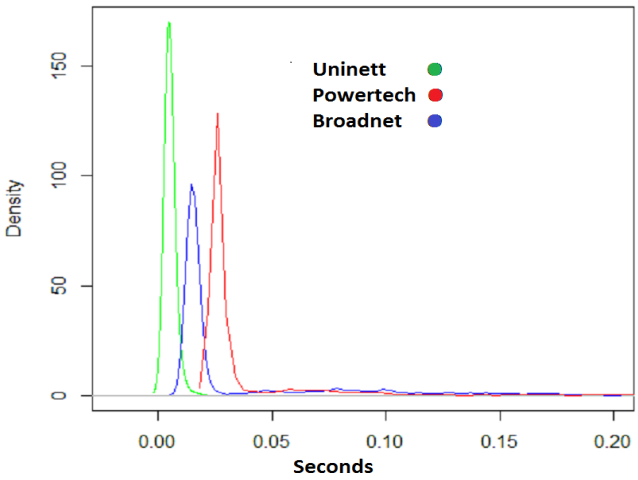


Figure 4. Density plot of connection times

With the redundant transport between the nodes it is easy to determine the overall condition of an individual tunnel and an individual sliver between the measuring node and the sliver. By automating the plot generation, daily plots are easily generated like the one shown in Figure 3. This, however, results in great numbers of plots and checking them all for abnormalities can be a daunting task. By setting limits on the invocation time T_t on tunnels or slivers with issues are easily identified and can then be inspected further. By visually comparing plots between different physical sites, it is straight forward to identify global issues or particular issues only manifesting themselves at one site or on one single tunnel to that site.

It is also desirable to assess the network characteristics of the tunnels on a daily basis by a statistical analysis. Since the tunnels are tunnelled over Internet or some local transport, their characteristics varies over time. The connection times T_c for a particular tunnel and sliver pair, are shown as a density plot in Figure 4 or as a visual plot of an empirical cumulative distribution as shown in Figure 5. Given the shape of the distributions and the number of samples per day, the Kolmogorov-Smirnov test [9] is chosen to be the most suitable test to compare the daily connection time data.

The daily connection time distribution can be determined for each tunnel and sliver pair and the result gives an indication if there are changes in the communication between the measuring node and a particular sliver. Uninett is the research network in Norway, whereas Powertech and Broadcom are commercial ISP providers in Norway.

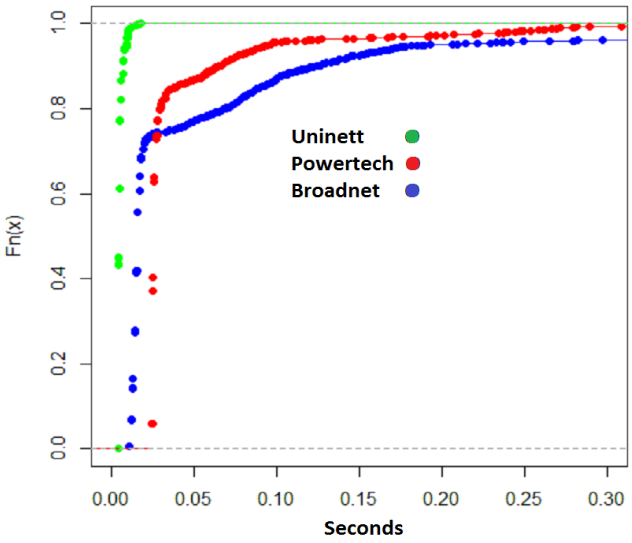


Figure 5. Empirical Cumulative Distribution Function

IV. RESULTS

By assessing daily measurements first at HTTP invocation level, and by defining an acceptable maximum invocation time, depending on the application and the usage, and comparing connection-time distributions and slivers memory usage patterns, enables detection of changes in all involved parts of the test-bed. By overlying the daily HTTP invocation and status plots one can identify "global" issues affecting the entire test-bed, and as well as "local" issues affecting one site, or one ISP tunnel between the measuring node and the site or sliver. By "global" issues we mean events that impact the entire NorNet Core network, like the DNS or the management functions, where as "local" issues

 ISBN: 978-1-61208-429-9

are issues that affect only one site. Even though there are variations the connection-time distribution is stable, unless there is a change in the IP packet route or a change in the tunnel-box SW. By viewing the density plots (Figure 4) or empirical cumulative distribution functions (ECDP) (Figure 5) on tunnel and sliver pairs, the tunnels repeat the same plots. In addition the Kolmogorov-Smirnov tests have been run to show that the days without changes or downtime give the same distribution.

The memory usage on the slivers has also been checked, and the slivers do not appear to be disturbed by other processes on each server. They all show a regular pattern in the amount of free available memory, and are hence not disturbed or affected by external factors.

The Web Server SW and the scripts used to run the experiments are all available at github [6].

V. CONCLUSIONS

The NorNet test-bed provides a multi-homed environment for large scale Internet experiments, but it has unfortunately focused the technical aspects of such a test-bed, and primarily at the transport level between the slivers. Most of the experiments published address multi-home transport and their protocols, and are not addressing Internet style client-server usage [2].

The physical distribution of sites adds some transport time between them, and occasionally the routing changes between sites add a constant to the transport time.

The NorNet Core test-bed provides monitoring tools with graphical interfaces. However, this does not give a detailed picture of the communication between the sites nor the status or quality of the tunnels between the sites. When a site goes off-line there is limited support for bringing the site back on-line other than contacting the personnel at the site. This has some grave implications on availability if parts of the test-bed run into issues or go down outside office hours or vacation times. Being a research network NorNet Core does not provide a service level agreement (SLA) for their users, so you do not get your money back when there are failures [11]. To be able plan and carry out long term experiments it is necessary with more than only best effort guarantees on a test-bed, to provide repeatable experiments.

The NorNet Core should add some rudimental monitoring SW for each node and each tunnel, and provide this information on the NorNet Core web page. This information could also be used internally at Simula A/S to alert the personnel in charge to quicker respond to failures or errors that are bound to happen at some point. A SLA for the users of NorNet Core could be beneficial for all parties involved.

ACKNOWLEDGMENT

I would like to thank Professor Rolv Bræk and Professor Bjarne 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] NorNet Core, https://www.nntb.no/pub/nornet-configuration/NorNetCore-Sites.html , Last seen June 2015

[2] NorNet publications, https://www.nntb.no/publications/, Last seen June 2015

[3] Planet lab, https://www.planet-lab.org/ , Last seen June 2015

[4] Fedora 18 (Spherical Cow), http://docs.fedoraproject.org/en-US/Fedora/18/html/Release_Notes/index.html , Seen June 2015

[5] Jetty, application server, http://eclipse.org/jetty/ , Seen June 2015

[6] GitHub, code base, https://github.com/sunejak/EmbeddedJetty , Seen June 2015

[7] Curl, tool, http://curl.haxx.se/docs/manual.html , Seen June 2015

[8] Crontab, tool, http://www.unix.com/man-page/linux/5/crontab/ , Seen June 2015

[9] Wireshark, tool, https://www.wireshark.org , Seen June 2015

[10] Vito Ricci, "Fitting distributions with R", http://cran.r-project.org/doc/contrib/Ricci-distributions-en.pdf , Seen June 2015

[11] Brian Harry, "How do you measure quality of service?", http://blogs.msdn.com/b/bharry/archive/2013/10/14/how-do-you-measure-quality-of-a-service.aspx , Seen June 2015

Article in conference proceedings:

[12] Sune Jakobsson, "Estimation of Performance and Availability of Cloud Application Servers through External Clients", in DEPEND 2013, 6th International Conference on Dependability, Pages 1-5, ISBN: 978-1-61208-301-8

[13] Thomas Dreibholz, Jarle Bjørgeengen, and Jonas Werme: "Monitoring and Maintaining the Infrastructure of the NorNet Testbed for Multi-Homed Systems", in 5th International Workshop on Protocols and Applications with Multi-Homing Support (PAMS) 2015, Pages 611–616, ISBN 978-1-4799-1775-4, DOI 10.1109/WAINA.2015.76

 ISBN: 978-1-61208-429-9

Beyond Integration Readiness Level (IRL): A Multi-Dimensional Framework to Facilitate the Integation of System of Systems

Clarence L. Eder
Systems Engineering PhD Candidate
George Washington University
Washington DC, United States of America
Email: edercl@gwu.edu

***Abstract*—Integration Readiness Level (IRL) can be an effective systems engineering tool to facilitate integration of systems. With further research and the use of systems architecture methodology, IRL principles could enhance the use of systems integration in Department of Defense (DoD) Acquisitions. DoD space systems are great examples of system of systems, and analyzing space systems' integration issues will help identify critical integration variables. Integration data will be collected to develop a framework to enhance IRL notional definitions that will help improve space systems' availability and dependability.**

Keywords-Integration Readiness Level (IRL); Department of Defense (DoD) Acquisitions; Technology Readiness Level (TRL).

I. INTRODUCTION

Integration Readiness Level (IRL) was introduced to help understand the maturity of integrating one system to another [1]. The need to expand the use of IRL is increasingly becoming more relevant in the United States' Department of Defense (DoD) Acquisitions as programs try to acquire systems with the intent to have multiple capabilities and interfaces.

Throughout the years, DoD has continuously reduced the budget for weapon systems acquisitions. DoD Acquisitions implemented several systems engineering processes and tools to help meet budgetary requirements and still produce the best weapon systems available. The budget reduction along with the need to expedite the deployment of capabilities into operations trigger the drive to improve these processes and tools that program managers can depend on when making program decisions. In order to make decisions about a system and the technology available for the system, DoD Acquisitions adopted the use of Technology Readiness Level (TRL) in 2002 [2]. TRL provides close to a quantitative measure for explaining the maturity of a system based on the technology used for that system.

To further the use of TRL, IRL was introduced as an integration tool to complement TRL (Figure 1). IRL was developed to align with the TRL definitions, but it was never officially implemented by DoD to help with integration assessment. Other readiness levels such as System Readiness Level (SRL) and Test Readiness Level were also introduced but not officially recognized by DoD Acquisitions. Although not implemented, the use of IRL could become a necessary tool to help reduce integration risks of complex systems. Integrating system of systems are becoming more complex and the current definitions of IRL do not allow it to be independent of the TRL process, which could be one reason why IRL is heavily scrutinized in current systems engineering literature.

Lvl	TRL	IRL
1	Basic principles observed and reported	An interface between technologies has been identified with sufficient detail to allow characterization of the relationship
2	Technology concept and/or application formulated	There is some level of specificity to characterize the interaction between technologies through their interface
3	Analytical and experimental critical function and/or characteristic proof of concept	There is compatibility between technologies to orderly and efficiently integrate and interact
4	Component and/or breadboard validation in laboratory environment	There is sufficient detail in the quality and assurance of the integration between technologies
5	Component and/or breadboard validation in relevant environment	There is sufficient control between technologies necessary to establish, manage, and terminate the integration
6	System/subsystem model demonstration in relevant environment	The integrating technologies can accept, translate, and structure information for its intended application
7	System prototype demonstration in relevant environment	The integration of technologies has been verified and validated with sufficient detail to be actionable
8	Actual system completed and qualified through test and demonstration	Actual integration completed and mission qualified through test and demonstration in the system environment
9	Integration is mission proven through successful mission operations	Execute a support program that meets operational support performance requirements and sustains the system in the most cost-effective manner over its total life cycle

Figure 1. IRL and TRL Levels Defined [1]

II. THEORY

IRL can be an effective systems integration assessment tool and given the right multi-dimensional framework, it can facilitate the integration of system of systems. Utilizing other integration variables and expanding the current notional definitions of IRL can significantly impact the assessment of integration of system of systems. IRL was also proposed as an intermediate step by making it part of a matrix function with TRL in order to determine the SRL [2].

 ISBN: 978-1-61208-429-9

When IRL is used as a function of SRL, IRL could be overlooked from being a significant independent assessment value, and the IRL level may be influenced by what is needed as the SRL value. There are others who determine integration readiness can be assessed as part of DoD Acquisition's Technology Readiness Assessment (TRA) process, which is the official process to determine TRL score, but this process does not capture the purpose of integration. It is important to understand that a system with mature technology does not automatically equate to having a high IRL when interfacing with another system with mature technology. The current high-level definition given to IRL levels from 1 to 9 allows room for different interpretations when working with complex systems.

DoD space systems continue to provide examples of complex system of systems. With very limited opportunities to do operational tests and analyses for satellite systems and rocket launches, space systems provide a platform to incorporate the latest technologies and processes to attain successful operational systems. IRL can be used to assess the integration of these systems given a rigorous process that account for other variables. An assessment based on the current definition, which allows subjectivity that may be misinterpreted, will not work with current space systems.

A research is being performed to show the effectiveness of IRL in facilitating integration of system of systems. The research will focus on understanding the integration points with additional critical variables, and focus on the development of a systems architecture that will provide the framework to explain enhanced IRL levels. A systems architecture will be used as the methodology to prove the effectiveness of a newly defined IRL process.

III. GOALS/RESULTS

The goal is to expand beyond the IRL notional identified levels using architectural framework and assessed integration variables. To determine the integration variables, the research will focus on understanding the integration issues of six major DoD space systems. The data will be collected from the following family of systems: 1) Advanced Extremely High Frequency (AEHF) satellite; 2) Evolved Expendable Launch Vehicle (EELV); 3) Global Positioning Satellite (GPS); 4) National Polar-Orbiting Observing Satellite System (NPOESS); 5) Space Based Infrared Systems (SBIRS); and 6) Wideband Global SATCOM (WGS). The research will focus on integration issues from 1999 to 2014, and the data will be analyzed to understand the overall impact on capability, schedule, and cost. The focus of the integration issues will be at the space segment integration points (Figure 2) along with the subsystems integrated into each of the space segment.

The data collected will be used to construct an architectural framework and to determine weights for each identified variable. The framework and weighted variables will determine an objective IRL level. Initial integration variables that are being considered include: 1) Schedule (need date, allowed timeline to integrate); 2) Resources (Funding, Personnel, Available tools); 3) Processes (Documented approach, Binding Agreements, Testing); 4) Policies (Directives, Guidance); 5) Communication (Documentation, Semantics, Expectations); and 6) Risks (Cost, Schedule, Technical).

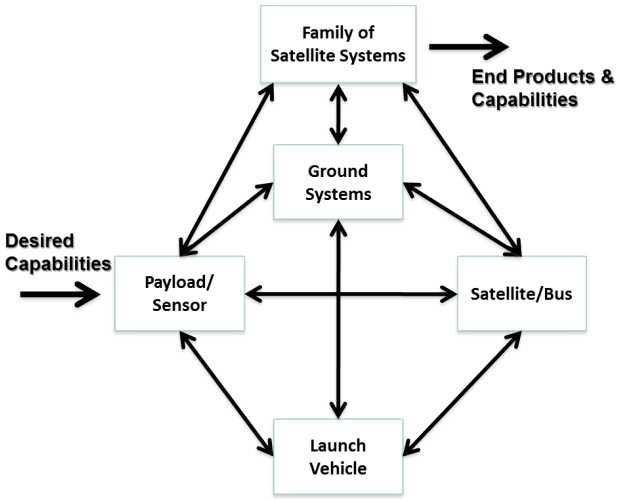


Figure 2. Major DoD Space Systems Integration Points

IV. CONCLUSION

The data will be validated through systems architecture application of the integration activities for all six space systems. The data collected will also be manipulated through regression analyses to determine possible trends that can support or object to the theory being researched. The systems architecture methodology will help scope the data collected and help facilitate the use of critical integration variables into relevant products that can be used to support overall program decisions and improve system availability and dependability.

The expected result is to have a list of integration issues and an understanding of how those issues impacted the system delivery through time, level of capability, and schedule. This will help identify attributes that will be used as variables for systems integration. Although the current DoD process of deploying space system capabilities for operational use does not require assessment of integration maturity, the result of this research should help quantify an integration tool that can further the use of IRL principles. Thus, making it very useful for stakeholders' decisions. With further research, using IRL with additional variables applied into a multidimensional architectural framework will provide a systems engineering quantitative tool that can enhance the facilitation of integrating system of systems.

V. REFERENCES

[1] Sauser, B., Gove, R., Forbes, E., Ramirez-Marquez, J. (2010). Integration maturity metrics: Development of an integration readiness level. *Information. Knowledge. Systems Management.*, v 9, n 1, p 17-46

[2] Electronic Publications DoD Instruction 5000.02-R

[3] McConkie, E., Mazzuchi, T., Sarkani, S., Marchette, D. (2013). Mathematical properties of System Readiness Levels. *Systems Engineering*, v 16, n 4, p 391-400.

 ISBN: 978-1-61208-429-9

On Handling Redundancy for Failure Log Analysis of Cluster Systems

Nentawe Gurumdimma*
Arshad Jhumka† and
Maria Liakata‡

Department of Computer Science
University of Warwick
Coventry, CV4 7AL UK

Email: *N.Y.Gurumdimma@warwick.ac.uk
†arshad@dcs.warwick.ac.uk
‡M.Liakata@warwick.ac.uk

Edward Chuah§ and
James Browne¶

Department of Computer Science
University of Texas - Austin

Email: §chuah@acm.com
¶J.Browne@utexas.com

***Abstract*—System event logs contain information that capture the sequence of events occurring in the system. They are often the primary source of information from large-scale distributed systems, such as cluster systems, which enable system administrators to determine the causes and detect system failures. Due to the complex interactions between the system hardware and software components, the system event logs are typically huge in size, comprising streams of interleaved log messages. However, only a small fraction of those log messages are relevant for analysis. We thus develop a *novel, generic* log compression or filtering (i.e., redundancy removal) technique to address this problem. We apply the technique over three different log files obtained from two different production systems and validate the technique through the application of an unsupervised failure detection approach. Our results are positive: (i) our technique achieves good compression, (ii) log analysis yields better results for our filtering method than normal approach.**

***Keywords-Cluster Log Data; Unsupervised learning; Compression; Levenshtein distance; filtering*.**

I. INTRODUCTION

The size and complexity of computer systems required for computationally-heavy jobs such as scientific computations is increasing and failures are expected to be the norm rather than exceptions. The unscheduled downtime of such large production computer systems carries huge costs: (i) applications running on them have to be executed again, potentially requiring hours of re-execution, (ii) checkpointing has to be performed regularly and (iii) lots of effort is required to find and fix the causes of the downtime. These systems generate a large amount of data, typically in the form of system logs, and these data files represent the main avenue by which system administrators can gain insight into the behaviour of the systems.

Due to the size of such data files and the complexity of such systems, system administrators usually adopt a divide and conquer approach to analyse the data. Such log files are typically incomplete and redundant; that is, the files may not contain all the relevant events to characterize a failure while containing several interleaved events related to the same failure.

There are several possible ways to increase the dependability of these computer systems [1], with failure prediction [2] or failure diagnosis [3][4] being the most prominent ones. One of the basic tasks of *automatic analysis* of log files for the purposes mentioned above is *preprocessing*, which typically involves filtering the logs. However, such analysis techniques are invariably expensive due to the size and type of the logs being processed. Specifically, these log files are highly redundant and unstructured. To handle the lack of structure in log files, further information is added, often manually, to capture specific aspects of the data, i.e., the data is labelled using system information. To address the redundancy problem, the logs are *filtered*, or *preprocessed*, to aid the log analysis process. Specifically, to handle redundancy, *Filtering/compression techniques*, or redundancy handling techniques are used to remove events that are not useful for any analysis. A common problem with such compression techniques is that they may remove important information that are pertinent to the analysis phase, as captured by the targeted high compression or filtering rate [5]. The terms *compression* and *filtering* are used interchangeably.

This paper seeks to bridge this gap; we filter event logs based on their similarities. We propose a *novel and generic* clustering approach to log filtering where events that are not similar but causality related are also kept. This is to preserve events patterns that serve as precursor to failures. We focus on trying to achieve good failure analysis, hence we evaluate the impact of the filtering on failure pattern detection approach.

The rest of the paper is organised as follows: In Section II, we present the system logs and models we assumed in the paper. We present the methodology for achieving our objective in Section III, while failure pattern detection for performance evaluation is presented in Section IV. In Section V, we discuss the results when applying the approach

 ISBN: 978-1-61208-429-9

to log data from different supercomputer systems. Related works are presented in Section VI. We conclude the paper and provide direction for future work in Section VII.

II. Models and System Logs

A. Basic Definitions

We will refer to Figure 1 (sample logs from Ranger supercomputer) in this section for definition of terms.

- **Event:** A single line of text containing various fields (*time-stamp, nodeID, protocol, application, error message*) that reports the activity of a particular cluster system. Such an event is also often called a *log message*.
- **Event logs:** A sequence of events containing the activities that occur within a cluster system.
- **Similar Events:** These are events containing similar log messages based on the similarity measure used. From Figure 1, events 5 and 6 can be considered similar.
- **Identical Events:** These are events believed to be exactly the same and/or are produced by the same 'print' statement, e.g., events 7 and 8 in Figure 1.
- **Failure Event:** This is an event that is often associated with and/or is indicative of a system failure.
- **Sequence** A sequence consists of one or more consecutive events logged within a given time period. In this paper, *sequence* and *patterns* means the same and are used interchangeably.

Table I. summary of logs used from production systems

System	Log Size	Messages	Start Date	End Date
Syslogs	1.2 GB	$> 10^7$	2010-03–30	2010-08-30
Ratlogs	4.3 GB	$> 2 \times 10^7$	2011-08-01	2012-01-20
Blue Gene/L	730 MB	$4,747,963$	2005-06-03	2006-01-04

B. System Model and Cluster Logs

Here we explain the model of our system and explain the logs we work with as well as the event types contained in the logs.

1) Cluster System: A cluster system contains a set of nodes, jobs or tasks, production time, job scheduler and sets of software components (e.g., parallel file system). The job scheduler allocates jobs to nodes with certain production time, and all the components involved write logs to a writing container. This is a common model for most of the cluster vendors like Ranger, Cray, IBM etc. In this research, we use the log of two popular cluster systems, namely (i) Rationalised logs (ratlogs) from Ranger Supercomputer, (ii) syslog from Ranger supercomputer and (iii) IBM BlueGene/L. Table I shows a summary of the logs from the cluster systems we focused on in this research.

2) Cluster Event Logs: Different attributes are used by supercomputer vendors to represent its components. The IBM standard for Reliability, Availability, Serviceability (RAS) logs incorporates more attributes for specifying event types, severity of the events, job-id and the location of the event[6].

An example of Ranger's (syslog) event can be seen below: *Apr 4 15:58:38 mds5 kernel: LustreError: 138-a: work-MDT0000: A client on nid .*.*.5@o2ib was evicted due to a lock blocking callback to .*.*.5@o2ib timed out: rc -107*

It has five attribute fields namely: **Time-stamp** (*Apr 4 15:58:38*) containing the month, date, hour, minute and second at which the error event was logged. **Node Identifier** or **Node Id** (*mds5*) identifies the nodes from which the event is logged. **Protocol Identifier** (*kernel*) and **Application** (*LustreError*) provides information about the sources of logs. **Message** (*A client on nid *.*.*.5@o2ib was evicted due to a lock blocking callback to *.*.*.5@o2ib timed out: rc -107*) contains alphanumeric words and English-only words. The English-only words (A client on nid was evicted due to a lock blocking callback to timed out) is believed to give an insight into the error that has occurred. They are referred to as *Constant*. The alpha-numeric tokens (*.*.*.5@o2ib ,rc-107*) also called *Variable*, signify the interacting components within the cluster system. The ratlogs have and additional field *(job-id)* which differentiates it from syslogs. Detailed example of the Ranger logs is seen in Figure 1 and IBM's Blue Gene/L (BGL) is seen in Table II.

C. Failure Model

The failures we focused on in this paper are those that cause the system to malfunction, i.e., execution of some jobs has stopped. For example, such a failure in IBM BlueGene/L is characterized by *FAILURE* severity level while, in the Ranger Supercomputer, these failures are characterized by a *compute node soft lockups*.

These failures usually occur as a result of faults occurring in the system, i.e., caused by the fault(s) of one or more subsystems/components in the system [7]. These faults result in a log entry or fault message in the log data file. For example, a network timeout will result in a "Network timeout" log message being recorded. Hence, in a typical sequence, there will be an interleaving of fault events and normal events.

III. Methodology

We first briefly explain the existing filtering approach, which we call *normal filtering*. We next explain our filtering approach which is based on simple iterative clustering. The clustering is based on the notion of Levenshtein's Distance (LD), defined on the events messages to capture their similarities.

Filtering based on defined heuristics is applied to purge out redundant events. The resulting event sequences are then transformed into term frequency matrices which serve as input to detection algorithm.

A. Normal Event Filtering

Filtering or compression as it may be called here, is meant to reduce the complexities that comes with analysing logs. It is generally agreed that filtering or pre-processing logs is an important process. The process helps eliminate redundant events from logs, thereby reducing the initial huge size. This

 ISBN: 978-1-61208-429-9

TABLE II. AN EXAMPLE OF EVENT FROM BLUE GENE/L RAS LOG

Rec ID	Event Type	Facility	Severity	Event Time	Location	Entry Data
17838	RAS	KERNEL	INFO	2005-06-03-15 .42.50.363779	R02-M1-N0-C:J12-U11	instruction cache parity error corrected

```
1:  Mar 29 10:00:44   i128-401  kernel:  [8965057.845375] LustreError: 11-0: an error occurred while communicating with *.*.*.36@o2ib. The
    ost_write operation failed with -122
2:  Mar 29 10:00:53  i128-401  kernel:   [8965077.319555] LustreError: 11-0: an error occurred while communicating with *.*.*.28@o2ib. The
    ost_write operation failed with -122
3:  Mar 29 11:27:16  i182-211  kernel:   [8981960.031578] a.out[867]: segfault at 0000000000000000 rip 0000003351c5b2a6 rsp 00007fffdcd318c0
    error 4
4:  Mar 29 11:27:16  i115-209  kernel:   [2073150.255467] a.out[22921]: segfault at 0000000000000000 rip 0000003ad725b2a6 rsp 00007fffbf1a6d40
    error 4
5:  Mar 30 10:02:24  i107-308  kernel:   [8966098.630066] BUG: Spurious soft lockup detected on CPU#8, pid:4242, uid:0, comm:ldlm_bl_22
6:  Mar 30 10:02:24  i107-308  kernel:   [8966098.642055] BUG: soft lockup detected on CPU#10, pid:21851, uid:0, comm:ldlm_bl_13
7:  Mar 30 10:09:25  i107-111  kernel:   [8966563.203631] Machine check events logged
8:  Mar 30 10:09:51  i124-402  kernel:   [8965663.148499] Machine check events logged
9:  Mar 30 10:10:22  master    kernel:   LustreError: 28400:0:(quota_ctl.c:288:client_quota_ctl()) ptlrpc_queue_wait failed, rc: -3
10: Apr  1 05:23:54  i181-409  kernel:  [9203054.301173] Machine check events logged
11: Apr  1 05:23:58  visbig      kernel:  EDAC k8 MC0: general bus error: participating processor(local node response), time-out(no timeout)
    memory transaction type(generic read), mem or  i/o(mem access), cache level generic)
```

Figure 1. Sample Log events for RANGER Supercomputer

however, must avoid removing useful events or event patterns that are important for failure pattern detection. In normal log filtering, events that repeats within certain time window are removed, only the first is kept. This simple log filtering is what we refer to as *normal filtering* in this work. Details can be seen in [6].

B. Preprocessing

Tokenization and Parsing

This phase involves parsing the logs to obtain the event types and event attributes, using simple rules. Tokens that carry no useful information for analysis are removed. For example, numeric-only tokens are removed but *attributes* (alpha-numeric tokens) and the *message types* (English-like only terms) are kept. Also, fields like protocol identifier and application are removed or omitted during the parsing and tokenizing phase.

Message part contains English words, numeric and alphanumeric tokens. The English tokens show a pattern providing information pertaining to the state of the system. The alpha-numeric tokens capture the interacting components or software functions involved. These interacting components, which do not occur frequently and show less or no pattern, are also important since we are interested in interacting nodes of the cluster system. The numeric only tokens are removed as they only add noise.

C. Filtering: Redundancy Handling

1) Logs Message types Extraction and Labelling through Clustering: Generally, data clustering techniques group similar data points together, based on some closeness measure. The output of such clustering algorithms is a set of clusters, where each of the clusters contain members (data points) that are similar (or close) to each other and very dissimilar to members of other clusters. In order to identify all the unique events in the logs, we first extract the message types and we introduce a clustering technique (see Algorithm 1) that partitions the logs based on events similarities given by an *edit* distance. Each cluster represents a unique event.

Edit Distance - (Levenshtein's Distance): The closeness of events is measured using Levenshtein's Distance (LD) [8]. It is a metric that measures differences between two strings. It is defined based on edit operations (insertion, deletion or substitutions) of the characters of the strings. Hence the Levenshtein's distance between two strings s_1 and s_2 is the number of operations required to transform s_2 into s_1 or vice versa. LD is an effective and widely used string comparison approach. We found it more useful as we easily can define it on tokens rather than characters. We equally found it to be more suitable here than cosine similarity as the later is a vector-based similarity measure.

Events Similarity: In our algorithm, we define LD as the number of operations required to transform one message type into another. Therefore, instead of defining the operations on characters of event message types, we define the operations on the tokens or terms t_i of the event types, $e_i = \{t_1, t_2, ..., t_n\}$. It should be noted that message types of event logs mostly do not have many terms or tokens, therefore the computational overhead is reduced.

Consider the log entries of Figure 1. Events 1 and 2 are both failed communication events by the same node; the communication is, however, with different nodes. Events 7 and 8 are both normal machine checked exceptions. The challenge is that these events greatly increase the feature space of distinct events, making it difficult to handle for any meaningful analysis. In solving this, we consider the following: (1) *Similar events need to be grouped together and considered the same* and (2) *identical events are also considered same and the redundant ones are removed*. We propose an algorithm (Algorithm 1 - see Figure 3) to first find the similarity between these events and then cluster those events that are similar. Then, events in the same cluster are indexed with the same identity (IDs).

From the sample logs of Figure 1, it is necessary that

 ISBN: 978-1-61208-429-9

	Event ID	Time-stamp	Node Identifier	Message
1	LEO	1269856844	i128-401	LustreError: error occurred while communicating with 129.114.97.36@o2ib. The ost_write operation failed with
2	LEO	1269856853	i128-401	LustreError: error occurred while communicating with 129.114.97.36@o2ib. The ost_write operation failed with
3	SEGF	1269862036	i182-211	segfault at rip rsp error
4	SEGF	1269862036	i115-209	segfault at rip rsp error
5	SSL	1269943344	i107-308	BUG: Spurious soft lockup detected on CPU, pid:4242, uid:0, comm:ldlm_bl_22
6	SSL	1269943344	i107-308	BUG: soft lockup detected on CPU, pid:21851, uid:0, comm:ldlm_bl_13
7	MCE	1269943765	i107-111	Machine check events logged
8	MCE	1269943791	i124-402	Machine check events logged
9	CQF	1269943822	master	client quota ctl ptlrpc queue wait failed,
10	MCE	1270099434	i181-409	Machine check events logged
11	GBE	1270099438	visbig	general bus error: participating processor local node response, time-out no timeout memory transaction type generic read, mem or io mem access cache level generic

Figure 2. Sample pre-processed logs

any similarity metric used must consider the order of the terms in the events for meaningful result. For example, the event messages *...error occurred while communicating with...* and *...Communication error occurred on...* may appear similar but semantically different. A similarity metric that does not take order of tokens/terms into consideration will cluster these events together, i.e., these events will be seen as similar, because they have similar terms. To address this challenge, we define an *edit distance* metric on terms without transposition, taking term order into consideration. Also, defining this metric based on terms or tokens reduces the computational cost incurred as opposed to when it is defined on string characters.

Finally, to capture the similarity of events, we define a similarity threshold, where the lesser the number of edits, the higher the similarity. Hence, we define the threshold such that, when the edit distance between a pair of messages is less than or equal to the threshold λ (hence highly similar), these events are regarded as similar and thus clustered together.

Event Similarity Threshold

It has also been observed that events that can be regarded similar do not have much difference in terms of the number of terms contained in the event messages. Using an *iterative approach* [9], we start with a small value of similarity threshold λ, then increase the value in small increments and monitor the output, until a satisfied similarity value is obtained. We observed that with a very small similarity threshold, only events that are exactly similar are clustered together. But, as the value of threshold is increased to values higher than 3, events that are often dissimilar were being classed as similar. Therefore, to have a more acceptable result, we chose a threshold of 2.

Clustering event logs and ID assignment The challenges addressed by this algorithm and its approach can be explained in two steps:

STEP I: The events are grouped based on the value of the edit distance or LD. In this step all events with equal terms or token length are clustered together. This is because they will have same value of LD.

Algorithm 1
Input: *Log events* $e_0 \ldots e_n$, *MinimumSimilarityThreshold* λ
Output: Log events with cluster IDs

1: *initialise* $s_0 = e_0$;

2: **for** all log events $e_i, i = 0 \ldots n$ **do**{ }
3: Obtain events similarities using Levenshtein Distance *similarity*(s_0, e_i)=LD(s_0, e_i);
4: **end for**
5: **for** all log events **do**
6: **if** *similarity* $<= \lambda$ **then**
7: Assign log events to cluster
8: Assign ID to log event representing its cluster
9: **end if**
10: **end for**
11: Repeat step 2 for clusters with problem explained in **STEP II** above
Until all log events are clustered
12: Return() {outputs log events with their cluster ID}

Figure 3. An algorithm that Clusters event logs base on similarity and assign event IDs (represent the clusters).

STEP II: Since LD gives the number of operations performed to transform one event to the other, different event types with same token length are clustered together from the step 1. For example, ... *"machine check event logged"* and ... *"Spurious soft lockup detected"* will belong to the same cluster. This step partitions clusters with such problems with smaller LD value. This step is performed recursively until the clusters contains only events of similar message type. More on this is shown is Algorithm 1 seen in Figure 3.

As example of the output of this step is shown in Figure 2. These logs still contain redundant events, for example, events 5 and 6 (please observe that events 5 and 6 are clustered together, though being slightly different, and are indexed using the same id).

2) Removing Redundant Events: There are several seemingly identical error events reported frequently in cluster logs. These events are then clustered together and have the same ID from the clustering step. The events are sometimes reported by the same cluster node and they occur within a small time difference (temporal aspect). Also, sometimes

some of these events are reported by different cluster nodes (spatial aspect) but still within the small time difference.

According to Iyer and Rosetti [10], occurrence of similar or identical events within a small time window might likely be caused by the same fault. Thus, these messages are related (and hence redundant) as they potentially point to the same root-cause. Therefore, removing these redundant messages may prove to be beneficial to the analysis stage. In another sense, removing the "redundant" events could be useful in understanding the behaviour of a particular fault in terms of the frequency of the event generated within the period. Therefore, in filtering of redundant log events we consider events in a sequence having the following properties:

- Similar events that are reported in sequence by the same node within a small time window are redundant. This is because nodes can log several similar messages that are triggered by the same fault.
- Similar events that are reported by different nodes in a sequence and within a time window. This could be triggered by the same fault resulting in similar misbehaviour by those affected cluster nodes.
- Identical events occurring in sequence and within a small time difference are redundant.

General approach to filtering will keep the first similar event of sequence and subsequent ones removed [5]. It is pertinent to note that it is possible that the same error messages logged by different nodes are caused by different faults and at close time interval. Some events are causally-related. In our approach, we keep such events. The process of identifying and grouping the error events exhibiting the above properties is done using a combination of both tupling and time grouping heuristics [9]. We define some heuristics that captures the properties outlined above.

With careful observation of the logs and experts' input, we realised that achieving high compression rate and yet preserving patterns are important and dependent on how informative and well-labelled a given log is. For example, Ranger's *Ratlogs* contains more information regarding the nodes and jobs involves which provides more information regarding an event. *Job-ids* in logs indicates particular job that detects the reported event. The job-ids when correlated with failure events, tells which job is the source of the failure. This implies that identical job-ids present on different events within a given event sequences would have high correlation as regards the faults and failure that is eventually experienced [11]. In order to achieve high *events compression accuracy* (ability to keep unique events) and *completeness* (remove redundant events), yet maintaining events which are possible precursor to failure (preserving failure pattern), we propose a filtering approach that removes redundant events or events that are related based on the causes, sources, similarity and time of their occurrence.

Specifically, given two events e_1 and e_2, with the time of occurrence Te_1 and Te_2 respectively, they are both causality related or emanates as result of same faults if:

- $nodeid(e_1) == nodeid(e_2)$ && $|Te_1 - Te_2| \leq t_w$ && $sim(e_1, e_2) \leq \lambda$
- $nodeid(e_1) == nodeid(e_2)$ && $jobid(e_1) == jobid(e_2)$ && $|Te_1 - Te_2| \leq t_w$ && $sim(e_1, e_2) \leq \lambda$,

where $sim(.)$ is the similarity given by LD, λ is similarity threshold.

IV. Case Study: Pattern Detection

A. Introduction

The aim of compression or filtering event logs of large-scale computer systems is to reduce the massive size by properly removing redundant events; and preserving the necessary events patterns to enhance any log analysis. Such analysis can be failure prediction, root cause analysis, failure detection etc. In this section, we introduce an unsupervised pattern detection approach in logs of distributed systems. This is an approach used to evaluate the accuracy and efficiency of our filtering approach. That is, if the approach preserve useful event patterns in logs that improves failure detection.

Filtered logs sequences are now extracted transformed into term-frequency matrix. This matrix comprises of row vectors representing distribution or counts of each event types within a given time and the column vectors representing the sequences or patterns.

We further utilise clustering approach [12] to group similar behaving patterns. Each of these patterns are then expected to be normal event sequences or comprising faulty events.

B. Failure Pattern Detection

According to Gainaru et al. [13], event log sequences can be categorised as noisy, periodic or silent in their behaviour. Noisy sequences occur with high frequency (busty or chatty) and the level of interaction of the nodes involved increases within short period. The characteristics of these patterns are captured through entropy [14] and mutual information. High entropy signifies that the cluster is likely *failure cluster*.

Hence, given a cluster with set of sequences or patterns $C = \{c_1, ..., c_m\}$ and each pattern c_i contains a set of similar events sequences,s, i.e., $c_i = \{s_1, ..., s_n\}$. Then detection is achieved as follows:

$$f(c) = \begin{cases} 1 & \text{if } \varphi(c) < 0 \\ else & \begin{cases} 1 & \text{if } \varphi(c) > \tau \ \& \ H(c) > 0 \\ 0 & \text{otherwise} \end{cases} \end{cases} \quad (1)$$

Where,

$$\varphi(c) = MI(c) - H(c) \quad (2)$$

and $MI(c)$ and $H(c)$ are the mutual information and the entropy of patterns c, τ is detection threshold, the value of $\varphi(c)$ for which we can decide if c contained failure sequences or not.

V. Experiments, Results and Discussion

A. Experimental Setting

We performed our experiments in order to evaluate the effectiveness of our filtering method is preserving useful events patterns that may potentially improve failure detection. Further, we aim to assess the efficiency of our unsupervised detection method on the various logs. The experiment was conducted on three different logs obtained from two cluster systems. The ratlogs and syslogs are obtained from the Ranger supercomputer sited at Texas Advanced Computing Center at the University of Texas at Austin, and the BGL logs from IBM Blue Gene/L supercomputer. These systems were chosen because of the availability of the logs and they are among the top 500 widely used supercomputers. Further, their event logging system is representative of a many other similar systems.

Following, a sequence, an input vector for the pattern detection algorithm is labelled as either a failure or non-failure. We implemented normal filtering approach as explained earlier in order to compare with our approach. Note that we could not implement the approach by Zheng et al. [5] to compare with ours because it is log-specific. It cannot be generalised with logs that are not labelled with severity levels, which is the case for most systems. Hence we compare with *normal filtering* method which is the most used.

To form the basis of our evaluation, we use information retrieval metrics *precision* (the relative number of correctly detected failure patterns to the total number of detections); *recall* (the relative number of correctly detected failure sequences to the total number of failure sequences) and *F-measure* (harmonic mean of precision and recall) to measure the performance of our approach. They are as expressed in Equations (3) - (5). We capture the parameters in the metrics as follows: *True positives (TP)*: Number of failure sequences/patterns correctly detected. *False positives (FP)*: Number of non-failure (good) sequences detected as failure. *False negatives (FN)*: Number of failure sequences identified as non-failure sequences.

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

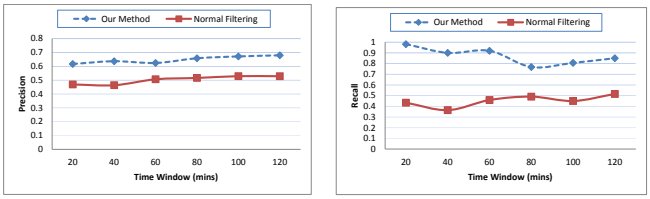
$$F - Measure = 2 * \frac{Precision \times Recall}{Precision + Recall} \tag{5}$$

B. Results

The results are captured in Figures 5, 6, 7 and 8. Each plot of the graphs contains two curves, one is representing the detection efficacy of *our method*, and the other representing that of *normal filtering*.

As mentioned in the introduction, we obtained a good log events compression from the original size. We obtained compression rate of 78%, 80% and 84% on syslogs, ratlogs and Blue Gene/L logs respectively with $LD = 3$. Normal filtering achieved an average compression of 88%. We show from Figure 4, the compression rate on syslogs data as the value of LD increases.

Figure 4. Compression rate on syslogs data against LD

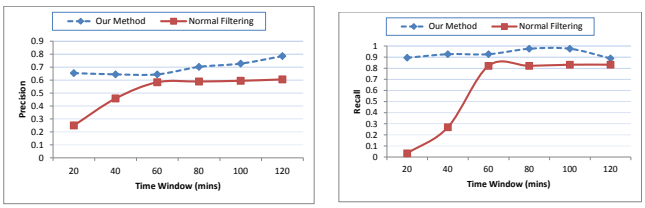


(a) Precision (b) Recall

Figure 5. Precision and recall showing effectiveness of failure detection on *syslogs* filtered with our method and normal filtering

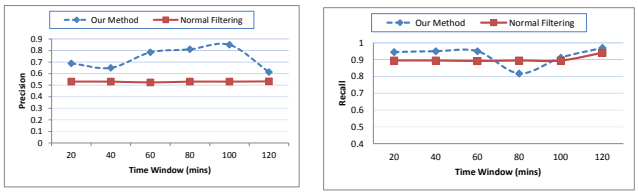
syslogs: Results, as seen in Figure 5, shows that the *precision* and *recall* on logs filtered by *our method* is consistently higher than on those filtered by normal filtering through all the time windows captured. Furthermore, filtering using our method achieve highest precision and recall of 69% and 88%, respectively, *normal filtering* on the other hand is considerably lower with peak precision and recall of 53% and 52% respectively. Our filtering method achieved a relative improvements of about 16% and 26% over normal filtering, for precision and recall respectively.

ratlogs: On ratlogs (see Figure 6), both *precision* and *recall* for our filtering method are consistently high across time windows, ranging between $67\% - 80\%$ for the former and between $79\% - 98\%$ for the latter. However, both precision and recall for normal filtering is inconsistently low, with maximum precision of 60% and recall of 82%.



(a) Precision (b) Recall

Figure 6. Results showing effectiveness of failure detection on *ratlogs* filtered with our method and normal filtering



(a) Precision (b) Recall

Figure 7. Showing effectiveness of failure detection on *BlueGene/L (BGL)* logs filtered with our method and normal filtering

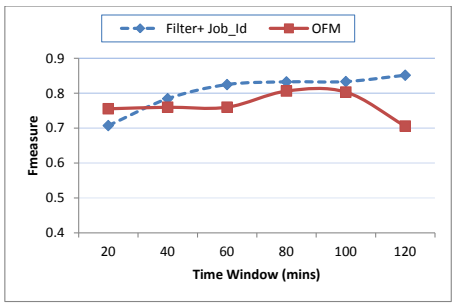


Figure 8. Detection performance on ratlogs using our filtering method without using additional structure (legend: *OFM*) and logs compressed with additional useful structure (legend: *Filter + Job_ID*).

Similarly, on **IBM BlueGene/L (BGL)**, the *precision* as seen in Figure 7 shows there is improvement in detection using our method over normal filtering. It achieved an average improvement of about 30% over normal filtering. *Recall* for both methods are high, however, our method performed better at smaller time windows.

What is the implication of these results? Our method achieved an improvement over *normal filtering* of about $10\% - 30\%$ across all logs used. One of the reasons for this is that, after careful manual investigation, we discovered that failures experienced as captured in these logs are often preceded by event patterns within a short time window. Further, our filtering method was able to preserve these precursor events to failures. This implies that our approach can aid system administrators take necessary failure preventive measures earlier.

We show the result of compression taking *job-ids* into consideration in Figure 8. This result is for ratlogs only, being the only logs with job-id field among the three logs used. The result shows that there is a remarkable improvement over not using job-ids for compression with an average detection improvement of about 15%. The increased detection in logs compressed with job-ids can be explained by the fact that events which are reported by same jobs and are semantically related, yet not similar are properly filtered.

VI. Related Work

Data mining and machine learning techniques are the mostly used in recent works that focused on analysing logs for failure analysis in cluster systems. These works can be found in [15][16][17][18] and [19], and they all developed algorithms that mine patterns of events in the logs. The works in [20] and [18] combines console logs with source code and employed PCA to obtain faulty patterns in the logs. The authors of [2] proposed a method for analysing system generated messages by extracting sequences of events that frequently occur together. In [21], the authors proposed a technique and developed a tool based on clustering called HELO, to extract event templates and describes the templates for system administrator's use. None of the above work considers removing any redundancy in the events logs. They considered every event useful for analysis.

Zheng et al. [5] proposed a method that pre-processes logs and removes redundant events without losing important ones, for failure prediction. In their approach, redundancy from both a temporal and spatial viewpoint is considered. They also filter events based on their causal relationship. Unlike this method, we assume that temporal events must occur in sequence to be removable and we believe that causally-related but semantically unrelated events are patterns or signatures to failure, therefore we keep them. Since the method of [5] cannot be implemented on logs without severity levels, we conjecture that the approach will yield either a high false positive or negative, should it be used on these types of logs. Hence we couldn't compare this method with ours.

Pecchia et al. [22] developed an approach based on heuristics combined with statistical techniques that provides likelihood of events produced by different nodes to removed unwanted events. Their approach is different from ours as they focused on analysing the effects of tupling on compression while we proposed a new filtering approach.

Other approaches that use clustering can be found in [23] and [16]. The latter mainly focus on extracting the message types that can be used for indexing, visualization or model building. One of the caveats of this approach is that it clusters events/message types that are believed to have been produced by the same print statement and their occurrences is non-overlapping. In contrast, our approach can cluster overlapping and non-overlapping events together.

VII. Conclusion and Future Work

We have presented a novel, generic compression algorithm that can be instantiated according to the structure of the log files. Our method did not only compressed logs, it first extract message types in logs. The clusters formed and indexed with ids represents message types. These message types are useful in log analysis e.g., visualization, indexing. Our compression method make use of event similarity (Levenshtein distance), and event structure to determine a redundant event. The efficiency of the compression technique is validated through a proposed pattern detection algorithm. The results from three different logs demonstrate that compression does not only reduce log size which leads to low computational cost of failure analysis, but also enhance better detection of failure patterns. As future work, we intend to use the result and perform failure prediction.

 ISBN: 978-1-61208-429-9

ACKNOWLEDGEMENTS

The data which was analyzed in this paper was available through the SUPReMM project funded by NSF grant ACI-1023604, and has utilized and enhanced the NSF-funded system Ranger (OCI-0622780). We thank the PTDF Nigeria for partly funding this research.

REFERENCES

[1] J.-C. Laprie, "Dependable computing: Concepts, challenges, directions," in *COMPSAC*, 2004, p. 242.

[2] A. Gainaru, F. Cappello, J. Fullop, S. Trausan-Matu, and W. Kramer, "Adaptive event prediction strategy with dynamic time window for large-scale hpc systems," in *Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*, ser. SLAML '11. New York, NY, USA: ACM, 2011, pp. 4:1–4:8.

[3] E. Chuah, S. hao Kuo, P. Hiew, W.-C. Tjhi, G. Lee, J. Hammond, M. Michalewicz, T. Hung, and J. Browne, "Diagnosing the root-causes of failures from cluster log files," in *2010 International Conference High Performance Computing (HiPC)*, dec. 2010, pp. 1 –10.

[4] A. J. Oliner, A. Aiken, and J. Stearley, "Alert detection in system logs," in *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE, 2008, pp. 959–964.

[5] Z. Zheng, Z. Lan, B. H. Park, and A. Geist, "System log pre-processing to improve failure prediction," in *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*. IEEE, 2009, pp. 572–577.

[6] Y. Liang, Y. Zhang, A. Sivasubramaniam, R. Sahoo, J. Moreira, and M. Gupta, "Filtering failure logs for a bluegene/l prototype," in *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, June 2005, pp. 476–485.

[7] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," in *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, no. 1, 2004, pp. 11–33.

[8] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," in *Soviet physics doklady*, vol. 10, 1966, p. 707.

[9] J. Hansen and D. Siewiorek, "Models for time coalescence in event logs," in *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on*, jul 1992, pp. 221 –227.

[10] R. Iyer, L. Young, and P. Iyer, "Automatic recognition of intermittent failures: an experimental study of field data," in *Computers, IEEE Transactions on*, vol. 39, no. 4, apr 1990, pp. 525 –537.

[11] E. Chuah, G. Lee, W.-C. Tjhi, S.-H. Kuo, T. Hung, J. Hammond, T. Minyard, and J. C. Browne, "Establishing hypothesis for recurrent system failures from cluster log files," in *Proceedings of the 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, ser. DASC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 15–22.

[12] N. Gurumdimma, A. Jhumka, M. Liakata, E. Chuah, and J. Brwone, "Towards detecting patterns in failure logs of large-scale distributed systems," in *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2015 IEEE International*. IEEE, 2015.

[13] A. Gainaru, F. Cappello, and W. Kramer, "Taming of the shrew: Modeling the normal and faulty behaviour of large-scale hpc systems," in *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, May 2012, pp. 1168–1179.

[14] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," in *SIGCOMM Computer Communication Review*, vol. 35, no. 4. New York, NY, USA: ACM, Aug. 2005, pp. 217–228.

[15] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *IP Operations Management, 2003. (IPOM 2003). 3rd IEEE Workshop on*, 2003, pp. 119–126.

[16] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "A lightweight algorithm for message type extraction in system application logs," in *IEEE Trans. on Knowl. and Data Eng.*, vol. 24, no. 11. Piscataway, NJ, USA: IEEE Educational Activities Department, Nov. 2012, pp. 1921–1936.

[17] M. Aharon, G. Barash, I. Cohen, and E. Mordechai, "One graph is worth a thousand logs: Uncovering hidden structures in massive system event logs," in *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part I*, ser. ECML PKDD '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 227–243.

[18] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 117–132.

[19] X. Fu, R. Ren, J. Zhan, W. Zhou, Z. Jia, and G. Lu, "Logmaster: Mining event correlations in logs of large-scale cluster systems," in *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*, Oct 2012, pp. 71–80.

[20] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Mining console logs for large-scale system problem detection," in *Workshop on Tackling Computer Problems with Machine Learning Techniques (SysML), San Diego, CA*, 2008.

[21] A. Gainaru, F. Cappello, S. Trausan-Matu, and B. Kramer, "Event log mining tool for large scale hpc systems," in *Proceedings of the 17th International Conference on Parallel Processing - Volume Part I*, ser. Euro-Par'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 52–64.

[22] A. Pecchia, D. Cotroneo, Z. Kalbarczyk, and R. Iyer, "Improving log-based field failure data analysis of multi-node computing systems," in *Dependable Systems Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, June 2011, pp. 97–108.

[23] S. Jain, I. Singh, A. Chandra, Z.-L. Zhang, and G. Bronevetsky, "Extracting the textual and temporal structure of supercomputing logs," in *High Performance Computing (HiPC), 2009 International Conference on*, Dec 2009, pp. 254–263.

 ISBN: 978-1-61208-429-9

An Investigation of the Impact of Double Single Bit-Flip Errors on Program Executions

Fatimah Adamu-Fika and Arshad Jhumka

Department of Computer Science
University of Warwick
Coventry, CV4 7AL UK
Email: {fatimah,arshad}@dcs.warwick.ac.uk

***Abstract*—This paper investigates a novel variant of the *double bit errors* fault model and studies its impact on program execution. Current works have addressed the problem of both random bit upsets occurring in the *same* location (a given memory word or register). In contrast, we randomly *select two locations and flip a single bit at each location*, which we call *Double Single Bit-flip* (DSB) variant. We then evaluate the viability of this new variant in uncovering vulnerabilities in software (SW). As a baseline for comparison, we inject traditional single bit-flip (SBF) errors in registers. To better understand the impact of the injected faults on SW, we classify the behaviour of the program in five possible failure categories. Our results, based on nearly a million fault-injection experiments, show that (i) DSB causes a significantly higher proportion of SW failures than SBF errors, (ii) a large proportion of those failures was crash failure and (iii) under DSB, the proportion of silent data corruptions (SDC) varies significantly between programs from different application areas. The failure profile induced by DSB is very different to other fault models, such as SBF.**

Keywords–Multiple bit-flip errors; Fault injection; Failure profile; Evaluation.

I. INTRODUCTION

With the ever-decreasing size of hardware and issues such as temperature hotspots [1], computer systems are being subjected to increasing rate of transient faults. These transient faults originate from the transistor level. These faults typically cause a corruption of the state of the program, i.e., errors exist in the program [2]. To mimic these errors, bit-flip errors are typically artificially injected into the program state during a process called fault injection [3]. Traditionally, a single bit-flip error was injected in a single run of the program. This involves selecting a variable at a given location in the program and, when execution reaches this location, a single bit upset (SBUs) is performed on the selected variable. However, the increasing rate of transient faults have limited the usefulness of SBUs in uncovering vulnerabilities, necessitating multiple-bit upsets (MBUs) to be injected in a single run.

Fault injection is a widely used technique for the validation of dependable systems. Its importance is being increasingly recognised, with its recommendation as a highly valuable assessment method in the recently published ISO 26262 standard [4] for functional safety of road vehicles supporting this increasing importance. It is expected that single event upsets will likely create MBUs in forthcoming hardware circuits [5][6], including those in embedded systems. In anticipation of this problem, several work have started investigated double-bit upsets (DBUs) fault model [7][8]. However, these works focused on one variant of DBUs: at a given location, *two bits* are randomly selected and are subsequently inverted. There is a rareness of field data on how these hardware errors will manifest. This is also observed in [7][8]. In [9], it has been shown that multiple memory errors may occur as: (i) several bit upsets within a single location, (ii) one or more bit upsets across several locations or (iii) several bits upsets all across the chip. In this paper, we investigate a variant of DBUs: *two locations are selected and a SBU is injected at each location*. We call this new fault model the Double Single Bit-flip (DSB) fault model.

The usefulness of a fault model is its ability to uncover vulnerabilities in a system. Specifically, it is often the case that the *error sensitivity* of a software system is assessed with respect to the errors being injected according to the proposed fault model. Error sensitivity is commonly defined as the likelihood that a softare component will produce a SDC, which is a type of problem that often goes undetected by the system, as a result of a hardware error. It also often the case that the failure profile of the system is evaluated with respect to the fault model. Hence, we have conducted an extensive fault injection campaign, with close to one million fault injection experiments on five different software modules, each with different software structures, to validate the DSB fault model. Our contributions are: (i) we investigate the impact of DSB on program execution, (ii) we conduct a large-scale fault injection experiments, of close to a million executions, to assess the usefulness of DSB, and (iii) our results show that DSB induces very different failure profiles in software than existing fault models, such as SBF. We conclude that DSB is indeed useful in uncovering vulnerabilities.

The remainder of the paper is structured as follows: In Section II, we present the system and fault models we assume in the rest of the paper. We detail the experimental setup used in Section III. In Section IV, we present the results of our experiments. We present an overview of related work in Section V. We conclude the paper in Section VI.

II. MODELS

In this section, we present the system model and the fault model we assume in the rest of the paper.

 ISBN: 978-1-61208-429-9

A. System Model

In this paper, we consider modular software, i.e., software that consists of a number of discrete software functions, called modules, that interact to deliver the requisite functionality. We consider a module as a generalised white-box, having multiple inputs and outputs and whose codebase is available. We do not assume knowledge of the implementation details. The codebase is needed only to enable the software to be instrumented to enable errors to be injected.

Modules communicate with each other in some specified way using different forms of signalling, e.g., shared memory, parameter passing etc. This is usually down to the nature of the software and to the chosen communication model. A software module performs computations using the inputs received on its input channels to generate the outputs, which are then placed on the requisite output channels.

B. Fault Model

Our fault model is transient hardware faults that ultimately affect the software modules. These faults typically originate at the transistor level due to issues such as hardware size and temperature hotspots. These faults affect the state of the program by changing the content of memory and registers (i.e., different locations), causing *errors* [2] to exist in the software. These errors in software are typically mimicked by injecting bit-flip errors in main memory words and registers. In this paper, we focus only on errors in registers and the total number of errors that can occur in any run is two, i.e,, we randomly select two registers and flip one bit in each. We specifically corrupt the contents of registers immediately before they are written into main memory.

III. Fault-Injection Experiments

In this section, we empirically study how DSB and DBF affect program executions. In section III-A, we describe the target programs, the modules that are instrumented in each target program and the input sets that are processed by the programs during injection. We then describe how the modules are instrumented and how the fault injection experiments are done in III-B.

A. Target Programs

We select five different modules from two different software systems for instrumentation. The first system is an image recognition package, SUSAN (Smallest Univalue Segment Assimilating Nucleus) [10], developed for noise filtering and for recognising corners and edges in Magnetic Resonance Image (MRI) of the brain. The second software system is the Mathwork's implementation of a flight control system for the longitudinal motion of an aircraft [11]. We target five different modules within these systems, three from SUSAN and two from the flight control systems.

The three different modules we use in SUSAN are for corners detection, edges detection and noise filtering, which we refer to as corners, edges and smoothing, respectively, in the rest of the paper. We select two modules within the flight control system, (i) the module for updating derivatives for the root system and (ii) the module for updating model step. We refer to these modules as derivatives and step, respectively. Details are provided in Table I for description of input set.

TABLE I. SIZES OF TARGET MODULES AND DESCRIPTION OF THEIR INPUT SET.

Module	Size (bytes)	Input description
Corners	7975	PGM files:
Edges	6053	A simple four-sided geometric shape (7292 bytes)
Smoothing	3488	Multiple geometric shapes of various shapes and sizes (65551 bytes)
		An image (111666 bytes)
Derivatives	2915	Pilot Frequency in rads/secs:
Step	10249	Variable of type unsigned long long
		between 0.030000000000000000 to 0.1199999999999999

B. Experimental Setup

In this section, we provide details about the experimental setup and the fault injection experiments that we conducted.

1) System platform: The experiments were executed on a 3 GHz Intel Core i7 machine, with 16 GB, 1600 MHz DDR3 and 500 GB solid state drive. The machine was running Darwin OS version 14.0.0.

2) Target system: For our faullt injection experiments, we used a variant of LLVM fault injection tool (LLFI) [12], which we refer to as Fault-Rate LLFI (or FR-LLFI) [13]. LLFI works at the LLVM [14] compiler's intermediate representation (IR) level. FR-LLFI allows the injection of faults using a fixed probability, which is called *fault rate*, rather than a single fault per execution. We extended FR-LLFI to allow for multiple bit flips in specific points, we also added the functionality of allowing the selection of what bit(s) to flip at specific points.

To perform a fault injection, we first compiled the source files into a single IR file. The compiled IR file together with a fault injection configuration script (written in PyYaml format [15]) are then fed to the *extended FR-LLFI instrumentor (instrumentor)* for instrumentation. The instrumentor outputs executables (IR and C/C++ object files) to be passed to the *extended FR-LLFI Profiler (profiler)* for profiling and *the extended FR-LLFI fault-injector (fault-injector)* for fault injection.

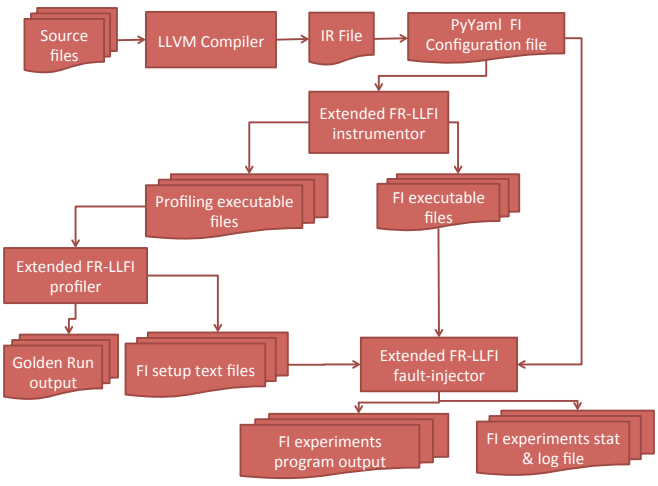


Figure 1. Extended FR-LLFI fault-injection (FI) workflow.

We then passed the executables generated for profiling into the *profiler*. The profiler then generates the setup files (text files) to be used by the *fault-injector* for the fault injection phase. In addition, the *profiler* executes a fault-

free execution of program. This fault-free execution is called *golden run*.

Finally, we fed the setup files generated by the *profiler*, the fault injection executables generated by the *instrumentor* and the fault-injection configuration script into the *fault-injector*.

The *fault-injector* selects an instance of the places of interest specified in the fault injection IR file generated by the *instrumentor* and then inject fault into it at runtime (execution of the fault injection C/C++ object file generated by the *instrumentor*). The output of the *fault-injector* is the fault injection experiments, consisting of program output, log and stat files. Figure 1 depicts the workflow of *extended FR-LLFI*.

TABLE II. VARIABLES SELECTED FOR FAULT INJECTIONS IN DIFFERENT BLOCK LOCATIONS.

Target Program	Module	Variable	Size (bits)	Location (Block)	Alias
SUSAN	Corners	x_size	32	early	SC_A
		y_size	32	early	SC_B
		n	32	central	SC_C
		c	8	central	SC_D
		xx	32	late	SC_E
		yy	32	late	SC_F
	Edges	x_size	32	early	SE_A
		y_size	32	early	SE_B
		n	32	central	SE_C
		m	32	central	SE_D
		c	8	central	SE_E
		w	32	late	SE_F
		x	32	late	SE_G
		y	32	late	SE_H
	Smoothing	x_size	32	early	SS_A
		y_size	32	early	SS_B
		n_max	32	early	SS_C
		x	32	central	SS_D
		center	32	central	SS_E
		area	32	late	SS_F
		tmp	32	late	SS_G
Flight Longitudinal Controller	Derivatives	Integrate_CSTATE	64	early	FD_A
		ActuatorModel_STATE	64	early	FD_B
		Integrategdot_CSTATE	64	early	FD_C
		Wgustmodel_CSTATE	64	central	FD_D
		Qgustmodel_CSTATE	64	central	FD_E
		AlphaSendorLowPassFilter_CSTATE	64	central	FD_F
		StickPrefilter	64	late	FD_G
		PitchRateLeadFilter	64	late	FD_H
	Step	Integrate	64	early	FS_A
		ActuatorModel	64	early	FS_B
		Integrateqdot	64	early	FS_C
		Wgustmodel	64	central	FS_D
		Qgustmodel	64	central	FS_E
		PitchRateLeadFilter	64	central	FS_F
		Gain3_h	64	late	FS_G
		Sum2_g	64	late	FS_H
		Sum1_m	64	late	FS_I

3) Experimental Procedure: To achieve the goals of the study, we run a number of fault injection experiments into a number of different variables (or combinations of variables) in five different modules. We run each target module on three input sets, one from each of three input categories, namely small, medium and large. Before running these experiments, we partition the source code of the program into three parts, namely (i) early, (ii) central and (iii) late. For each part, we choose two or three variables at random, i.e., variables are partitioned and selected according to their placement in the source code of the program. These variables are shown in Table II, with the part of the program source code they belong to. We define a *target location* (or location for short) as a given register used by the program. When a single bit-flip error is injected, a single location is selected. On the other hand, two locations are selected for DSB errors. A fault injection *experiment* is the injection of a an error under the assumed fault model in a given target location. A fault injection *campaign* for a fault model is a set of experiments for a given input set.

Once a location (or pairs of locations) have been selected, we then injected bit-flip errors exhaustively in the locations to cover all possible combination. For each selected location, fault is injected only once during the execution of the program. For the SBU fault model, we ran n experiment in each target location, n being the length of the register. We injected a total *5136* SBUs in the various modules. For the DSB model, for each location pair and a given input, we ran $n \times m$ experiments, m, n being the length of the target locations. Overall, we injected a total of *955392* DSB errors in the software modules. More details can be found in Table II for the size of target locations.

To better understand the profile of the program, we classify the outcome of each fault injection experiment as (i) a *Safe Run*, if the program terminates normally and with an output identical to that of the golden run's, (ii) as a *No Output failure*, if the program terminates normally but fails to produce an output, (iii) as a *Silent Data Corruption (SDC)*, if the program terminates normally but with an output different to that of the golden run's, (iv) as a *Program Hang*, if the program fails to terminate within a predefined time (we set this to 15 times larger than the execution time of the golden run), and (iv) as a *Crash failure*, if the program is terminated due to an exception by the either the program or the operating system.

IV. EXPERIMENTAL RESULTS

We now analyse the results of the various FI experiments, as presented in Tables III – IV and Figures 2 – 4.

A. Impact of DSB vs Impact of Single bit-Flip Error

The first goal of the paper was to evaluate the impact of DSB errors on programs compared to that of single bit-flip (SBF) errors in the same variables. The results for each module are summarised in Table III, while an overall summary is presented in Figure 2.

TABLE III. AVERAGE OUTCOME DISTRIBUTIONS FOR DIFFERENT MODULES.

Module	Fault Model	Outcome				
		Safe Run	*No Output Failure*	*SDC*	*Program Hang*	*Crash Failure*
Corners	SBF	29.4%	18.3%	8.5%	0.0%	43.8%
	DSB	12.5%	13.8%	15.2%	0.1%	58.4%
Edges	SBF	41.5%	0.0%	3.7%	0.0%	54.7%
	DSB	22.0%	0.0%	3.6%	0.1%	74.3%
Smoothing	SBF	14.3%	0.6%	40.0%	0.0%	45.1%
	DSB	6.7%	1.2%	16.4%	10.8%	64.8%
Derivatives	SBF	0.0%	7.1%	0.0%	0.0%	92.9%
	DSB	0.0%	0.5%	0.0%	0.0%	99.5%
Step	SBF	0.0%	25.6%	0.0%	0.0%	74.4%
	DSB	0.0%	6.7%	0.0%	0.0%	93.2%

1) Overall observations: The first observation to be made is that there is marked difference between the failure profile induced by DSB errors compared to that of single-bit flip errors. Further, the proportion of safe run (i.e., no impact) under DSB errors is halved when compared to the proportion of safe runs under SBF errors (see Figure 2). On the other hand, the proportion of crash failure is considerably higher ($\approx 16\%$) under DSB errors than under SBF errors. Also, we observe a reduction in the occurrence of SDCs under DSB errors than under SBF errors. We conjecture that this result is due to the fact DSB errors induce more severe crash

 ISBN: 978-1-61208-429-9

failures, which cause the programs to prematurely exit, and hence such executions cannot display SDCs.

As a matter of contrast, previous work on double bit-flip errors, where two-bit errors are injected into a given location, concluded that single and double bit-flip errors induce very similar proportions of SDCs. We conclude that DSB errors induce a failure profile different to that induced by the double bit-flip errors. As such, we conclude that DSB errors uncover new vulnerabilities in the system and, hence, need to be considered when validating dependable software systems.

2) Module-level observations: From Table III, we observe that the failure profile is dependent on the given target program. For example, we notice the proportion of safe runs under SBF errors in the SUSAN modules is twice as much as that observed under DSB errors. Further, we observe that all faulty runs, irrespective of fault model, in the modules from the control system end in either no output or crash failure. Additionally, we also notice that only the SUSAN modules suffer from SDCs and program hangs under both SBF and DSB errors.

We also observe that the modules from the control system experience mostly crash failures in the presence of DSB errors. Further, we also notice for the control system modules the proportion of no output failure is significantly higher for SBF errors. We also observe a higher proportion of crash failure for DSB errors than that for SBF errors in the SUSAN modules. Given the nature of control systems, which are at the heart of several safety-critical embedded systems, the fact that a high proportion of the DSB errors leads to failures implies that the control systems will not provide reliable service. SDCs have the property that they have not been detected by the system and, thus, provide a potential vulnerability to the system. Also, we observe that DSB errors induce different failure profiles in different modules.

B. Impact of injection location of failure profile

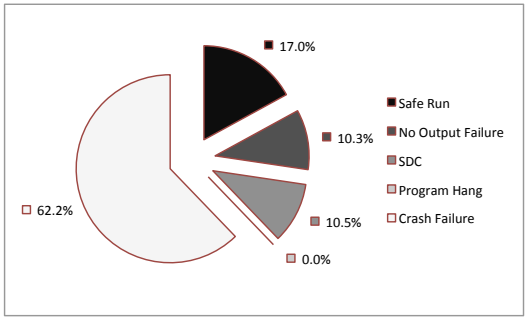
Figures 3 and 4 show the results of the impact injection location has on the failure profile.

1) SBF errors: From Figure 3, the highest proportion, 100%, of safe runs observed in the presence of SBFs is in location SE_D (Figure 3b) and the lowest proportion, 0.0%, is observed in all target locations in derivatives (Figure 3d) and step (Figure 3e).

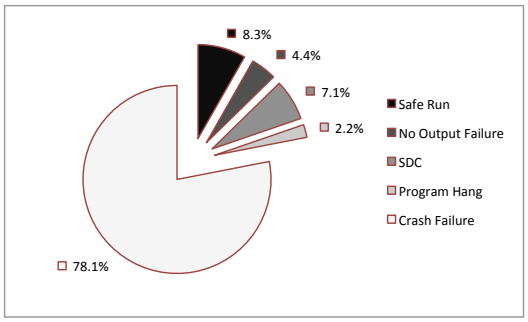
As can be observed from Figure 3, the two modules from the control software suffer a high proportion of crash failures, irrespective of injection location. The other failure type suffered by these two modules are the "no output failure" type. We also observe that, in general, the earlier the injection is performed, the higher the likelihood of a crash failure to happen, i.e., when SBF error is injected in the early part of the modules, the crash failure is more likely to result. On the other hand, crash failure is very likely to happen in the modules of the control software, irrespective of injection location.

2) DSB errors: To understand the impact of injection locations under the DSB errors, we focus on Figure 4 and Table IV

From Figure 4, we observe that failure profiles of the different modules differ from one another. This shows that DSB errors cause these modules to fail differently, thereby



a. Single bit-flip



b. Double single bit-flip

Figure 2. Average outcome distributions over all modules.

inducing different failure profiles in these modules and also that injection locations do affect the failure profiles of these modules.

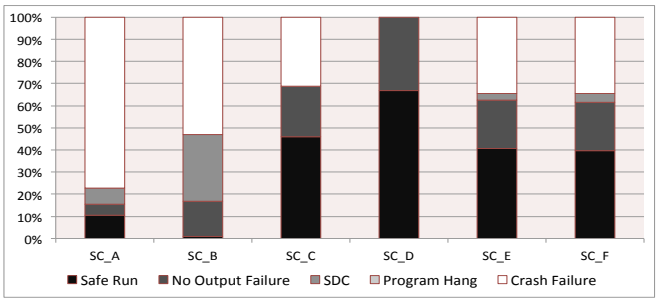
For the two control software modules, any combination of injection locations mostly lead to a crash failure, where the step module suffer a small proportion of the "no output" failure. Focusing on the SUSAN modules, it can be observed that, in general, the earlier an injection is done, the higher the likelihood the failure is a crash failure. On the other hand, it can also be observed that the later an injection is done, the likelihood of a safe run is non-negligible. We now perform a step-by-step comparison between different pairs of injection locations and their respective impact of the software module.

3) DSBs in early blocks vs DSBs in central blocks: We first compare the difference in the impact of DSB errors in early blocks against DSB errors in central blocks, which are shown in Table IV(a) and Table IV(b), respectively. For example, the highest safe run rate observed in early blocks is 0%, whereas the highest safe run rate observed for central blocks is 49.0% (smoothing). The highest proportion of DSB errors in early blocks resulting in crash failure is 99.4% (derivatives) and the lowest is 55.4% (corners), while the highest proportion of SBFs that resulted in crash failure is 99.5% (derivatives) and the lowest, 33.6% (corners).

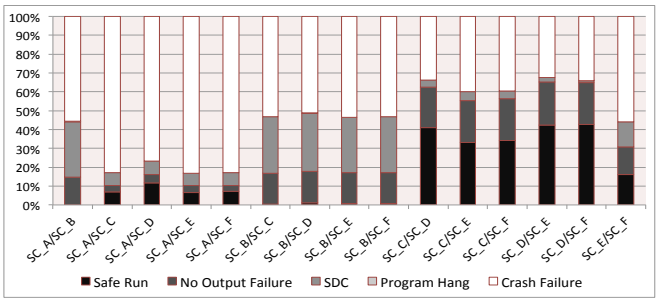
Comparing the results for the different modules, we observe that there is a higher proportion of crash failure, SDCs and program hang when DSB errors are injected in early blocks while, when DSB errors are injected in central block, this results in higher rate of safe run and no output failure.

4) DSBs in early blocks vs DSBs in late blocks: Here, we compare the results of DSB errors in early block with DSB errors in late blocks, as captured in Table IV(a) and

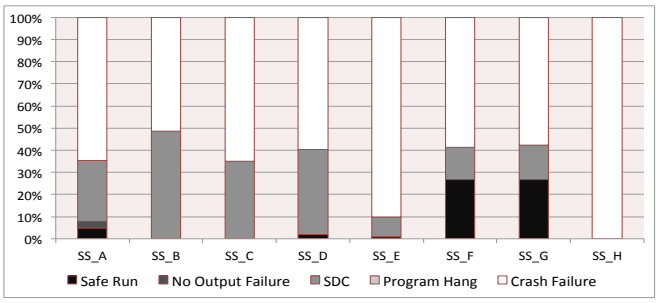
 ISBN: 978-1-61208-429-9



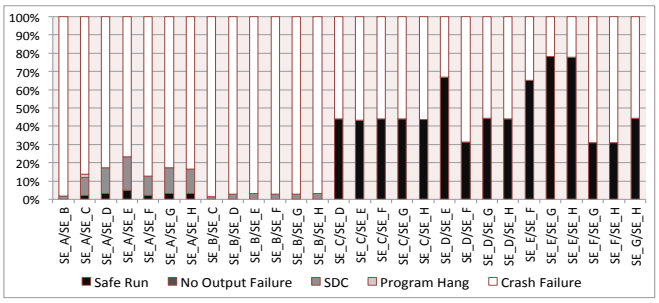
a. Corners



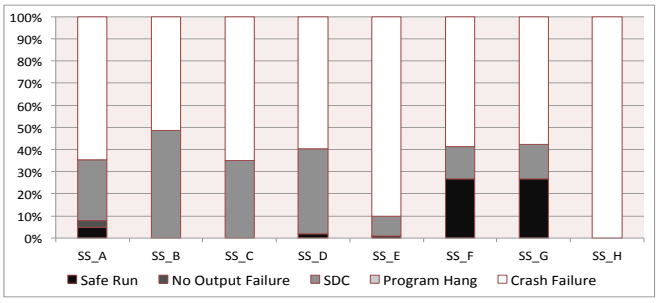
a. Corners



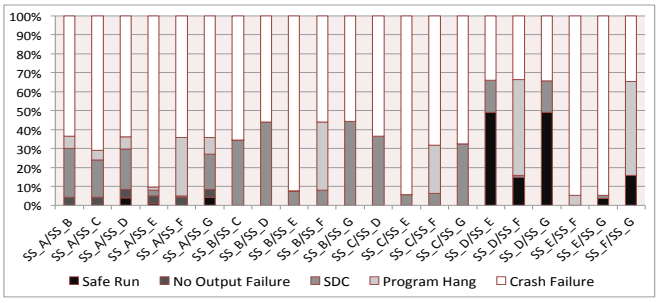
b. Edges



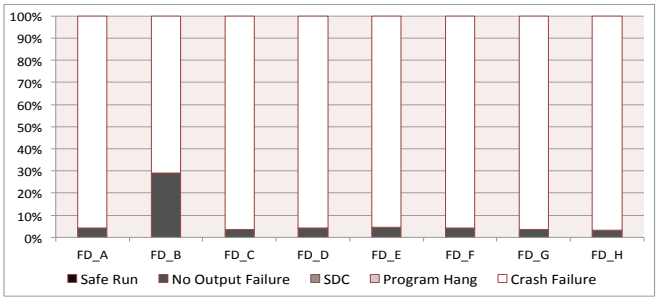
b. Edges



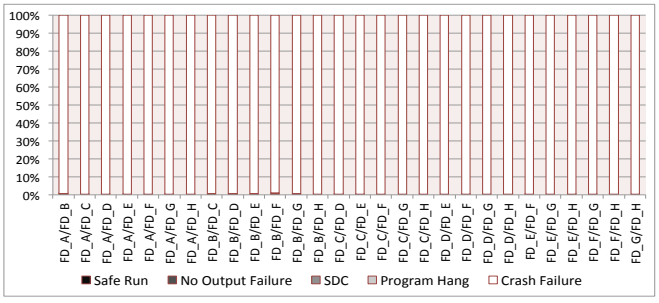
c. Smoothing



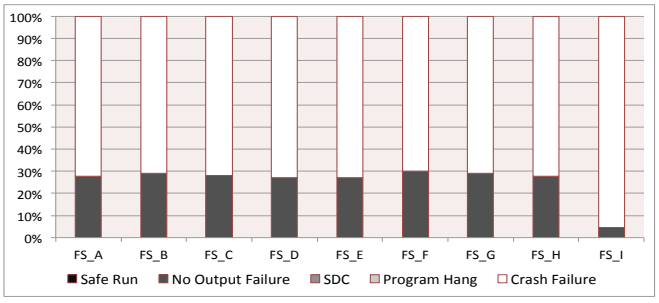
c. Smoothing



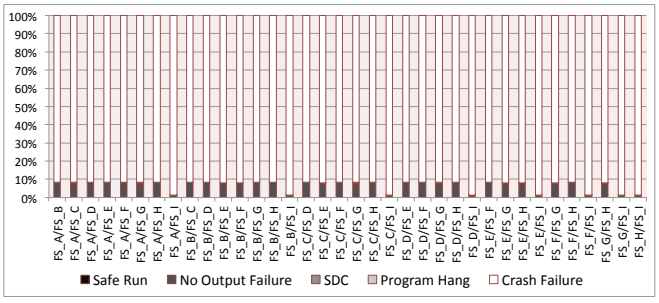
d. Derivatives



d. Derivatives



e. Step



e. Step

Figure 3. Average outcome distributions for SBF experiments for different modules.

Figure 4. Average outcome distributions for DSB experiments for different modules.

TABLE IV. AVERAGE OUTCOME DISTRIBUTIONS FOR DSB IN DIFFERENT BLOCKS COMBINATIONS FOR DIFFERENT MODULES.

(A) EARLY BLOCKS

	Safe Run	*No Output Failure*	*SDC*	*Program Hang*	*Crash Failure*
Corners	0.0%	14.8%	29.3%	0.5%	55.4%
Edges	0.0%	0.0%	1.8%	1.5%	96.6%
Smoothing	0.0%	1.4%	27.1%	4.0%	67.5%
Derivatives	0.0%	0.6%	0.0%	0.0%	99.4%
Step	0.0%	8.3%	0.0%	0.1%	91.7%

(B) CENTRAL BLOCKS

	Safe Run	*No Output Failure*	*SDC*	*Program Hang*	*Crash Failure*
Corners	40.9%	21.7%	3.8%	0.0%	33.6%
Edges	47.8%	0.0%	0.0%	0.0%	52.2%
Smoothing	49.0%	0.0%	16.9%	0.0%	34.1%
Derivatives	0.0%	0.5%	0.0%	0.0%	99.5%
Step	0.0%	8.3%	0.0%	0.0%	91.6%

(C) LATE BLOCKS

	Safe Run	*No Output Failure*	*SDC*	*Program Hang*	*Crash Failure*
Corners	16.1%	14.7%	13.2%	0.0%	56.0%
Edges	35.5%	0.0%	0.0%	0.0%	64.5%
Smoothing	15.7%	0.0%	0.1%	49.7%	34.6%
Derivatives	0.0%	0.4%	0.0%	0.0%	99.6%
Step	0.0%	3.6%	0.0%	0.0%	96.3%

(D) EARLY & CENTRAL BLOCKS

	Safe Run	*No Output Failure*	*SDC*	*Program Hang*	*Crash Failure*
Corners	4.3%	10.0%	18.4%	0.0%	67.3%
Edges	1.5%	0.0%	7.5%	0.3%	90.7%
Smoothing	0.8%	1.4%	19.6%	1.4%	76.7%
Derivatives	0.0%	0.5%	0.0%	0.0%	99.5%
Step	0.0%	8.3%	0.0%	0.0%	91.7%

(E) EARLY & LATE BLOCKS

	Safe Run	*No Output Failure*	*SDC*	*Program Hang*	*Crash Failure*
Corners	3.8%	9.9%	18.1%	0.0%	68.2%
Edges	1.4%	0.0%	7.8%	0.0%	90.8%
Smoothing	0.7%	1.4%	18.4%	16.9%	62.6%
Derivatives	0.0%	0.4%	0.0%	0.0%	99.6%
Step	0.0%	6.0%	0.0%	0.0%	94.0%

(F) CENTRAL & LATE BLOCKS

	Safe Run	*No Output Failure*	*SDC*	*Program Hang*	*Crash Failure*
Corners	35.5%	22.2%	3.8%	0.0%	38.5%
Edges	44.3%	0.0%	0.0%	0.0%	55.7%
Smoothing	16.9%	0.0%	4.9%	14.0%	64.3%
Derivatives	0.0%	0.4%	0.0%	0.0%	99.6%
Step	0.0%	5.9%	0.0%	0.0%	94.1%

Table IV(c), respectively. For example, the highest "no output" failure rate observed in the presence of DSB errors in early blocks in 14.8% (corners) whereas the highest "no output" failure rate observed in the presence of DSBs in late blocks is 14.7% (corners). The highest proportion of DSBs in early blocks resulting in data corruption is 29.3% (corners).

Comparing the results of the different modules, we observe that there is a higher proportion of crash failure, data corruption and no output failure in the presence of DSBs in early blocks, while the presence of DSBs in late blocks result in higher rate of safe runs. Also, the failure profile is more varied (different types of failures) when DSBs are injected in an early block, while the profile is more restricted when DSBs are injected late. Thus, we can conclude that, by *not* injecting in an early block, there is a reduced likelihood of uncovering vulnerabilities.

5) DSBs in early blocks vs DSBs in block combinations: Here, we compare the results of DSBs in early blocks against DSBs in different combinations of blocks, which we present in Table IV(a), Table IV(d), Table IV(e) and Table IV(f), respectively. For example, the lowest crash failure rate seen for DSBs injected in early blocks is 55.4% while the lowest crash failure rate observed for DSBs injected in both early & central block is 67.3%. The highest data corruption rate observed for DSBs in early blocks is 29.3%, while that seen for combination of DSB in early & late blocks is 18.4%. The highest proportion of safe run observed in the presence of DSBs in early block is 0.0%, while the highest observed for DSBs injected in both central & early blocks is 44.3%. Overall, we observed that the proportion of crash failures has increased when DSBs are injected in an early and central block compared when DSBs are injected in an early block only. However, this comes as a counterbalance to a corresponding decrease in SDCs when DSBs are injected in an early and central block.

We also observed that injecting DSBs in an early and central block results in very similar failure profile as when injecting DSBs in an early and late block. On the other hand, we observed that when DSBs are injected in a central and late block, the profile changes considerably. The proportion of safe runs increases while the proportion of crash failures decreases (except for the control software). Thus, with these results, we can conclude that the locations at which DSBs are injected has a strong impact on the failure profile of the system. We have shown that an early injection of a DSB error often leads to a failure.

C. Limitations

One limitation of the results presented here is the range of applications we have used to evaluate the DSB fault model. Though initial results show that the fault model can help uncover vulnerabilities that are otherwise not detected by single bit-flip errors, the fault model needs to be validated against several other applications.

A second limitation in the results presented here is that, to the best of our knowledge, there is little to no field data that shows how multiple bit upsets will manifest themselves. There is however increasing evidence that the rate of hardware errors is increasing. We only consider DSB errors here and, in our future work, we are considering multiple single bit-flip (MSB) errors. The relevance of the results presented here is only as far as the field data matches the DSB model introduced.

V. RELATED WORK

Fault injection is a widely used technique in dependability evaluation [7][12][16][17]. Hardware transient faults are injected into a target system by flipping bits in CPU registers or memory [16][17]. Recent research have shown that multiple fault injections can be very effective in detecting software vulnerabilities [7][8]. Other works have investigated impact of device-level fault injections that manifest as single bit-upsets in registers and main memory [18][19][20].

 ISBN: 978-1-61208-429-9

Recently, the effects of multiple bit-upsets on SRAMs and DRAMs have been studied. In [21], the authors investigated DRAM disturbance errors that manifests as multiple bit-upsets in memory. On the other hand, the authors of [22] investigated the geometric effects of multiple bit-upsets injected into DRAMs. The main difference between our study and these studies is the level of abstraction we focused on. The fault model under investigation in [22] is multiple bit-upsets in multiple cells within the same memory location while that under investigation in [21] is multiple bit-upsets in different memory locations. In spite of the fundamental differences between our work and theirs, they also showed higher rate of safe runs under the single bit-flip model. In addition, under the double bit-flip model, higher crash failure rate is observed. However, they reported that the proportion of SDCs is higher under the double bit-flip model, this is contrary to what our study showed. We observed lower proportion of SDCs under the variant of double bit-flip model studied here than when compared with the single bit-flip model.

Similar to our study, the authors of [8] mimicked bit-flips in registers of a real hardware platform. In addition, they investigated the impact of SBF and double bit-flips (two random bit-flips in same location) on program execution. Our study mainly differs from theirs in the assumed DBU fault model. The DBU fault model in their work selects a single location and flips two bits in that location, while in ours the model chooses two locations and flips one bit in each location. However, in [8], they also injected faults in memory words and investigated the error sensitivity for different target locations. Both works reported a higher level of safe runs for SBUs and a higher proportion of crash failures for DBUs.

VI. Conclusion and Future Work

We have investigated the impact of a novel variant of the double bit upsets, namely the double single bit-flip model, on software execution. We have evaluated it on five different modules from two different applications. Our results show that (i) the proportion of crash failures induced by DSBs is significantly higher than single bit-flip errors, (ii) the proportion of SDCs is lower with DSBs than with single bit-flips and (iii) DSBs induce different failure profiles in different applications.

As future work, we will investigate the reason behind the observed differences between this model and SBF. We will also extend the DSB model to include injection in memory words. Further, we will compare the failure profile of the DSB model with existing DBU models. We will also investigate the effectiveness of current software-based fault tolerance techniques, such as detectors, against DSBs and in turn determine the type of fault tolerance needed to handle the different types of failures. We will also generalise the work focusing on multiple single bit-flip (MSB) errors (instead of two).

References

[1] C. Yang and A. Orailoglu, "Processor reliability enhancement through compiler-directed register file peak temperature reductionprocessor reliability enhancement through compiler-directed register file peak temperature reduction," in Proceedings Dependable Systems and Networks, 2009, pp. 468–477.

[2] J.-C. Laprie, Dependability: Basic Concepts and Terminology. Springer-Verlag, December 1992.

[3] M. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," IEEE Computer, vol. 30, no. 4, April 1997, pp. 75–82.

[4] "Iso 26262-1:2011, road vehicles – functional safety – part 1: Vocabulary. iso, geneva, switzerland," 2011.

[5] 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 IEEE Symposium on VLSI Circuits, 2007, pp. 80–81.

[6] R. Reed and et al., "Heavy ion and proton-induced single event multiple upset," IEEE Transactions on Nuclear Science, vol. 44, no. 6, 1997, pp. 2224–2229.

[7] S. Winter, M. Tretter, B. Sattler, and N. Suri, "simfi: From single to simultaneous software fault injections," in Proceedings of Dependable Systems and Networks (DSN), 2013.

[8] F. Ayatolahi, B. Sangchoolie, R. Johansson, and J. Karlsson, "A study of the impact of single bit-flip and double bit-flip errors on program execution," in Computer Safety, Reliability, and Security, ser. Lecture Notes in Computer Science, F. Bitsch, J. Guiochet, and M. Kaniche, Eds. Springer Berlin Heidelberg, 2013, vol. 8153, pp. 265–276. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40793-2_24

[9] X. Li, M. C. Huang, K. Shen, and L. Chu, "A realistic evaluation of memory hardware errors and software system susceptibility," in Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 6–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855840.1855846

[10] S. Smith, "Susan version 2l," http://users.fmrib.ox.ac.uk/~steve/susan/susan2l.c, 1999, [Online; accessed 19-November-2014].

[11] MATLAB, version 8.3 (R2014a). Natick, Massachusetts: The MathWorks Inc., 2014. [Online]. Available: http://www.mathworks.co.uk/products/matlab/

[12] A. Thomas and K. Pattabiraman, "Llfi: An intermediate code level fault injector for soft computing applications," in Proceedings of IEEE Workshop on Silicon Errors in Logic, System Effects (SELSE), 2013.

[13] S. Smith, "Fault-rate llfi," https://github.com/ShadenSmith/LLFI, 2014, [Online; accessed 19-November-2014].

[14] C. Lattner and V. Adve, "Llvm: A compilation framework for lifelong program analysis & transformation," in Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization, ser. CGO '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 75–. [Online]. Available: http://dl.acm.org/citation.cfm?id=977395.977673

[15] PyYaml, "Pyyaml," http://pyyaml.org/wiki/PyYAMLDocumentation, 2011, [Online; accessed 19-November-2014].

[16] M. Hiller, A. Jhumka, and N. Suri, "An approach for analysing the propagation of data errors in software," in Proceedings of the 31st IEEE/IFIP International Conference on Dependable Systems and Networks, July 2001, pp. 161–172.

[17] G. A. Kanawati, N. A. Kanawati, and J. A. Abraham, "Ferrari: A flexible software-based fault and error injection system," IEEE Transactions on Computers, vol. 44, no. 2, February 1995, pp. 248–260.

[18] B. Sangchoolie, F. Ayatolahi, R. Johansson, and J. Karlsson, "A study of the impact of bit-flip errors on programs compiled with different optimization levels," in Dependable Computing Conference (EDCC), 2014 Tenth European, May 2014, pp. 146–157.

[19] D. Di Leo, F. Ayatolahi, B. Sangchoolie, J. Karlsson, and R. Johansson, "On the impact of hardware faults — an investigation of the relationship between workload inputs and failure mode distributions," in Proceedings of the 31st International Conference on Computer Safety, Reliability, and Security, ser. SAFECOMP'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 198–209. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33678-2_17

[20] M. Demertzi, M. Annavaram, and M. Hall, "Analyzing the effects of compiler optimizations on application reliability," in Workload Characterization (IISWC), 2011 IEEE International Symposium on, Nov 2011, pp. 184–193.

 ISBN: 978-1-61208-429-9

[21] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," SIGARCH Comput. Archit. News, vol. 42, no. 3, Jun. 2014, pp. 361–372. [Online]. Available: http://doi.acm.org/10.1145/2678373.2665726

[22] S. Satoh, Y. Tosaka, and S. Wender, "Geometric effect of multiple-bit soft errors induced by cosmic ray neutrons on dram's," Electron Device Letters, IEEE, vol. 21, no. 6, June 2000, pp. 310–312.

 ISBN: 978-1-61208-429-9

Efficient Simulation of Multiple Faults for Reliability Analysis of Analogue Circuits

Eduard Weber, Klaus Echtle
University of Duisburg-Essen
Dependability of Computing Systems
Essen, Germany
e-mail: (echtle, weber)@dc.uni-due.de

***Abstract*—Software-based fault simulation can support all abstraction levels, is flexible and allows reliability assessment at different stages in the design process. Fault diagnosis and reliability analysis are increasingly important in circuit design and determine the product's time-to-market. In this paper, we provide a new efficient method and systematic scheme for reducing the time for simulation for multiple simultaneous faults and/or multiple failure modes per element in an analogue circuit. By arranging similar multiple faults in groups, some failure classes can be interpolated with an adequate precision rather than being evaluated by time-consuming simulation. The technique can be used to perform efficient multiple fault diagnosis based on multiple fault injection. Finally, the implemented procedure is validated experimentally.**

Keywords*—*Fault simulation; fault modeling; multiple fault injection; fault diagnosis; reliability prediction

I. INTRODUCTION

Fault diagnosis of circuits is a well-developed research field with a long tradition. The first scientific publications are from early 1960s. Circuit simulation is nowadays an accepted standard in the development of electronic circuits. Small to complex analogue, digital and mixed signal circuits can be tested and verified with appropriate simulation software. A lot of progress has been made in the development of software tools for the design and verification of analogue and/or mixed-signal circuits, both in the open-source and in the commercial sector. Already two decades ago the method of analogue fault modelling has been suggested to enable both fault diagnosis and reliability evaluation. Different approaches have been developed for fault simulation of analogue and mixed-signal circuits. Previous work on analogue fault modelling focuses on parametric defects (soft faults) and catastrophic defects (hard faults). Parametric faults are typically simulated with parameter modifications, while open and short defects are dealt with via injecting a high or low resistance on transistor level, respectively. Fault simulation is generally done by injecting a fault on transistor level and analysing the circuit's behaviour by applying single DC, transient or AC simulation for linear or nonlinear circuit models. Also software tools for automatic fault injection and efficient test generation have been developed. However, mostly single faults have been considered in the past. Test cases for fault injection have been generated often by hand from an understanding of the design and fault expectations of major circuit elements. Most of the fault simulators for analogue circuits presented in the literature cover only parameter or catastrophic faults. Some tools have attempted to automate test generation and the fault simulation process for analogue circuits. Most existing fault simulators use the Simulation Program with Integrated Circuits Emphasis (SPICE) and modify SPICE net lists to represent faults [1] - [7]. The fault simulation software [8] used for the work presented in this paper defines circuit faults in Visual Basic (VB-Script) language and allows flexible and very accurate fault modelling. The main goal of this paper is to speed up the simulation for multiple faults.

II. DIAGNOSIS OF ANALOGUE CIRCUITS

Test and fault diagnosis of analogue circuits are necessary despite the ongoing digitalization. Analogue circuits are always required to form the interface to the physical environment. Analogue signals do not consist of just "low" or "high" values like in the digital field. In principle, infinite numbers of signal values are conceivable. The time and frequency characteristics of analogue signals bring another dimension, and are an additional issue within circuit assessment. The propagation of faults is more difficult than in the digital field. Typically it does not occur in just one direction, but could be from any element in all directions towards neighbour elements within the circuit. A particular fault in an element (like resistor, capacitor, transistor, etc.) does not provide explicit information about the resulting signal values. Therefore a calculation of signal values (done by circuit simulation) is always necessary. Nonlinear models, parasitic elements, charges between elements or energy-storing elements make diagnosis and reliability analysis more complex [9]. Because of these reasons, the automation level of fault diagnosis procedures for analogue circuits has not yet achieved the development level realized in the digital field. The reason for the limited automation is simply due to the nature of analogue circuits. The predominant design methodology for analogue circuits is still individual optimization of reusable topologies.

The simulation of multiple simultaneous faults is even more complex. The consideration of multiple faults is important for the following reasons. Different fault modes can be present in the elements of complex circuits. Their occurrence increases even more in rough environments. Also,

 ISBN: 978-1-61208-429-9

multiple parametric faults can be present in the field as a result of ageing, environmental stress and design errors. Moreover, multiple fault diagnosis is relevant when a new circuit design is introduced and a high failure density exists. The restriction to single fault simulation can lead to incorrect evaluation results.

One of the main issues in software-based fault simulation is the relatively long runtime in case of complex analogue circuits. In general, the runtime increases rapidly with the circuit size and the number of faulty elements (fault depth) and the failure modes per element. When performing fault simulation, the runtime is mostly determined by the number of fault injections. Each injection of a multiple fault has to be simulated separately. Usually the simulation time for single faults (at transistor level) is tractable because of available computer performance. Also the performance of Electronic Design Automation (EDA) tools has been increased during the last decade. However, multiple fault injection is a challenge with respect to runtime.

The fault simulation framework [8] used for the work presented in this paper can deal with several fault modes injected simultaneously into elements of a circuit. We consider permanent hard (open and short circuit) and soft faults (parametric faults). Please note, that even shorts and opens are dealt with as analogue (not digital) faults, because the simulator generates the analogue signal throughout the complete circuit in the case of these faults. Figure 1 shows how the total simulation time (here number of simulation runs) is influenced by the number of multiple faults and the failure modes per element. The diagram shows a medium-sized circuit example composed of 20 elements where faults are injected, each of which leads to two different failure modes. The solid line represents the number of simulation runs for all necessary test cases. This quantity increases rapidly with the number of multiple faults. The dashed line shows that the quantity of simulation runs can be reduced significantly by assuming monotonic behaviour as follows: When a set F of simultaneous faults is not tolerated, then also a superset of F will not be tolerated. Consequently the superset needs not be simulated. The assumption of monotonic behaviour is slightly pessimistic, because experience has turned out that in practice there are only few exceptions. This monotonicity does not always exist. Instead we have observed that it exists in overwhelming majority of cases with only very few exceptions.

III. STRATEGIES FOR REDUCING SIMULATION TIME

To reduce the runtime for simulation with fault injection the following two general approaches are possible: reduce the number of test cases (simulation runs) or speed up the simulation procedure for each test case. Several approaches are described in the literature to speed-up the simulation process, including fault or test case ordering [10] - [13] and distributed fault simulation [14][15]. Several approaches for multiple fault generation [16][17] and simulation [18][19] for reliability analysis are described in the literature. A general rule (if applicable) is the assumption of monotonic behaviour (see previous section). Two joint faults will not be tolerated, if at least one of them is not tolerated when injected as single fault. By "tolerated" we mean that the circuit under diagnosis (CUD) is still providing its function according to a given maximum deviation from the ideal output. The monotony assumption has the advantage that many irrelevant multiple fault combinations can be discarded before being simulated. The effect to the number of test cases (simulation runs) is quite substantial. Discarding dual faults will also result in a smaller number of considered triple faults, and so on. The simulation time is reduced for all fault depths (see Figure 1). In general, the monotony assumption reduces the number of both considered elements and failure modes per element.

In the remainder of the paper, we present a further method how the number of simulation runs can be reduced, see Sections 4 and 5. Before we describe the method we will formalize the selection of test cases to achieve a better precision in the description of the fault classes the new method is making use of.

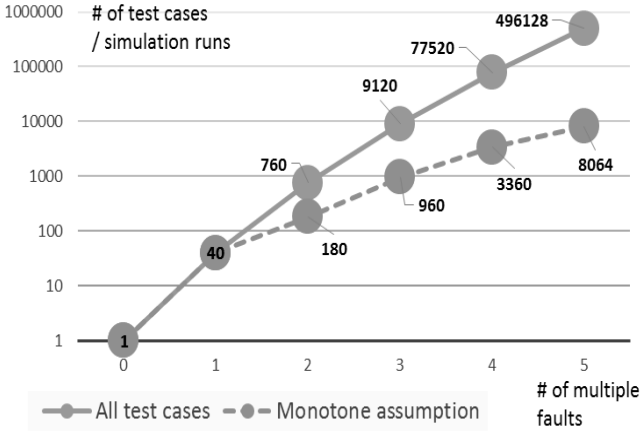


Figure 1. Complexity of fault simulation for an example medium sized circuit (20 elements with two fault modes per element).

Formally, the relationship between faults, elements of the circuit, injections and simulation runs is defined by the following tuples and functions:

1) $C = \{c_0, \dots, c_m\}$ is the set of circuits to be evaluated, $c_0 \in C$ is the fault-free circuit.
2) E = {transistor1, transisitor2, ..., resistor1, ... , … } is the set of elements of the circuit c_0.
3) F = {short_circuit, open_circuit, parameter_modification, ... } is the set of considered fault modes of the circuit c_0.
4) I = { (f, e) ∈ F × E : probability of fault f in element e} is the set of potential injections.
5) $I^* = \{ i^* \subset I : (x \in i^*, y \in i^*, x \neq y) \Rightarrow x|E \neq y|E \}$ is the set of potential multiple injections. I* is a subset of the power set of I. By x|E and y|E we denote the element of injection x or injection y, respectively. The inequality x|E ≠ y|E excludes joint injection of different faults to the same element of the circuit.

6) $Q : F \times E \rightarrow [0, 1]$
is the probability of fault $f \in F$ in a faulty element $e \in E$. If a fault $f \in F$ is not applicable to an element $e \in E$ then $Q(f, e) = 0$. For a given faulty element $e \in E$ the sum of fault probabilities is always 1:
$\Sigma_{f \in F}: Q(f, e) = 1$.

Example: If we assume only two fault modes F = {open, short} and only two elements $E = \{R_1, R_2\}$, there may be four injections $I = \{(open, R_1), (open, R_2), (short, R_1), (short, R_2)\}$ and four double injections. In all we obtain:
$I^* = \{ \{(open, R_1)\}, \{(open, R_2)\}, \{(short, R_1)\}, \{(short, R_2)\}, \{(open, R_1), (open, R_2)\}, \{short, R_1), (short, R_2)\}, \{(open, R_1), (short, R_2)\}, \{short, R_1), (open, R_2)\} \}$.
If shorts are more likely for R_1 and opens are more likely for R_2 we may get, say,
$Q(open, R_1) = 0.2$, $Q(short, R_1) = 0.8$ *(0.2 + 0.8 = 1).*
$Q(open, R_2) = 0.4$, $Q(short, R_2) = 0.6$ *(0.4 + 0.6 = 1).*

$P : E \rightarrow [0,1]$ is the function indicating the probability that element $e \in E$ is fault-free.

Function R: $I^* \rightarrow \{0,1\}$ is a simulation run with joint injection of all faults from $i \in I^*$. The method returns 1 if the injected faults are tolerated according to the tolerance criterion, otherwise 0. In the following the fault simulation procedure is described for single, double, triple, fault injection.

Single faults:
$I_1 = I$ is the set of single fault injections to be evaluated by simulation.
$T_1 = \{ i \in I_1 : R(\{i\}) = 1 \}$ is the set of single injections that have been tolerated. The function
$P_1 = \sum_{i \in I_1} R(i) \cdot (1 - P(i|E)) \cdot Q(i|F) \cdot \prod_{y \in (I_1 \setminus i)} P(y|E)$
expresses the probability of tolerated single injections.

Double faults:
$I_2 = \{\{(f, e), (f', e')\} : (f, e) \in T_1, (f', e') \in T_1, e \neq e' \}$ is the set of double injections to be evaluated by simulation. I_2 has been defined on the basis of T_1, not I_1, because the non-tolerated injections from the complement $I_1 \setminus T_1$ are excluded due to the assumption of monotony.
$T_2 = \{i^* \in I_2 : R(i^*) = 1\}$ is the set of double injections that have been tolerated.
$P_2 = \sum_{i^* \in I_2} R(i^*) \cdot \prod_{x \in i^*} (1 - P(x|E)) \cdot Q(x|F) \cdot \prod_{y \in (I_2 \setminus i^*)} P(y|E)$
expresses the probability of tolerated double injections.

Triple faults:
$I_3 = \{\{(f, e), (f',e'), (f'',e'')\} : \{(f, e), (f', e')\} \in T_2, (f'', e'') \in T_1, e \neq e', e \neq e'', e' \neq e''\}$ is the set of triple injections to be evaluated by fault simulation. Again, the non-tolerated previous injections have been excluded due to the assumption of monotony.
$T_3 = \{i^* \in I_3 : R(i^*) = 1\}$ is the set of triple injections that have been tolerated.
$P_3 = \sum_{i^* \in I_3} R(i^*) \cdot \prod_{x \in i^*} (1 - P(x|E)) \cdot Q(x|F) \cdot \prod_{y \in (I_3 \setminus i^*)} P(y|E)$
expresses the probability of tolerated triple injections.

The injections of higher numbers of joint faults are defined accordingly.

IV. FAULT CLASS ALGORITHM

Our new algorithm is an heuristic approach that is based on an observation of simulation results [8] of so-called fault classes. A fault class is a set of test cases (series of fault injections) all of which have the same number of faults and the same fault modes, independent of the elements where the faults are injected.

Experimental results show that three fault classes FC1, FC2 and FC3 for multiple faults mostly exhibit a monotonically increasing degree of tolerance, when the fault distance between FC1 and FC2 is 1, and also the fault distance between FC2 and FC3 is 1. By a fault distance d(FC, FC') (similar to the Hamming distance), we understand the number of fault modes that differ between FC and FC'. The degree t of tolerance is defined by the number of tolerated test cases divided by the number of all test cases of a fault class.

The case d(FC1, FC2) = d(FC2, FC3) = 1 means that each pair of fault classes differs by just one fault mode. For example, consider the following fault classes:
FC1 (open, open, open),
FC2 (open, open, short),
FC3 (open, short, short).
The fault distances are d(FC1, FC2) = d(FC2, FC3) = 1 and d(FC1, FC3) = 2. Typically this leads to
either $t(FC1) \leq t(FC2) \leq t(FC3)$
or $t(FC1) \geq t(FC2) \geq t(FC3)$.
From this observation we developed an algorithm that can be characterized as follows:

- Search for fault classes FC1, FC2, FC3 satisfying the condition above – or search for even longer chains of fault classes with this property.
- Determine which of the chains will typically lead to an ascending or descending degree of tolerance. To decide that, analysing the fault classes of the previous fault depth is necessary, see Step 2 of this section below.
- Quantify the tolerance of the first and the last fault class of a chain by simulation.
- Quantify the tolerance of the remaining fault classes of a chain by interpolation.

Fault classes are defined by the modes of the injected faults and their number of simultaneously injected faults. $FC_2(x, y)$ denotes a fault class for two joint injections, namely fault modes x and y. Since the fault classes $FC_2(x, y)$ and $FC_2(y, x)$ are identical, we enforce a unique notion by assuming an order among the fault modes. Since fault modes x and y may be identical (injection of two faults of identical mode into different elements), we require $x \leq y$ for $FC_2(x,y)$. For an arbitrary fault class $FC_n(x_1, x_2, \ldots, x_n)$ we require $x_1 \leq x_2 \leq \ldots \leq x_n$. Then, a fault class for double fault injection is defined as follows:

 ISBN: 978-1-61208-429-9

$FC_2(x, y) = \{ \{(f, e), (f', e')\} \in I_2 : f = x, f' = y \}$

A fault class for the injection of n faults is defined accordingly: $FC_n(x_1,\ldots, x_n) = \{\{(f_1, e_1),\ldots, (f_1, e_1)\} \in I_n : f_i = x_i\}$.

The subset of test cases in a fault class $FC_n(x_1,\ldots, x_n)$ that has been tolerated is called tolerance class $TC_n(x_1,\ldots, x_n)$. The following holds: $TC_n(x_1,\ldots, x_n) \subset FC_n(x_1,\ldots, x_n)$. Moreover, $TC_n(x_1,\ldots, x_n) = FC_n(x_1,\ldots, x_n) \cap TC_n$. The quotient of the cardinality of $TC_n(x_1,\ldots, x_n)$ and the cardinality of $FC_n(x_1,\ldots, x_n)$ is called tolerance degree $t_n(x_1,\ldots, x_n)$. Thus

$$t_n(x_1, \ldots, x_n) = \frac{|TC_n(x_1, \ldots, x_n)|}{|FC_n(x_1, \ldots, x_n)|}$$

The heuristic approach is defined in the following steps and the algorithm is shown in Figures 2 and 3. We assume that the tolerance classes $TC_1(\ldots)$ and $TC_2(\ldots)$ have already been generated by the respective fault simulations. Consequently, the tolerance degrees $t_1(\ldots)$ and $t_2(\ldots)$ are known. Then the following steps describe how the fault classes $FC_3(\ldots)$ for triple fault simulation – or interpolation! – are formed.

A. Step 1 – Generation Of Fault Classes

A fault class $FC_3(x, y, z)$ with 3 faults is generated by combining all test cases of T_2 with all test cases of TC_1 in the following way: Each union of a test case $tc_2 \in TC_2(x,y)$ and a test case $tc_1 \in TC_1(z)$ form a test case $tc_3 \in FC_3(x,y,z)$ provided x, y and z inject faults into different elements. Since we avoid double injections into a single element, the respective combined injections {x, y, z} are filtered out. The corresponding algorithm is shown in Figure 2. In the algorithm we denote the fault mode of injection x by x|F.

Procedure 1 Generate Fault Classes

```
for all test cases tc2 ∈ TC2 do
  for all test cases tc1 e TC1 do
  {  test case {x, y, z} = i ∪ j;
     if x|E ≠ y|E and x|E ≠ z|E and y|E ≠ z|E then
        FC3(x|F, y|F, z|F) = FC3(x|F, y|F, z|F) ∪ {x, y, z}
  }
```

Figure 2. Generate Fault Classes.

B. Step 2 – Search Fault Class Chains

The search of fault class chains starts with a search in TC_2. We inspect all pairs of tolerance classes $TC_2(x, y)$ and $TC_2(x', y')$ and filter out those with a fault distance of 1 and, moreover, with "significantly unequal" tolerance degrees (the difference should be at least Δ). Formally:
$d(TC_2(x, y), TC_2(x', y')) = 1$ and $|t_2(x, y) – t_2(x', y')| \geq \Delta$
where Δ may be in the range of 5% of the absolute values.

From the fault distance 1 we can conclude that either $x = x'$ or $y = y'$. In the following we assume $x = x'$ and $y \neq y'$ without loss of generality.

From the two tolerance classes $TC_2(x, y)$ and $TC_2(x, y')$ we derive the following chain of three fault classes:
$< FC_3(x, y, y),\ FC_3(x, y, y'),\ FC_3(x, y', y') >$

According to the observation of likely monotonicity (see beginning of section IV) we only simulate the test cases of the first and the last fault class in the chain to obtain the tolerance degrees $t_3(x, y, y)$ and $t_3(x, y', y')$, respectively. The tolerance degree $t_3(x, y, y')$ of the inner fault class in the chain is obtained by interpolation:
$t_3(x, y, y') = (t_3(x, y, y) + t_3(x, y', y')) / 2$.
The algorithm can be seen from Figure 3.

Procedure 2 Search Fault Class Chains

```
for all pairs (TC2, TC2') of tolerance classes with two injections do
  if d(TC2(x, y), TC2(x', y')) = 1 and |t2(x, y) – t2(x', y')| ≥ Δ then
  { fault class FC = FC3(x, y, y),
    fault class FC' = FC3(x, y, y'),
    fault class FC'' = FC3(x, y', y');
    t3(x, y, y) = simulation of FC3(x, y, y);
    t3(x, y', y') = simulation of FC3(x, y', y');
    t3(x, y, y') = (t3(x, y, y) + t3(x, y', y')) / 2;
  }
```

Figure 3. Search Fault Class Chains.

C. Step 3 – Calculation of Probabilities

The simulations of $FC_3(x, y, y)$ and $FC_3(x, y', y')$ deliver the set of all tolerated test cases, this means the two tolerance classes $TC_3(x, y, y)$ and $TC_3(x, y', y')$. The probability of tolerating the respective triple faults can be calculated by the formula presented in section III. When this formula is applied to tolerance class $TC_3(x, y, y)$ we obtain

$$\sum_{i^* \in TC_3(x,y,y)} \prod_{x \in i^*} (1 - P(x|E)) \cdot Q(x|F) \cdot \prod_{y \in (TC_3(x,y,y) \setminus i^*)} P(y|E)$$

For tolerance class $TC_3(x, y', y')$ we obtain:

$$\sum_{i^* \in TC_3(x,y',y')} \prod_{x \in i^*} (1 - P(x|E)) \cdot Q(x|F) \cdot \prod_{y \in (TC_3(x,y',y') \setminus i^*)} P(y|E)$$

The probability of tolerating the triple faults of the interpolated fault class cannot be obtained directly, because the test cases of this class have not been simulated. For this reason we approximate the probability by multiplying the respective formula with the tolerance degree:

$$t3(x, y, y') \cdot \sum_{i^* \in TC_3(x,y,y')} \prod_{x \in i^*} (1 - P(x|E)) \cdot Q(x|F) \cdot \prod_{y \in (TC_3(x,y,y') \setminus i^*)} P(y|E)$$

The tolerance class of the non-simulated fault class is generated by selecting a portion of $t_3(x, y, y')$ test cases at random. For the injection of more than three joint faults, steps 1 to 3 can be applied accordingly.

V. Experimental Results

In this section, the efficiency of the proposed solution to reduce the simulation time is evaluated. The fault simulation framework [8] is used to evaluate the dependability of four example electronic circuits. It should be noted, that for used circuits only permanent faults (e.g. short, open or parameter deviations) have been considered.

The simulation time (fault injection and simulation) depends on the number of elements, the number of injected

 ISBN: 978-1-61208-429-9

TABLE I. COMPARISON OF EXPERIMENTAL RESULTS

Circuit name	No. of simulation runs			Speed-up factor	Error
	Number of simulation runs for all possible fault combinations	Number of simulation runs with monotonicity assumption	Number of simulation runs for the new approach with fault classes	Our approach over simulation with monotonicity assumption	Our approach over fault simulation with monotonicity assumption
Two stage BJT amplifier with feedback (Fault depth 1-4)	22422	356	284	1.25	5.4 %
LM741 AMP [20] (Fault depth 1-4)	3923175	2090	1612	1.30	0.6 %
Broadband VHF/UHF amplifier [21] (Fault depth 1-3)	695525	18187	10928	1.66	1.8 %
Limiter BSP [22] (Fault depth 1-4)	1045256	1208	758	1.59	2.7 %
				Average: 1.45	Average.: 2.62 %

faults per element and the fault depth. Appropriate fault tolerance criteria have been defined on circuit outputs.

All of the circuits have been evaluated twice: The first evaluation was without generation of fault classes (chains have not been formed and all test cases have been simulated with the monotonicity assumption). The second evaluation applied the new method with fault classes (only a portion of the test cases has been simulated). The remaining ones have been evaluated by interpolation according to the algorithm in steps 1 to 3). This way the new method can be compared directly to the solution without fault classes.

The result is shown in table 1. The last but one column shows the speedup achieved by the new approach: 45% in the average. It has to be paid by an error in the results (see last column). The error refers to the number of tolerated test cases. A deviation of 2.62% has been noticed in the average.

VI. CONCLUSION

Fault simulation of analogue circuits with multiple faults is an important problem to deal with, since multiple faults appearance is unavoidable in real systems. In this paper we have introduced the fault class concept for our approach to reduce the simulation time of multiple fault analysis. We discussed the idea of faults classes, providing conditions that ensure chains of fault classes with ascending or descending degree of tolerance. We implemented the procedure and evaluated it experimentally.

In this paper, we have successfully reduced the duration of software-based fault simulation for multiple faults and different fault modes. In the evaluated example circuits, our methodology shows that the number of simulation runs is significantly lower while preserving the precision quite well.

REFERENCES

[1] Z. R. Yang and M. Zwolinski, “Fast, robust DC and transient fault simulation for nonlinear analogue circuits,” in *Design, Automation and Test in Europe Conference and Exhibition 1999. Proceedings*, 1999, pp. 244–248.

[2] J. Jagodnik and M. Wolfson, “Systematic fault simulation in an analog circuit simulator,” vol. 26, no. 7, pp. 549–554, 1979.

[3] Y. Cao, Z.-h. Cen, J.-l. Wei, X. Ma, B. Yang, and M. Li, “FDSAC-SPICE: fault diagnosis software for analog circuit based on SPICE simulation,” in *International Conference on Space Information Technology 2009*: SPIE, 2010, pp. 765120–765120-8.

[4] C. Sebeke, J. P. Teixeira, and M. J. Ohletz, “Automatic fault extraction and simulation of layout realistic faults for integrated analogue circuits,” in *the European Design and Test Conference. ED&TC 1995*, pp. 464–468.

[5] S. Spinks, “ANTICS analogue fault simulation software,” in *IEEE Colloquium on Testing Mixed 23 Oct. 1997*, p. 13.

[6] Bernd Straube, Bert Müller, Wolfgang Vermeiren, Christoph Hoffmann, Sebastian Sattler, “Analogue Fault Simulation by aFSIM,” *DATE 2000 - User Forum, Paris*, 2000.

[7] H. Spence, “Automatic analog fault simulation,” in *Conference Record. AUTOTESTCON '96*, pp. 17–22.

[8] Weber E, Echtle K, and 52nd IEEE International Reliability Physics Symposium, IRPS 2014, “Simulation-based reliability evaluation for analog applications,” (English), *IEEE Int. Reliab. Phys. Symp. Proc. IEEE International Reliability Physics Symposium Proceedings*, 2014.

[9] P. Kabisatpathy, A. Barua, and S. Sinha, *Fault Diagnosis of Analog Integrated Circuits*. Boston, MA: Springer, 2005.

[10] Junwei Hou and A. Chatterjee, “Concurrent transient fault simulation for analog circuits,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst*, vol. 22, no. 10, pp. 1385–1398, 2003.

[11] P. N. Variyam and A. Chatterjee, “FLYER: fast fault simulation of linear analog circuits using polynomial waveform and perturbed state representation,” *Tenth International Conference on VLSI 4-7*, pp. 408–412, 1997.

[12] A. V. Gomes, R. Voorakaranam, and A. Chatterjee, “Modular fault simulation of mixed signal circuits with fault ranking by severity,” *IEEE International Symposium on Defects and Fault Tolerance in VLSI Systems*, pp. 341–348, 1998.

[13] H. Hashempour, J. Dohmen, B. Tasić, B. Kruseman, C. Hora, M. van Beurden, and Yizi Xing, “Test time reduction in analogue/mixed-signal devices by defect oriented testing: An industrial example,” *Design, Automation & Test in Europe*, 2011.

 ISBN: 978-1-61208-429-9

[14] T. Markas, M. Royals, and N. Kanopoulos, “On distributed fault simulation,” *Computer*, vol. 23, no. 1, pp. 40–52, 1990.

[15] C. P. Ravikumar, V. Jain, and A. Dod, “Faster fault simulation through distributed computing,” *Tenth International Conference on VLSI*, pp. 482–487, 1997.

[16] S. Kajihara, T. Sumioka, and K. Kinoshita, “Test generation for multiple faults based on parallel vector pair analysis,” *International Conference on Computer Aided Design (ICCAD)*, pp. 436–439, 1993.

[17] H. H. Zheng, A. Balivada, and J. A. Abraham, *A Novel Test Generation Approach for Parametric Faults in Linear Analog Circuits: Proceedings / 14th IEEE VLSI Test Symposium, Princeton, New Jersey*. Los Alamitos, Calif: IEEE Computer Society Press, 1996.

[18] K. Saab, N. Ben-Hamida, and B. Kaminska, “Parametric fault simulation and test vector generation,” *Meeting on Design Automation*, pp. 650–656, 2000.

[19] Yong Chang Kim, V. D. Agrawal, and K. K. Saluja, “Multiple faults: modeling, simulation and test,” *7th Asia and South Pacific Design Automation Conference*, pp. 592–597, 2002.

[20] National Semiconductor, *LM741 Operational Amplifier*. Available: http://web.mit.edu/6.301/www/LM741.pdf (2015, Mar. 05).

[21] C. G. Gentzler and S. K. Leong, “Broadband VHF/UHF amplifier design using coaxial transformers,” *High Frequency Electronics*, pp. 42–51, http://www.polyfet.com/HFE0503_Leong.pdf and https://awrcorp.com/download/faq/english/appnotes/uhf_vhf_amplifier.aspx, 2003.

[22] AWR Corporation, *Bipolar Limiting Amplifier Circuit.* Available: https://awrcorp.com/download/faq/english/docs/Getting_Started/Tonal_Analysis.html (2015, Mar. 05).

 ISBN: 978-1-61208-429-9

Reducing the Communication Complexity of Agreement Protocols By Applying A New Signature Scheme called SIGSEAM

Omar Bousbiba
Dependability of Computing Systems
University of Duisburg-Essen
Essen, Germany
bousbiba@dc.uni-due.de

***Abstract*—Distributed computing systems need agreement protocols when global consistency must be achieved in a fault-tolerant way. However, solving the Byzantine agreement problem in an efficient way in terms of communication complexity is still a challenging task. In synchronous systems with stringent time requirements not only the fault tolerance, but also the limitation of the communication complexity are crucial for practical usability. Many agreement protocols use digital signatures. This paper presents a novel signature generation technique to merge several signatures into a single one. This advantage opens a design space for agreement protocols with significantly reduced message overhead. Moreover, the new signature technique can also be applied to existing agreement and/or consensus protocols (Turquois and ESSEN, for example) without affecting the fault tolerance properties of the protocol.**

Keywords— Malicious Byzantine Faults; Agreement protocols; Digital Signatures for Fault Tolerance.

I. INTRODUCTION

Distributed systems are becoming more and more important in our electronic society. In case of safety relevance, it is important to make these systems resilient against faults. Fault tolerance techniques can be applied to increase various dependability properties. Many real-time applications require fail-operational behaviour. Take a fail-safe brake-by-wire system as an example. It has to provide its functionality all the time. In the presence of a fault, the four-wheels braking is reduced to diagonal-wheel braking. Consequently, a decision has to be taken which pair of wheels has to be passivated (in a non-blocking way, of course).

The agreement problem is recognized as a fundamental element in fault-tolerant distributed computing (i.e., safe brake, collision avoidance, semiautomated vehicles, etc.). The problem has been known for decades as Byzantine agreement (BA) [1][2]. In order to solve it, two conditions have to be satisfied, known as interactive consistency (IC):

IC1: All fault-free nodes obtain exactly the same view

IC2: The information provided by a fault-free node is part of this view.

Due to its paramount importance, the problem has attracted a great deal of attention in the past. It has been investigated extensively and many solutions have been proposed. Many of the approaches [3][4][5] focused on reducing the communication complexity in terms of the number of messages, the number of nodes (related to the number of faults to be tolerated), and required storage.

Signature techniques contribute a lot to a reduction in communication complexity, because they protect the origin of the message against undetectable corruption when the message is forwarded from node to node [2].

Typical sequences of actions during the execution of an agreement protocol are the following ones:

1. Send a signed message to one/more neighboring node(s)
2. Forward a message from node to node(s), where each forwarding node cosigns the message
3. Collect incoming messages (which can be numerous) including signature checks
4. a) Take a local decision on the message to be sent in the next round, b) termination with some value or a consistency vector [3][4].

The steps 1 to 4 may be repeated several times, depending on the particular protocol.

Typically, the following situation occurs frequently: In some phases, a node X receives different messages $M_1,\ldots,M_k$ all of which it has to forward to a neighbor node Y. If all the messages $M_1,\ldots,M_k$ have been signed by different nodes $N_1,\ldots,N_k$ and all nodes contain identical payload contents A (see Figure 1), then node X cannot summarize the messages and send only one message with payload contents A to node Y, because the signatures would be lost then. Instead X has to forward the k messages separately (in some protocols it is sufficient to filter out a subset of the messages).

Consequently, a signature mechanism which allows messages to be merged has the potential to greatly reduce the communication overhead of an agreement protocol. Figure 1 illustrates an example of the idea behind signature merging.

 ISBN: 978-1-61208-429-9

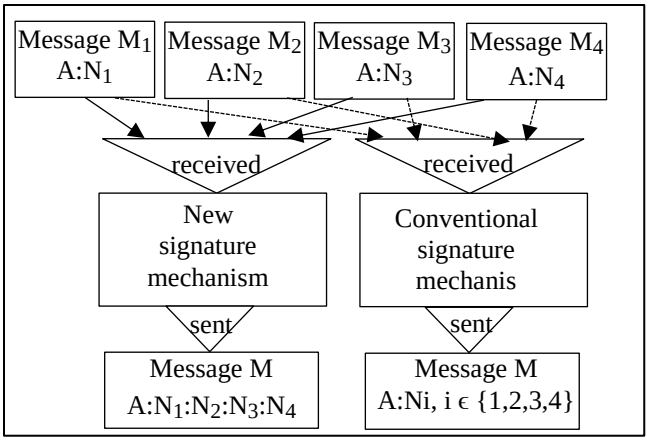


Figure 1. (left) new (right) conventional - signature mechanism.

A. Contribution and Outline

The goal of this work is the provision of a novel signature mechanism, which opens an extended design space for agreement protocols with lower communication complexity in terms of message transmissions. By signature merging, the number of messages, and thus the overall transmitted information can be reduced. The new method does not use cryptographically strong signatures. Instead, the signatures are designed to withstand faults, even with Byzantine behaviour, but not intelligent attacks of humans. Besides the (very short) computation time for signature merging, the protocol does not need extra time for reaching agreement.

The rest of the paper is organized as follows: Section II characterizes the agreement protocols relevant for this paper. The new signature method is presented in Section III and its application to agreement protocols in Section IV. The improvement is shown in Section V by providing a quantification of the overhead. A summary and an outline of ongoing work are given in Section VI.

II. CONSIDERED AGREEMENT PROTOCOLS

A. Protocols

Since the time when the agreement problem was introduced by Lamport et. al. [1], many solutions have been proposed. Most of the work is focused on reducing the number of messages, the required number of nodes per tolerated fault and the storage consumption.

It has turned out that signatured protocols need significantly less messages. However, without signature merging there is a limitation to further reduction. In this paper, the merging approach is applied to two protocols: Turquois [3] and ESSEN [5]. For each of these protocols a variant is derived that takes benefit of signature merging.

Turquois is a protocol which solves the consensus problem in asynchronous systems composed of **n** ad hoc nodes where a subset **f** (with $f < \frac{n}{3}$) of them can fail in an arbitrary manner. It is the first work which addresses the problem of reaching consensus in the presence of omission faults. However, Turquois solves the problem at the expense of a relatively high communication and storage overhead. The high number of message transmissions is caused by the message validation process. In the worst case, a node has to transmit more than $\frac{n+f}{2}$ messages received from previous round(s). A signature technique has a great impact on the message and storage overhead as will be shown later in this paper.

ESSEN is a protocol that solves the Byzantine agreement problem even in the presence of "malicious cooperation" faults. This means two faulty nodes may "secretly" exchange their information, such as keys and signed messages. In ESSEN, the communication complexity is very low for up to four arbitrary faults. The protocol requires a fully synchronous system (clock synchronization is presupposed). The message storage consumption is the space of only three messages. The required number of nodes grows quadratically with the number of tolerated faults. As with Turquois, the protocol uses signatures without merging functionality. The benefit of adding a signature scheme with merging capability will be shown later in this paper.

B. Signatures

For the purpose of fault tolerance, cryptographically strong signatures are not needed, because the signatures serve as countermeasures against "stupid faults" rather than "intelligent attacks". Consequently, signatures with relatively low computation time can be used (as reported in [7]). Signature techniques greatly improve the communication complexity of agreement/consensus protocols. However, when using an existing signature technique [7][8] a receiver has only two options to deal with after a message has been received. Either each received signature is stored separately, as is done in Turquois, or some kind of filter mechanism is applied (e.g., only the message with the highest number of signatures is stored). In both cases the overhead for both message storage and communication may become high.

By the proposed signature merging technique the receiver(s) get the opportunity to combine the messages into a single one without affecting the information about the signature source. This means, the new message will still contain the information of all signature sources (as shown in Figures 1 and Section III).

III. NEW SIGNATURE SCHEME SIGSEAM

The proposed signature scheme is intended to withstand arbitrary technical faults rather than intelligent attacks. For the purpose of fault tolerance simple signature generation methods are sufficient [6][7][8]). The signature technique presented in this paper is based on a multiplication scheme similar to [7]. It achieves almost the same effectiveness as

 ISBN: 978-1-61208-429-9

[7]. A thorough investigation on the new signature merging mechanism is still work in progress. Next, the algorithms for signature generation and checking are presented in detail.

All calculations are done modulo **m**, where **m** is set to a power of two (**m** = 2^x with x ϵ $\mathbb{N}$). Typical values may be m = 2^{16} or m = 2^{32}. The generation of private and public signature keys is done as follows: Each node chooses two arbitrary natural numbers **a, b** from $\mathbf{m_{odd}}$ = [m/16, m/2] $\cap$ $\mathbb{N}_{odd}$. The product **c** = **a** ▪ **b** is calculated. The value of parameter **a** is used as private key. The pair (**b**, **c**) is taken as public key, which is publically distributed to all nodes to be used for signature checking.

Signature generation: The signature of the original sender of a message is calculated over the payload data **d** and the sequence number **n** (e.g., the sequence number is changed from round to round) only. The following signature function σ_0 and a usual CRC function are used by the first signing node (e.g., source node, indexed with zero):

$$\sigma_0(\mathrm{n},\mathrm{d}) \coloneqq \mathrm{CRC}(\mathrm{n},\mathrm{d}) \cdot \mathrm{a_i}. \quad (1)$$

Cosignature generation of a forwarding node is done as follows: The cosignature value is calculated over the signature value σ_j with j $\geq$ 0 by applying the following cosignature function σ_i. The index represents the number of signing and/or cosigning nodes:

$$\sigma_i \coloneqq \begin{cases} \sigma_j + \mathrm{CRC}(\mathrm{n},\mathrm{d}) \cdot (\mathrm{a_i}+1), & \text{if j is even} \\ \sigma_j + \mathrm{CRC}(\mathrm{n},\mathrm{d}) \cdot (\mathrm{a_i}-1), & \text{otherwise} \end{cases}, \quad (2)$$

where j < i. Depending on the number of signatures in σ_j the secret key a_i of the cosigning node is used as either (a_i + 1) or (a_i – 1) depending or whether or not the number of already added (co-) signatures is even.

Compared to usual (co-) signature schemes there is an important point: In the proposed merging signature scheme the new cosignature σ_i replaces the existing (co-) signature in a message to be forwarded. However, the indices of all signing nodes are kept in the message. Thus, there is a list "Who has signed?" in each message.

The signature value signed by j nodes can be expressed by the following sum function:

$$s = \mathrm{CRC}(\mathrm{n},\mathrm{d}) \cdot \left((\mathrm{j}+1) \bmod 2 + \sum_{\mathrm{i}=0}^{\mathrm{j}} \mathrm{a_i}\right). \quad (3)$$

A receiver uses the following signature check function τ(n, d, s) after reception of a message with number n, payload data d and signature s. The check is passed if the following equation is correct:

$$s \cdot \prod_{i=0}^{j} b_i = \quad (4)$$

$$\mathrm{CRC}(\mathrm{n},\mathrm{d}) \cdot \left(\mathrm{e} \cdot \prod_{i=0}^{j} b_i + \sum_{i}^{j} \left[c_i \cdot \prod_{k=0,k\neq i}^{j} b_k \right]\right).$$

If **s** has been signed by an odd number of nodes, then parameter e is set to zero. Otherwise parameter e is set to one. In case of an odd number of nodes having (co-) signed the message we obtain:

$$s \cdot \prod_{i=0}^{j} b_i = \mathrm{CRC}(\mathrm{n},\mathrm{d}) \cdot \sum_{i=0}^{j} \left(\mathrm{c_i} \cdot \prod_{k=0,k\neq i}^{j} b_k\right) \quad (5)$$

$$s \cdot \prod_{i=0}^{j} b_i = \mathrm{CRC}(\mathrm{n},\mathrm{d}) \cdot \left((\mathrm{j}+1)\bmod 2 + \sum_{\mathrm{i}=0}^{\mathrm{j}} \mathrm{a_i}\right) \cdot \prod_{i=0}^{j} b_i \Leftrightarrow$$

$$s \cdot \prod_{i=0}^{j} b_i = \mathrm{CRC}(\mathrm{n},\mathrm{d}) \cdot \sum_{\mathrm{i}=0}^{\mathrm{j}} \mathrm{a_i} \cdot \prod_{i}^{j} b_i \Leftrightarrow$$

$$s \cdot \prod_{i=0}^{j} b_i = \mathrm{CRC}(\mathrm{n},\mathrm{d}) \cdot \sum_{i=0}^{j} \mathrm{a_i} \cdot \mathrm{b_i} \cdot \prod_{k=0,k\neq i}^{j} b_k \Leftrightarrow$$

$$s \cdot \prod_{i=0}^{j} b_i = \mathrm{CRC}(\mathrm{n},\mathrm{d}) \cdot \sum_{i=0}^{j} \left(\mathrm{c_i} \cdot \prod_{k=0,k\neq i}^{j} b_k\right) \text{ Q. E. D.}$$

The implications from right to left are obvious. The implication from left to right based on the same conclusion, as shown in [8]. The proof is done by contradiction: Due to the fact that all calculations are done modulo m (**m** = 2^x with x ϵ $\mathbb{N}$) for b ϵ m_{odd} the product $s \cdot \prod_{i=0}^{j} b_i$ returns a unique value in modulo m. However, parameter b is odd and the only prime factor of m is 2. Consequently, 2 must be a prime factor of value b ϵ m_{odd} (contradiction) Q.E.D.

IV. MODIFIED AGREEMENT PROTOCOL

A detailed explanation of the two protocols ESSEN and Turquois can be found here [3][5]. In the following, only the parts of the algorithm which have been modified are discussed in detail. The modified protocol variants are called SEAM and Turquois*, respectively.

A) SEAM

Storing of received messages: A data message is stored in the secondary buffer, when (in addition to the four conditions given in [5]) also the following two conditions are satisfied:

1. The node is member of group ExtG (see [5])
2. The node has not transmitted a message yet.

Otherwise, if all six conditions are not satisfied, the data message is rejected (for more details see [5]).

Merging of signatures: The messages in the primary and the secondary buffer are merged, iff both messages contain at least 2f – 2 (parameter f indicates the number of tolerated faults) different signature sources. The content of the

secondary buffer is deleted after transmission (regardless of whether or not the message has been merged). Moreover, in contrast to [5], only the primary and default buffer are used for the final decision. All other parts of the algorithm remain unchanged.

B) Turquois*

Merging of signatures: In each round, all messages with identical content are merged (instead of stored separately). This means not more than two different messages are stored within a round. All other parts of Turquois remain unchanged. Both variants SEAM and Turquois* have been simulated for up to 30 nodes and up to 10^9 rounds. The number of simultaneously faulty nodes has been limited to 6 in case of SEAM and 9 in case of Turquois (for more details see [3][5]). In all simulation runs the interactive consistency was fully preserved.

V. Complexity of the investigated protocols by Using SigSeam

In this section, the communication complexity in terms of redundant nodes as well as message transmission overhead is quantified by simulation. The modified protocols SEAM and Turquois* are compared with their original versions ESSEN and Turquois, respectively. The outcomes are shown in Figure 2. Summarizing the results it can be said that the new signature technique greatly improves the communication complexity of both protocols. In case of ESSEN the number of redundant nodes as well as the number of required message transmissions has been reduced from $1 + \frac{f^2+f}{2} + \left\lfloor \frac{(f-1)}{2} \right\rfloor$ down to $\frac{3f+2\left\lceil \frac{f-1}{2} \right\rceil + \left\lceil \frac{f^2}{2} \right\rceil}{2}$. In case of Turquois, the high number of $(3f+1)\frac{(n+f+2)}{2}$ message transmission (worst case) has been reduced to a constant of 3 messages per node, whereas the number of required nodes remains unchanged. This means 9f + 3 messages in all.

VI. Conclusion and Future work

The simulation results have clearly shown that the proposed signature technique with merging functionality significantly improves the efficiency of agreement protocols and does not affect the time taken to reach agreement.

The work on signature merging is still in progress. The coverage of special fault cases affecting the signatures themselves must be evaluated in detail. Besides bursts, bit flips, wrong data, also signature-related faults like copy-and-paste of signatures between messages, etc., have to be assessed with respect to the achieved coverage. Moreover, these results will be compared with 16-bit or 32-bit "light" versions of existing cryptographic signature techniques.

Acknowledgment

The author gratefully acknowledges the helpful discussions with Prof. Klaus Echtle who participated in the development of the concept of signature merging.

References

[1] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults", JACM, vol. 27, Apr. 1980, pp. 228–234, doi: 10.1145/322186.322188.

[2] L. Lamport, R. Shostak, and M. Pease, "The byzantine generalsproblem", ACM and TOPLAS, vol. 4, July 1982, pp. 382–401, doi:10.1145/357172.357176.

[3] H. Moniz, N. F. Neves, and M. Correia, "Turquois: Byzantine consensus in wireless ad hoc networks", IEEE, DSN, Chicago, IL, pp. 537-546, June 2010, pp. 537–546, doi: 10.1109/DSN.2010.5544268.

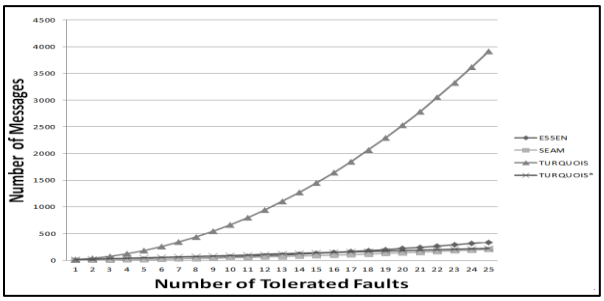
[4] M. Jochim and T. M. Forest, "An Efficient Implementation of the SM Agreement Protocol for a Time Triggered Communication Systems", SAE International Journal of Passenger Cars - Electronic and Electrical Systems, vol. 3, 2010, pp. 106-116, doi:10.4271/2010-01-2320.

[5] O. Bousbiba, "ESSEN - An Efficient Single Round Signature Protected Message Exchange Agreement Protocol for Wireless Distributed Networks", ACRS 28th, Workshop Proceedings, March 24 - 27, 2015, Porto, Portugal, ISEP, Berlin: VDE Verl., ISBN: 978-3-8007-3657-7.

[6] K. Echtle and T. Kimmeskamp, "Fault-Tolerant and Fail-Safe Control Systems Using Remote Redundancy", ARCS 22th, Workshop Proceedings, March 11, 2009, Delft, The Netherlands, Berlin: VDE Verl., ISBN: 978-3-8007-3133-6.

[7] L. Martin, "Relative signatures for fault tolerance and their implementation", Dependable Computing – EDCC-1, Lecture Notes in Computer Science, vol. 852, Oct. 1994, pp. 561–580, doi:10.1007/3-540-58426-9_158.

[8] K. Echtle, "Avoiding Malicious Byzantine Faults by a New Signature Generation Technique", Depenable Computing – EDCC3, Lecture Notes in Computer Science, vol. 1667, Sept. 1999, pp. 106-123, doi:10.1007/3-540-48254-7_9.



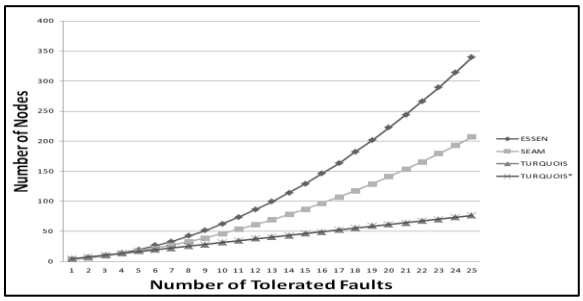


Figure 2. The idea of signature merging. Communication complexity: (a) message transmission overhead (b) required number of redundant nodes.

Dependability of Active Emergency Response Systems

Jane W. S. Liu
Institute of Information Science
Academia Sinica
Taipei, Taiwan
e-mail: janeliu@iis.sinica.edu.tw

Edward T. H. Chu
Computer Science and Information Engineering
National Yunlin Science and Tech. University
Yunlin, Taiwan
e-mail: Edwardchu@yuntech.edu.tw

***Abstract*— Recent technological and infrastructure advances along several fronts have enabled smart embedded devices, systems and applications that can enhance preparedness of our living environments against common natural and man-made disasters. They can also help us to be safer when disasters strike. This paper first discusses issues in configurability, maintainability and safety specific to this type of smart things and systems. It then describes models and tools for assessing their effectiveness and ensuring their safety.**

Keywords - disaster preparedness and response; system safety; cyber-physical elements; simulation environment; testbed

I. INTRODUCTION

The term *Active Emergency Response Systems* (AERS) [1] refers to systems of smart embedded devices and mobile applications that can process standard-compliant disaster alert messages from authorized senders and respond by taking appropriate actions to prevent loss of lives, reduce chance of injuries and minimize property damages and economical losses when the forewarned disaster strikes. We call such devices and applications *iGaDs* (*intelligent Guards against Disasters*) collectively [2]-[4]. Examples of iGaDs include smart devices that shut natural gas intake valves and turn off electricity to prevent fire, open doors to ease evacuation, bring elevators to the ground floor, turn on hazard flashers and warn the drivers of trucks and cars on highways, and deliver location-, environment- and situation-specific alerts and instructions to people via their mobile devices upon receiving an alert of a strong earthquake.

iGaDs and AERS have been made feasible in developed regions by recent advances along four directions: First, advances in sensor and analysis technologies have enabled the predication and detection of common types of natural disasters and issuance of accurate early warnings about them. For example, in developed countries frequented by earthquakes, systems of strong motion sensors networked via RF links with computers running analysis tools can generate early warnings of strong earthquakes within second(s) of their occurrences, providing receivers in affected areas with warnings, often second(s) before ground motion starts.

The second enabler is Common Alert Protocol (CAP) for encoding alert messages [5]. The OASIS standard has been adopted in US, Canada, Australia and parts of Asian Pacific region, including Taiwan and Japan. Being XML-based, CAP alert messages can be processed automatically by smart devices and applications. Hereafter, we assume that all alert messages are in CAP format and sometimes call iGaDs CAP-aware devices, systems or applications.

Third, iGaDs and AERS are enabled by platforms for receiving and authenticating CAP-compliant alerts from alerting authorities and then broadcasting them. An example is Integrated Public Alert and Warning System (IPAWS) - OPEN [6], which has been operational in USA and Canada since 2011 [6]. IPAWS-OPEN and similarly platforms in other parts of the world enable CAP alerts to be disseminated via multiple communication pathways, including broadcast channels, cellular broadcast and Internet.

The fourth enabler is Building Information Models (BIM) [7] and associated digital data exchange standards. BIM has been adopted increasingly more widely. The integration of BIM with facility management and building automation systems (e.g., [8] [9]) has enabled the systems to provide 3D-4D data on buildings and their facilities, interior layouts, and so on that are vital to support decisions of individual iGaDs in their choices of protective actions.

To illustrate this, Figure 1 shows an earthquake scenario: A strong earthquake alert in CAP format is issued by Central Weather Bureau, the agency authorized to issue such alerts in Taiwan. Today, earthquake alerts are sent directly to safety equipment of power plants, trains and fabrication lines. Alerts are also sent to Emergency Alert Services (EAS) and mobile alert services, including Google Public Alerts. These services in turn warn the general public. Limitation in human's ability to react in time and the lack of specific instructions limit the effectiveness of the warnings.

Figure 1. A earthquake scenario illustrating active use of alert [2]

 ISBN: 978-1-61208-429-9

Our white paper [2] advocates an alternative: Broadcast the alerts in the original CAP format directly to iGaDs pervasively deployed throughout our living environment. CAP-aware embedded devices can respond with humanly impossible speed to make the environment safer in ways illustrated by the examples mentioned earlier and shown in the lower right corner of Figure 1. CAP-aware mobile applications can instruct people how to stay safe based the seismic codes of buildings, interior layouts, and furnishings around them. Indeed, if such applications were available at the time of 2011 5.8 Virginia Earthquake [10], most people from New York City to Washington DC would be instructed to stay where they were: That is, do not evacuate. The chaos and economic loss occurred on the day could be avoided.

From this and other scenarios [2], one can see that iGaDs are mission critical. Ubiquitous iGaDs are Internet of Things (IoTs), and AERS containing iGaDs and remote and local sensors are cyber-physical systems. So, the title "No dependability, no internet of things" of the article [11] published by Newsroom Editor of European Commission is applicable to iGaDs/AERS. Challenges in making them adaptable and dependable, unless satisfactorily overcome, are roadblocks to their becoming pervasive elements of future disaster prepared smart living environment.

Following this introduction, Section II presents related work on dependability of IoTs and cyber-physical system in general and discusses dependability issues specific to iGaDs and AERS. To date, the results of our work include iGaDs and AERS prototypes built for proof of concept purposes and as solutions of configurability and adaptability problems. They are described in Section III. Safety is an important dependability requirement of iGaDs and AERS. Section IV describes our current and future work on models and tools for assessing the safety of AERS containing a large number of diverse iGaDs. Section V summarizes the paper.

II. Related Work

The above-mentioned statement on dependability of IoTs [11] and similar observation by researchers and developers worldwide have motivated vast efforts on IoT dependability. Examples of recent results include mechanisms and protocols for enhanced availability and reliability of IoTs and networks and middleware in applications/services built from them [12]-[14]. Other efforts (e.g., [15]-[18]) aim at providing frameworks, tools, benchmarks to support the design, implementation and assessment of dependable IoT applications and cyber-physical systems. These applications and systems, including AERS, have long lifetime. Support infrastructures, including tools for maintenance and upgrade, need to be put in place (e.g., in [19]) to ensure non-disruptive operations of existing devices and systems as they adapt to inevitable changes in message delivery platforms, message format standards, security mechanisms, and technological advances during their lifetime.

Our work on the dependability of iGaDs and AERS has the same general goal as these related efforts. We leverage existing solutions as much as possible. Section III will present examples. By doing so, we can better focus on dependability issues specific to iGaDs and AERS.

A focal point of our current effort is safety of AERS that contain vast numbers of diverse iGaDs and local sensors (e.g., intelligent emergency evacuation systems for large and complex buildings). To explain the challenges, we note that an iGaD may need to process at the same time multiple types of alerts (e.g., a strong earthquake alert for the region and a local fire or flash flood alarm) that call for conflicting responses (e.g., open all doors and close some doors, respectively). Alerts may be cancelled and reissued as conditions changes. Even most advanced disaster prediction and detection systems may issue false alarms and have missed detections. Protocols for handling such events need to be put in place, however rarely they may happen. Even when all alert messages arrive correctly and in time and all devices function correctly, the combinations of their actions may lead to catastrophic consequences.

Section IV will further elaborate issues related to safety of AERS and present our current work on building an extensible simulation framework, called AERS Simulation Framework (AERS-SF). The framework is agent-based. It resembles many existing toolkits (e.g., [20]-[22]) for the development of agent-based applications in their use of agents as model elements. Existing safety studies and emergency and disaster simulators (e.g., [23]-[25]) typically consider specific kind of emergency (e.g., fire) in a specific environment (e.g., in high rises or planes). In contrast, AERS-SF aims to provide models, tools and benchmarks needed to support simulation of diverse AERS in diverse operating environments and disaster scenarios for sake of assessing safety of AERS throughout their development.

III. Configurable and Adaptable Prototypes

Thus far, our work aims to demonstrate the concept of configurable and adaptable AERS [1]-[4] for homes, office buildings, and large public places. They contain diverse iGaDs capable of responding to alerts of natural disasters affecting the region in general, as well as alerts of emergency conditions within the building.

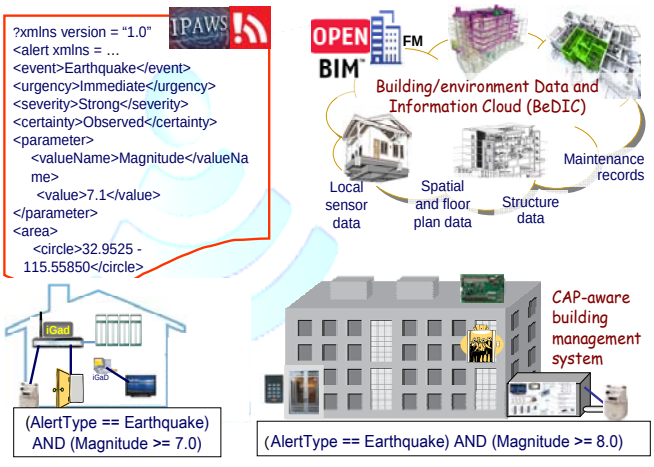


Figure 2. Underlying assuptions

Figure 2 highlights three of the underlying assumptions: First, all messages are compliant to the XML-based CAP standard. They are sent by trustworthy entities (e.g., in US,

responsible authorities via IPAWS-OPEN) and the building management system. So, their contents can be secured and authenticated by the existing XML security mechanism [26].

Second, the decisions of individual iGaDs on whether and how to respond to an alert are based in part on the alert type and severity specified by the alert. In an AERS for indoor spaces, their decisions are also based on data on the building, including its seismic code and maintenance records. For example, suppose that the home and office building in Figure 2 are designed to withstand earthquakes of magnitude 7.0 and 8.0, respectively. Then, CAP-aware door and gas value controllers in the home should respond to the magnitude 7.8 earthquake alert in Figure 1, but the devices of the same types in the office building should ignore the alert. Building data are provided by an information system, called Building and environment Data and Information Cloud (BeDIC) in Figure 2. It contains datasets selected from BIM and facility management system of the building.

Third, the response decision of an iGaDs also depends on how the device(s) is used and data (e.g., sensor data) from local sources. For example, upon receiving a Enhanced Fujita (EF) [27] scale 5 tornado alert, an iGaD controlling a public shelter door should open the door unconditionally. An iGaD controlling the front door of a house may wait until the tornado is about to strike the house, indicated by drastic decrease of outside air pressure, and then opens the door.

From these examples, we can see that iGaDs must be configurable and customizable, not only at installation times but also at maintenance and runtimes. Figure 3 shows an architectural framework for iGaDs for building configurable and customizable iGaDs for diverse purposes from the same set of components [2][3]. Specifically, every iGaDs has a CAP message processor/parser for validating CAP-compliances of the message and extracting from each CAP message the type and severity of the disaster, areas targeted by the message and so on. Every iGaD has a location filter that determines whether the device is located in an affected area and hence is targeted by the alert. An embedded iGaD has a device controller that interfaces with one or more physical devices. Customization of the kinds mentioned above is enabled by using a rule engine to process action activation rules such as the ones shown in Figure 2. The rules are selected and their parameters set at installation and maintenance time of each iGaD.

Some iGaDs are reachable only via the Internet. Examples include CAP-aware elevator, smart gas valve and door controllers. These devices receive alerts relayed by the building (home) management system that is connected to the Internet and serves as an aggregation server. Clearly, iGaDs and people can take protective actions in preparation of an imminent calamity only when they receive warnings about the calamity in time. This means that the end-to-end delay of earthquake warning messages should be a second or less, and delay for tornado and flash flood warnings a minute to a few minutes, and so on. Performance data of Asynchronous Message Delivery Service (AMeDS) [3] [4] for delivering CAP messages asynchronously over the Internet show that end-to-end delay requirements of this order are feasible and AMeDS offers a way to do so.

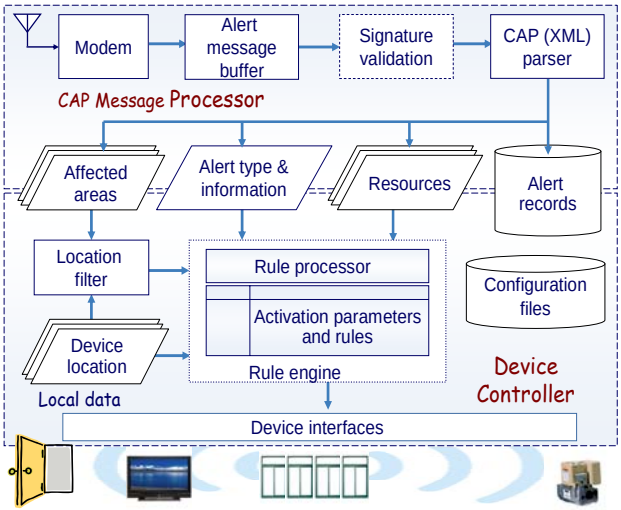


Figure 3. iGaD architecture and key components

IV. AERS SIMULAITON FRAMEWORK

Again, a major thrust of our current work is on safety of AERS, in particular, systems containing a large number of diverse iGaDs and local sensors and serving large complex buildings and facilities, such as transport hubs, major hospitals, sports centers, and shopping malls. A common definition of safety is the absence of *dangerous conditions* that can cause death, injury, damage to property and economical loss [28]. This definition is not appropriate for AERS since such systems work in the presence of dangerous conditions. As an alternative definition of safety, we may say that an AERS is *safe* if its actions never create new dangerous conditions and never increase the probability of occurrence of dangerous conditions known to exist when the system is not in use.

We work with a definition that is more practical from the point of view of validation: We say that a system is *safe as specified* when it always removes the dangerous conditions identified by disaster and emergency response experts and defined in its safety requirement specification. We need to be able assess to what degree a given AERS is safe (i.e., safe as specified) under all likely operating conditions/demands, including occurrences of nearly simultaneous multiple alerts that require conflicting responses; arbitrary sequences of alerts, cancellations, and re-issuances; and false alarms and missed detections of specified rates. The combined actions of a large number of iGaDs may lead to unexpected dangerous conditions, even when all alerts are correct and delivered in time and every device and application works correctly. The problem of making AERS serving large public buildings safe is further complicated by two factors. First, iGaDs may need to collaborate and coordinate their actions for error/failure handling and conflict resolution purposes. The complexity thus introduced may actually make the system less safe. The second complicating factor is the presence of people and crowds, who are also smart entities and may respond to alerts on their own in unsafe ways unless constrained from doing so. The problem is to identify the constraints.

 ISBN: 978-1-61208-429-9

Motivated by the fact that highly available, secure and configurable and maintainable AERS may nevertheless be unsafe, we are developing the simulation framework AERS-SF capable of supporting simulation experiments on diverse AERS for purposes of finding safety flaws and assessing their safety throughout their design, development and deployment. We also want to evaluate via simulation constraints on operations of the system and its components, which when adhered to, can make the system safer.

Figure 4 shows the major components of AERS-SF. The framework will offer libraries of models, tools, and test scenarios generators, together with a simulation environment, using which a user (i.e., a designer or a developer) can construct customized simulator(s) of his/her AERS in building(s) targeted by the system and conduct experiments with design choices (e.g., action activation rules and conflict resolution protocols) of individual iGaDs and alternative Standard Operating Procedures (SOPs) governing alert cancellations and false alarms, for the system as a whole. Specifically, AERS-SF model libraries have (1) agent-based models of active entities in AERS and operating environment, including executable models of iGaDs; (2) behavior models of people as individuals and as members of crowds; (3) BIM-based models of representative buildings and facilities controlled by iGaDs; and (4) conflict resolution and collaboration protocols for iGaDs and representative SOPs. Similar to model libraries of the Agent-Based Disaster Simulation Environment ABDiSE [22], AERS-SF model libraries are extensible: Model elements in the underlying model of each simulation experiment are dynamically loaded during set up and initialization time. The user can add new types of models by providing dynamic linked library functions defining the behavior of new types.

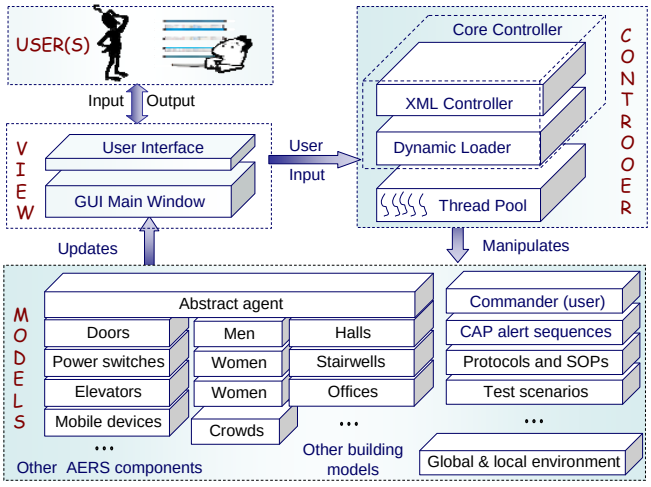


Figure 4. Structure and major components of AERS-SF

To support what-if experiments, the framework will also have extensible libraries of test scenarios. In particular, it will provide traces of disaster and emergency alerts, both actual traces from CAP alert message records that have been released as open data in many countries and synthetic traces that can be used as benchmark input to the system being evaluated. Some of the scenarios detailing the development of emergencies within the targeted building are generated from historical records of common types of disasters and local emergencies. For example, scenario generation scripts can use as input information extracted from historical records on impacts of past typhoons and debris flows on similar buildings. We also plan to link AERS-SF with ABDiSE and through it, to import external disaster simulation programs.

AERS-SF will adopt two other features of ABDiSE. One is to build model elements on common-sense concepts. For example, every simulation experiment has one and only one simulation world, i.e., the geographical area specified by the user for the experiment at set up time. The world may have many regions with specified boundaries. The simulation world has a global environment, and some regions may have local environments that differ from the global environment. Each environment is defined by a set of environment parameters. The behaviors of all agents around any point in space and time within a region depend on the values of local environment parameters at that point in space and time. Thus, we eliminate the need to model sensors explicitly.

Also, similar to ABDiSE, AERS-SF makes tools for building the underlying model for each series of simulation experiments and for controlling simulation runs accessible to the user from the GUI of the framework. Figure 5 uses a marked up screen dump of ABDiSE to illustrate this point. The most prominently displayed tool is the Map Explorer in area B, which displays a 2-D map of a region (e.g., an office area shown here). The tool provides the user with an easy way to specify locations of agents (e.g., two CAP-aware doors). Area A provides access to tools using which the user can select and retrieve model elements from libraries and use them to construct and customize simulation models of the target AERS and its operating environment. When new agent types need to be created, a click of “Create New Agent” button in area A is the first step. Area C displays the list of all model elements that have been selected. Area D lets the user set up and control simulation experiments (e.g., lengths of time steps and the current simulation run). Area E lets the user to specific environment parameters of the region displayed in area B. The user can also visualize via the GUI the development of the scenario within the part displayed in area B during the simulation run.

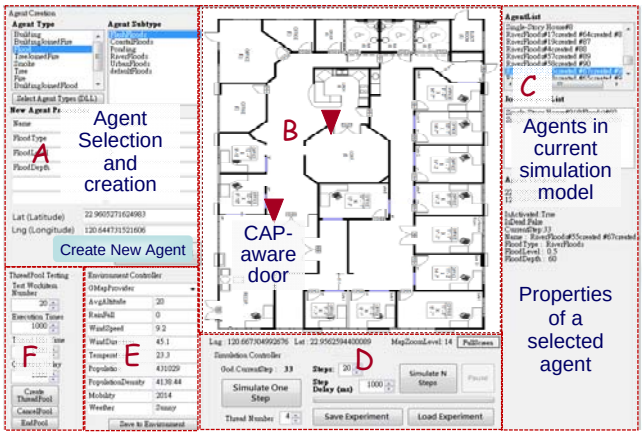


Figure 5. GUI, tools and use scenario

V. CONCLUSION

The previous sections first presented the need for AERS and ways to make them configurable, maintainable and secure. Among all attributes of dependability, safety is the most challenging one for AERS for reasons stated earlier. We are developing the simulation framework AERS-SF designed to support the use of simulation as a tool for assessing the safety of AERS of diverse AERS in diverse operating environments throughout their development and deployment process. Thus far, we have been focusing on its design; especially we want to make sure that the framework will support the underlying models, simulation methods, data capture and analysis methods required to meet its design goals. We have adopted some of the approaches of ABDiSE. Compared with that framework, AERS-SF is far more complex in almost all aspects. Nevertheless, we believe that the software architecture of ABDiSE, as well as some of its software components, can be adopted and enhanced to give the implementation of AERS-SF a head start.

ACKNOWLEDGMENT

This work is supported by the Academia Sinica, Sustainability Science Research Project OpenISDM.

REFERENCES

[1] C. Y. Lin, E. T.-H. Chu, L.-W. Ku, and J. W. S. Liu, "Active Disaster Response System for a Smart Building," Sensors, 14, 2014, pp.17451-17470, doi:10.3390/s140917451.

[2] J. W. S. Liu, E. T. H. Chu and C. S. Shih, "Cyber-Physical Element of Disaster Prepared Smart Environment," IEEE Computer, Vol. 46, No. 2, Feb. 2013, pp. 69 – 75, doi:10.1109/MC.2012.149.

[3] W. P. Laio, Y. Z. Ou, E. T. H. Chu, C. S. Shih, and J. W. S. Liu, "Ubiquitous Smart Devices and Applications for Disaster Preparedness," Proc. of the 2012 Int. Symp. Ubiquitous Intelligence & Computing, Frontiers Workshop, IEEE Press, Sep. 2012, pp. 764 – 770, doi:10.1109/UIC-ATC.2012.98.

[4] Y. Z. Ou, C. M. Huang, C. T. Hu, E. T. H. Chu, C. S. Shih, and J. W. S. Liu, "Responsive Alert Delivery over IP Network," Proc. of IEEE Int. Conf. on Cyber Physical Systems, Networks and Applications, IEEE Press, Aug. 2013, pp. 19 – 25, doi:10.1109/CPSNA.2013.6614241.

[5] Common Alert Protocol Version 1.2, OASIS Standard, http://docs.oasis-open.org/emergency/cap/v1.2/CAP-v1.2-os.html (Retrieved: June 2015).

[6] Integrated Public Alert & Warning System (IPAWS)-OPEN, https://www.fema.gov/integrated-public-alert-warning-system-open-platform-emergency-networks, (Retrieved: June 2015).

[7] Building Information Models/Modeling (BIM) - Wikipedia, http://en.wikipedia.org/wiki/Building_information_modeling, (Retrieved: June 2015).

[8] P. Teicholz (editor). BIM for Facility Managers, Mar 2013, IFMA Foundation.

[9] G. Percivall, "Smart Cities Spatial Information Framework," Jan 2015, Open Geospatial Consortium Document, https://portal.opengeospatial.org/files/?artifact_id=61188, (Retrieved: June 2015).

[10] 2011 Magnitude 5.8 Virginia Earthquake, August 23, 2011, http://en.wikipedia.org/wiki/2011_Virginia_earthquake, (Retrieved: June 2015).

[11] "No dependability, no Internet of Things," Mar. 2004, https://ec.europa.eu/digital-agenda/en/news/no-dependability-no-internet-things, (Retrieved : June 2015)

[12] P. H. Su, C. S. Shih, J. Y-J. Hsu, J.Y.-J., K. J. Lin and Y. C. Wang, "Decentralized fault tolerance mechanism for intelligent IoT/M2M middleware," Proc. of IEEE World Forum on IoT, IEEE Press, Mar. 2014, pp. 45 – 50, doi:10.1109/WF-IoT.2014.6803115.

[13] S, Cherrier, Y. M. Ghamri-Doudane, S. Lohier, and G. Roussel, "Fault-recovery and coherence in Internet of Things choreographies," Proc. of IEEE World Forum on IoT, IEEE press, Mar. 2014, pp. 532 – 537, doi:10.1109/WF-IoT.2014.6803224

[14] N. Maalel, N., E. Natalizio, A. Bouabdallah, P. Roux and M. Kellil, "Reliability for emergency applications in Internet of Things," Proc. of Int. Conf. on Distributed Computing in Sensor Systems (DCOSS), IEEE Press, Mar. 2015, pp. 361 – 366, doi:10.1109/DCOSS.2013.40.

[15] RELYonIT Project, http://www.relyonit.eu/, (Retrieved: June 2015).

[16] RERUM: REliable, Resilient and secUre IoT for sMart city applications, https://ict-rerum.eu/, (Retrieved: Jun 2015).

[17] L. Wu, and G. Kaiser, "FARE: A Framework for Benchmarking Reliability of Cyber-Physical Systems," Proc. of IEEE 2013 Long Island Systems, Applications and Technology Conference, IEEE press, May 2013, pp. 1-6, doi:10.1109/LISAT.2013.6578226.

[18] L. Silva, R. Leandro, D. Macedo , and L. A. Guedes, "A Dependability Evaluation Tool for the Internet of Things," ACM Jr of Comp. and Elect. Eng, Vol. 39, No.7, Oct. 2013, pp. 2005-2018, doi: 10.1016/j.compeleceng.2013.04.021.

[19] NIST Cyber-Physical Systems Testbed Workshop, http://www.nist.gov/cps/cyber-physical-systems-testbed-workshop.cfm, (Retrieved: Jun 2015)

[20] C. M. Macal and M.l J. North, "Introductory Tutorial: Agent-Based Modeling and Simulation", Journal of Simulation, Vol. 4, 2010, pp. 151–162. doi:10.1057/jos.2010.3.

[21] Rob Allan. "Survey of Agent Based Modeling and Simulation Tools," http://www.grids.ac.uk/Complex/ABMS/, (Retrieved: June 2015).

[22] Hsu, T. L. and J. W.S. Liu, "An Agent-Based Disaster Simulation Environment," Academia Sinica Technical Report No. TR -IIS-15-005, March 2015.

[23] Case Study: Sophisticated fire safety systems enable rapid isolation of incidents and evacuation of occupants, http://w3.siemens.com/topics/global/en/sustainable-cities/resilience/pages/sophisticated-fire-safety-systems.aspx, (Retrieved: June 2015)

[24] A. Sagun, C. J. Anumba, and D. Bouchlaghem, "Safety Issues in Building Design to Cope with Extreme Events: Case Study of an Evacuation Process," J. Archit, Eng, Vol. 20, No.3, Sept. 2014, doi: 10.1061/(ASCE)AE.1943-5568.0000147.

[25] B. Wang, H. Li, Y. Rezgui, A. Bradley, and H. N. Ong, "BIM Based Virtual Envionrment for Fire Emergency Evacauation," The Sci. World Jr., 2014, Article ID 589016, 22 pages.

[26] W3C XML Security 2.0, http://www.w3.org/TR/xmlsec-reqs2/, April 2013, (Retrieved: June 2015).

[27] Enhanced Fujita (EF) scale, from Wikipedia, http://en.wikipedia.org/wiki/Enhanced_Fujita_scale, (Retrieved: June 2015).

[28] NASA System Safety Handbook, Vol. 1, NASA/SP-2010-580, Nov. 2011, http://www.w3.org/TR/xmlsec-reqs2/, (Retrieved: June 2015).

 ISBN: 978-1-61208-429-9

Trust-based Service Management of Mobile Devices in Ad Hoc Networks

Yating Wang, Ing-Ray Chen
Virginia Tech
Department of Computer Science
Falls Church, VA, USA
Email: {yatingw, irchen}@vt.edu

Jin-Hee Cho
U.S. Army Research Laboratory
Computational and Information Sciences Directorate
Adelphi, MD, USA
Email: jinhee.cho@us.army.mil

***Abstract*— With the proliferation of fairly powerful mobile devices and ubiquitous wireless technology, traditional mobile ad hoc networks (MANETs) now migrate into a new era of service oriented MANETs wherein a mobile device can provide and receive service from other mobile devices it encounters and interacts with. We discuss our ongoing research efforts in trust management and trust-based algorithm design for service-oriented MANET applications to answer the challenges of MANET environments, including no centralized authority, dynamically changing topology, limited bandwidth and battery power, limited observations, unreliable communication, and the presence of malicious nodes who act to break the system functionality as well as selfish nodes who act to maximize their own gain. We also highlight key ideas and experiences learned, and provide future research directions.**

Keywords-service-oriented mobile ad hoc networks; multi-objective optimization; trust; performance analysis.

I. INTRODUCTION

An autonomous service-oriented mobile ad hoc network (MANET) is populated with service providers (SPs) and service requesters (SRs). A realization of service-oriented MANETs is a web-based peer-to-peer service system with mobile nodes providing web services and users (through their mobile devices) invoking web services. Unlike a web service system in which nodes are connected to the Internet, nodes in service-oriented MANETs are mobile and the communication between peers not within radio range is multi-hop with nodes in the system serving as routers. One can view a service-oriented MANET as an instance of Internet of Things (IoT) systems [7] with a wide range of mobile applications including smart-city, smart tourism, smart car, smart environmental monitoring, and healthcare [1]. It is particularly suitable to military applications where all nodes are mobile with multi-hop communication.

This paper discusses our ongoing research work in trust management and trust-based algorithm design for service-oriented MANETs, key ideas and experiences learned, and future research directions. Our aims are to (1) identify trust dimensions for service-oriented MANET applications; (2) develop an efficient and effective trust protocol for service-oriented MANETs; and (3) develop efficient and effective trust-based algorithms for a set of service-oriented MANET applications. The overarching principle is the design notion of adaptive control, allowing trust computation, aggregation, propagation, formation (out of multiple trust dimensions) and update decisions to be dynamically adjusted to minimize trust bias and maximize application performance. This goal is to be achieved in the presence of malicious mobile devices performing a wide range of attacks, including bad-mouthing, ballot-stuffing, packet dropping, opportunistic service, self-promotion, conflicting behavior, and on-off service attacks for personal gain.

The rest of the paper is organized as follows. Section II discusses related work. Section III discusses the threat model for service-oriented MANETs. Section IV presents our solutions toward trust management of mobile devices in service oriented MANETs. Section V presents our solutions toward trust-based service management for performance optimization of service-oriented MANET applications. Section VI summarizes key research ideas and experiences learned. Finally, Section VII concludes the paper and outlines future research directions.

II. RELATED WORK

Many existing trust models for predicting trust are based on Bayesian inference [3]. Bayesian inference treats trust as a random variable following a probability distribution (e.g., Beta distribution) with its model parameters being updated upon new observations. A shortcoming of Bayesian inference is that trust value does not reveal the uncertainty of trust since it is just a mean. For example, the same trust value can be given to two nodes despite one node was observed for just 2 times, while the other node was observed for 20 times. Belief theory or subjective logic trust models [9] have been proposed to remedy the problem mentioned above, by introducing uncertainty into trust calculation. Fuzzy logic based trust models are also well studied in the literature [12]. Instead of using a binary set, a membership function is defined indicating the degree to which a node is considered trustworthy. Relative to the works cited above based on Bayesian inference, belief theory, or fuzzy logic, we take an entirely different approach. Our root is in statistical analysis. We develop a regression-based trust model to learn the behavior pattern of a SP, taking *context* information into consideration to estimate the reliability trust of a SP that is selected by a SR to execute a service request under a particular environment context.

A significant amount of work has been done in the area of trust-based defenses against attacks in MANETs [13]-[18], [35]-[38]. A common drawback is that dynamically tuning trust parameters may perform poorly when a node

 ISBN: 978-1-61208-429-9

does not have enough self-observation experiences with other nodes in MANET environments and must rely on recommendations. Different from the works cited above, we advocate the use a robust statistical kernel to tolerate false recommendations to effectively achieve resiliency against recommendation attacks. Also unlike existing work, our goal is not to identify "bad" SPs, but to predict whether a SP, whether "good" or "bad," can provide good service, given a set of context variables characterizing the MANET operational environment, including dynamically changing topology, limited bandwidth, battery power, and unreliable communication. In our approach, a SR learns and predicts a SP's service behavior taking context information into consideration, instead of just judging a SP's trustworthiness from self-observations or recommendations received, as having been done in existing works.

III. THREAT MODEL

Just like Internet-based web services, in a service-oriented MANET there are malicious SPs acting for their own gain. The common goal of malicious nodes is to increase their chance of being selected for providing service. Malicious nodes can collude to achieve this common goal. We consider the following malicious attacks in our research:

1. *Bad-mouthing attacks*: a malicious node can ruin the trust of a good node (by providing bad recommendations against it) so as to decrease the chance of that node being selected for service. This is a form of collusion recommendation attacks, i.e., a malicious node can collaborate with other malicious nodes to ruin the trust of a good node.
2. *Ballot-stuffing attacks*: a malicious node can boost the trust of another malicious node (by providing good recommendations) so as to increase the chance of that malicious node being selected as a SP. This is another form of collusion recommendation attacks, i.e., a malicious node can collaborate with other malicious nodes to boost the trust of each other.
3. *Packet-dropping attacks*: when serving as a packet relaying node, a malicious node can delay forwarding or simply drop data packets to ruin the trust of the source node.
4. *Opportunistic service attacks:* a malicious node can provide good service to gain high reputation when it senses its trust status is low, and can provide bad service when it senses its trust status is high.
5. *Self-promotion attacks:* A malicious node can boost its service quality information so as to increase its chance of being selected as a SP.
6. *Conflicting behavior attacks*: a malicious node can selectively provide satisfactory service for some SRs while unsatisfactory for others. Here, we note that a node's best service quality is dictated by the environmental and operational conditions at the time a service request is issued. Therefore, a malicious node can only perform conflicting behavior attacks with a service quality not exceeding its best service quality.
7. *On-off attacks*: instead of always performing its best service, a malicious node can perform bad service. With on-off attacks, a malicious node performs bad service on and off (or randomly) so as to avoid being labeled as a low trust node and risk itself not being selected as a SP, as well as not being able to effectively perform bad-mouthing and ballot-stuffing attacks. One can view on-off attacks as random attacks.

A malicious node may also perform data modification attacks to ruin the reputation of a good node. We assume data/source authentication techniques based on PKI can prevent such attacks. A malicious node may also jam the communication channel or perform denial of service (DoS) attacks to overwhelm a SP. We assume that standard intrusion detection techniques [8] are in place to mitigate such attacks.

IV. TRUST MANAGEMENT

One challenge for implementing trust management in service-oriented MANETs is to reliably estimate the trust levels of SPs in a fully distributed manner, in contrast with an e-commerce system with a centralized authority for trust management. Most existing works take direct evidence for direct trust assessment and propagates its observations to other nodes as recommendations for indirect trust assessment. However, a malicious node may violate this protocol. Further, trust management of mobile devices must take "service context" information into consideration. Such service context information includes the current capability of a SP (e.g., energy status), the service environment (e.g., congested wireless traffic), the identity of the SR (e.g., a friend or a stranger), the payoff obtained (which is application-dependent), and the service cost (e.g., energy consumed). All these factors are called "context" variables based on which the service behavior of a node forms a pattern. The key to effective trust management is therefore to learn the service behavior pattern of a node toward these context variables. The behavior pattern learned can be used to assess the *reliability trust* [3] of a SP when it is selected to service a request in a particular context state characterized by these context variables.

More specifically, within a specific type of service, SR i's observation s_{ij}^t at time t of the service quality received from SP j is either "satisfactory" or "unsatisfactory." If the service quality is satisfactory, then s_{ij}^t =1 and SP j is considered *trustworthy;* otherwise, s_{ij}^t =0 and SP j is considered *untrustworthy*. Let the operational and environmental conditions at time t be characterized by a set of distinct context variables deemed appropriate for an application, denoted by a column vector $\mathbf{x}^t = [x_0^t, \dots, x_m^t]^\top$, where x_i^t represents the ith context variable. Then, *reliability trust* or just *trust* for short is the probability that SP j is capable of providing satisfactory service under the operational and environment conditions at time t described by the context variable set $\mathbf{x}^t$.

Let k ($k \neq i$) be a recommender who had a prior service experience with SP j and is asked by SR i to provide its feedback regarding SP j. The recommendation from node k is in the form of $[\mathbf{x}^t, s_{kj}^t]$ specifying the specific operational and environmental context conditions in $\mathbf{x}^t$ under which the

observation in s_{kj}^t was made. For notational conveniences, let $\mathbf{S_{ij}} = [s_{ij}^{t_0}, \ldots, s_{ij}^{t_n}]^\top$, $i \neq j$, denote the cumulative evidence gathered by SR *i* regarding SP j's service quality over $[t_0, t_n]$ including self-observations and recommendations. Also let $\mathbf{X} = [\mathbf{x}^{t_0}, \ldots, \mathbf{x}^{t_n}]^\top$ denote the corresponding operational and environmental context conditions when the observations are made.

The problem is to learn the service behavior pattern of SP *j* by a latent variable $\boldsymbol{\beta}_j$ between $\boldsymbol{S_{ij}}$ and $\mathbf{X}$, and predict the probability that SP j is trustworthy at time t, given the context environment set at time *n*+1, $\mathbf{x}^{t_{n+1}}$, as input, i.e., $T_{i,j}^{t_{n+1}} = \Pr\{s_{ij}^{t_{n+1}} = 1 | \mathbf{x}^{t_{n+1}}, \boldsymbol{\beta}_j\}$. Essentially $T_{i,j}^{t_{n+1}}$ obtained above is the *reliability trust* of SP *j* at time t_{n+1} from SR *i*'s perspective. The service quality at time *n*+1, $\hat{s}_{ij}^{t_{n+1}}$, can be predicted by setting a trust threshold, depending on the SR's tolerance for the risk.

A common practice is to set the trust threshold as a value greater than 0.5. For example, if the trust threshold is set to be 0.6 by SR *i* then the requested service performed by SP *j* is predicted to be satisfactory when the predicted reliability trust is greater than 0.6.

In [2], we utilized *logit regression* as the behavior pattern learning mechanism to solve the above trust assessment problem, resulting in a trust management protocol which we call LogitTrust.

LogitTrust assesses each SP in terms of its service behavior patterns in response to operational and environmental changes characterized by three context variables: $[x_e^t, x_c^t, x_p^t]$ for energy, capability, and price (or reward). Energy is used to measure the cost of task execution. In a congested environment the probability of wireless channel contention and signal interference will be high, so it will cost more for a SP to execute a task because the SP needs to consume more energy in listening to the channel and repeating packet transmission. The reasons for considering the above context variables in service-oriented MANET environments are: (a) a SP is more likely to provide inferior service when the cost of servicing the task is high (b) a SP is likely to provide inferior service when it is limited in resources and capability; and (c) a profit-aware SP is more likely to provide quality service when the SR offers a higher price.

SR *i* will assess the three context variables $[x_e^t, x_c^t, x_p^t]$ while it sends a service request to SP *j* as follows: x_e^t is estimated by the number of neighbors sharing the channel as more energy is consumed for channel contention and packet retransmission when there are more nodes sharing the channel; x_c^t is estimated by the packet traffic to SP *j* as more traffic to SP *j* hinders its processing capability; x_p^t is SR *i*'s reward to SP *j* upon satisfactory service completion. When SP *j* completes the service, SR *i* will assess if the service is satisfactory (1) or not (0), and store the service outcome together with $[x_e^t, x_c^t, x_p^t]$ context information as one record in the dataset set for learning. It can also pass this experience record to another node as a recommendation. A SR in the system uses its own self-observations and recommendations received to learn the behavior pattern of a SP, and predict the reliability trust of the SP on a service request in a particular context environment.

Relying on its robust learning engine, LogitTrust is highly effective against dishonest recommendations (through bad-mouthing and ballot-stuffing attacks). It significantly outperforms existing trust computation models such as Beta reputation with belief discounting [3] and Adaptive Trust Management [4] in terms of trust accuracy because it takes context information into consideration in service behavior assessment. LogitTrust is also efficient in terms of computational complexity as it utilizes a simple linear model to model the relation between context variables and observations.

With conflicting behavior attacks, a SP can selectively provide satisfactory service for some SRs while providing unsatisfactory service for others. In general, the relation between a SR and a SP determines the SP's service attitude toward the SR. This is naturally solved by LogitTrust since LogitTrust is based on SR-SP pairing. That is, each SR evaluates each SP based on its own self-observations and filtered recommendations. If SP *j* provides bad services to a particular SR, then this evidence will be considered by this SR as it learns SP *j*'s behavior pattern (that is, $\boldsymbol{\beta}_j$) and will not trust SP *j* with its service request.

With on-off attacks, a malicious node will attack only randomly so as to evade detection and avoid being classified as a malicious node. To the system, this malicious node is not 100% of the time providing bad service, but just a percentage of time providing bad service. Therefore, SP *j* performing on-off attacks translates into SP *j* providing bad service only randomly instead of persistently, which is a pattern that can be learned by SR as LogitTrust learns SP *j*'s behavior pattern (that is, $\boldsymbol{\beta}_j$). This in effect allows each SR to cope with a particular SP's on-off attack behavior.

V. TRUST-BASED ALGORITHM DESIGN FOR APPLICATION PERFORMANCE MAXIMIZATION

Service-oriented MANET applications are on the rise thanks to the proliferation of fairly powerful mobile devices and ubiquitous wireless technology. We aim to design and validate trust-based algorithms for application performance maximization for service-oriented MANET applications with the goal of satisfying multiple objectives with conflicting goals to achieve multi-objective optimization (MOO).

Trust-based service composition and binding (with or without MOO) has been studied in the web services domain but only a single-trust, i.e., a single dimension of trust, was considered. This largely ignores the fact that trust is multi-dimensional. Identifying proper trust components and forming the overall trust out of multiple trust components to maximize application performance is of paramount importance. We advocate the use of two key trust dimensions in service request execution, namely, *competence* and *integrity*, as the building blocks of a composite trust metric.

Below we discuss our trust-based service management

algorithm designs for solving two service-oriented MANET applications with MOO.

In [5], we investigated a trust-based dynamic task assignment algorithm for performing dynamic task-to-node service assignments to satisfy multiple objectives with conflicting goals. The results demonstrated that our trust-based solution has low complexity and yet can achieve performance comparable to that of the ideal solution with perfect knowledge of node reliability, and can significantly outperform the non-trust-based solution. We analyzed how MOO is achieved by the ideal, trust-based and non-trust-based solutions, and identified parameter settings under which the trust protocol performance in terms of MOO is optimized for the trust-based solution which can best balance multiple objectives with conflicting goals. The results obtained are useful for dynamic trust management to maximize application performance in terms of MOO in the presence of malicious attacks.

In [6], we investigated a trust-based service composition algorithm designed to satisfy mobile user service requests with multiple objectives including maximizing quality-of-service (QoS) and quality-of-information (QoI) while minimizing the service cost (e.g., pricing) with the *user satisfaction* ultimately measuring success. With a service request in hand, a SR has to first formulate a service composition plan based on the available SPs it encounters and interacts with dynamically, and then determine the best node-to-service assignment for achieving MOO. Dynamic service composition and binding is especially complicated in MANETs because of the space-time complexity of mobile devices. This issue is further compounded by the fact that the information received is often malicious, erroneous, partly trusted, uncertain and incomplete in MANET environments. Our trust-based service composition and binding algorithm based on multi-trust outperforms the non-trust-based counterpart using blacklisting, as well as a single-trust-based algorithm using a traditional beta reputation system.

Our trust-based algorithm has a linear runtime complexity and is able to achieve a solution quality approaching that generated by Integer Linear Programming without sacrificing much solution accuracy. We conducted a comparative performance analysis of single-trust vs. multi-trust protocols for peer-to-peer trust evaluation in service-oriented MANETs. We utilized trust to effectively prevent malicious nodes from disrupting the operation of a service-oriented MANET. We conducted a detailed performance analysis and demonstrated that our trust-based algorithm can effectively penalize malicious nodes performing bad-mouthing, ballot-stuffing packet dropping, self-promotion, or opportunistic service attacks, thus filtering out malicious nodes from service participation, and can ultimately lead to high user satisfaction.

VI. KEY IDEAS AND EXPERIENCES LEARNED

The major difference between a service-oriented MANET and an Internet-based web service system is that the information received in MANET environments is often malicious, erroneous, partly trusted, uncertain and incomplete. In this paper we discussed key research ideas for trust-based service management of mobile devices in service-oriented MANETs wherein every node can be a service provider or a service requester.

The first key idea is to take special characteristics of service-oriented MANET environments into consideration so as to design an efficient and effective trust protocol. We discussed a novel logit regression-based trust model called LogitTrust to dynamically estimate the trust of a mobile device based on how it behaves in response to dynamically changing MANET environments characterized by a set of context variables. LogitTrust outperforms traditional approaches based on Bayesian Inference with belief discounting in terms of trust accuracy and resiliency against attacks, while maintaining a low false positive rate. It is efficient as it adopts a simple liner model for behavior learning with low computational complexity. It is effective since it reflects dynamic MANET characteristics, such as limited bandwidth and battery power, as context variables in the learning model formulation.

The 2nd key idea is to use multi-trust instead of single-trust for trust-based algorithm design, recognizing multi-dimensional trust assessment is critical for decision makings.

The 3rd key idea is that multi-trust-based algorithm design is application specific. One must apply the best trust formation tailored to the application requirements to achieve application performance maximization, especially for those applications with multi-objective optimization goals. We demonstrated that our multi-trust-based algorithm outperforms its non-trust-based and single-trust-based counterparts with multi-objective optimization over a range of service-oriented MANET applications, including node-to-service composition and binding, and node-to-task assignment MANET applications. Furthermore, we demonstrated that our multi-trust-based algorithms for solving these problems are efficient (with linear runtime complexity) and effective without compromising solution optimality, when compared with non-trust-based solutions, and other single-trust-based solutions based on Bayesian inference.

VII. FUTURE RESEARCH DIRECTIONS

There are several future research directions for trust management of mobile devices in service-oriented MANETs:

First, we plan to address the issue of runtime learning and decision making for MANET nodes with limited storage and computation resources. This may involve the use of heuristics for each resource-limited node to store most relevant trust records [10].

Second, we plan to incorporate adaptive control to the trust protocol design. A possible direction is to use a recommendation filtering mechanism to dynamically decide if a recommendation is to be taken or not. Adaptive control may be achieved by adjusting the recommender filtering threshold value based on the hostility level in the environment. When the hostility level is low (i.e., not many "bad" nodes are out there), one can set a low threshold so as to take in recommendations into the dataset, because chances are all recommendations are benign. On the other

 ISBN: 978-1-61208-429-9

hand, when the hostility level is high, one can set a high threshold to filter out false recommendations so as not to contaminate the dataset for effective behavior learning.

Third, although we have reflected MANET environment characteristics such as limited bandwidth and energy power as context variables in our trust model formulation, we have not considered node social behaviors which can also be treated as context information. A context variable such as "friendship" can dictate whether a node will perform good service or bad service toward another node, or if a node will perform ballot-stuffing or bad-mouthing attack toward another node. We plan to further test the resiliency of LogitTrust [2] against more complicated environmental and operational scenarios such as noisy environments, social-based service behaviors, as well as more sophisticated attack behaviors such as opportunistic, collusion and insidious attacks [11].

Lastly, we plan to leverage game theory and artificial intelligence principles [19]-[23], and stochastic Petri net modeling techniques [24]-[34] to capture the dynamics between attacker/defense behaviors [39]-[42] and reason how a service requester can perform counterattacks by adaptive trust-based service management for achieving multi-objective optimization of service quality.

ACKNOWLEDGMENT

This work is supported in part by the U. S. Army Research Laboratory and the U. S. Army Research Office under contract number W911NF-12-1-0445. This research was also partially supported by the Department of Defense (DoD) through the office of the Assistant Secretary of Defense for Research and Engineering (ASD (R&E)). The views and opinions of the author(s) do not reflect those of the DoD or ASD (R&E).

REFERENCES

[1] E. Borgia, "The Internet of Things vision: Key features, applications and open issues," Computer Communications, vol. 54, 2014, pp. 1-31.

[2] Y. Wang, Y. C. Lu, I. R. Chen, J. H. Cho, A. Swami, and C. T. Lu, "LogitTrust: A Logit Regression-based Trust Model for Mobile Ad Hoc Networks," 6th ASE International Conference on Privacy, Security, Risk and Trust, Boston, MA, Dec. 2014, pp. 1-10.

[3] A. Jøsang and R. Ismail, "The Beta Reputation System," 15th Bled Electronic Commerce Conf., 2002, pp. 1-14.

[4] I. R. Chen, J. Guo, and F. Bao, "Trust Management for SOA-based IoT and Its Application to Service Composition," IEEE Transactions on Service Computing, 2015, in press.

[5] Y. Wang, I. R. Chen, and J. H. Cho, "Trust-Based Task Assignment in Autonomous Service-Oriented Ad Hoc Networks," IEEE 12th International Symposium on Autonomous Decentralized Systems, Taichung, Taiwan, March 2015, pp. 71-77.

[6] Y. Wang, I. R. Chen, J. H. Cho, K. S. Chan, and A. Swami, "Trust-based Service Composition and Binding for Tactical Networks with Multiple Objectives," 32th IEEE Military Communications Conference (MILCOM 2013), San Diego, CA, USA, Nov. 2013, pp. 1862-1867.

[7] F. Bao and I. R. Chen, "Dynamic Trust Management for Internet of Things Applications," 2012 International Workshop on Self-aware Internet of Things, San Franscisco, CA, USA, Sept. 2012, pp. 1-6.

[8] R. Mitchell and I. R. Chen, "A Survey of Intrusion Detection in Wireless Network Applications," Computer Communications, vol. 42, April 2014, pp. 1-23.

[9] A. Jøsang, "A Logic for Uncertain Probabilities," International Journal of Uncertainty, Fuzziness and Knowledge-based Systems, vol. 9, no. 3, 2001, pp. 279-311.

[10] I. R. Chen, F. Bao, and J. Guo, "Trust-based Service Management for Social Internet of Things Systems," IEEE Trans. on Dependable and Secure Computing, 2015, in press.

[11] R. Mitchell and I. R. Chen, "Effect of Intrusion Detection and Response on Reliability of Cyber Physical Systems," IEEE Transactions on Reliability, vol. 62, no. 1, 2013, pp. 199-210.

[12] H. Xia, Z. Jia, L. Ju, and Y. Zhu, "Trust Management Model for Mobile Ad Hoc Network Based on Analytic Hierarchy Process and Fuzzy Theory," IET Wireless Sensor Systems, vol. 1, no. 4, 2011, pp. 248-266.

[13] I. R. Chen, J. Guo, F. Bao, and J. H. Cho, "Trust Management in Mobile Ad Hoc Networks for Bias Minimization and Application Performance Maximization," Ad Hoc Networks, vol. 19, August 2014, pp. 59-74.

[14] I. R. Chen, F. Bao, M. Chang, and J. H. Cho, "Dynamic Trust Management for Delay Tolerant Networks and Its Application to Secure Routing," IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 5, 2014, pp. 1200-1210.

[15] P. Michiardi and R. Molva, "Core: A Collaborative Reputation Mechanism to Enforce Node Cooperation in Mobile Ad Hoc Networks," IFIP 6th Joint Working Conference on Communications and Multimedia Security, Portorož, Slovenia, 2002, pp. 107-121.

[16] J. H. Cho, A. Swami, and I. R. Chen, "Modeling and Analysis of Trust Management for Cognitive Mission-Driven Group Communication Systems in Mobile Ad Hoc Networks," in Int'l. Conf. Computational Science and Engineering, 2009, pp. 641-650.

[17] J. H. Cho, A. Swami, and I. R. Chen, "Modeling and analysis of Trust Management with Trust Chain Optimization in Mobile Ad Hoc Networks," Journal of Network and Computer Applications, vol. 35, no. 3, 2012, pp. 1001-1012.

[18] K. Govindan and P. Mohapatra, "Trust Computations and Trust Dynamics in Mobile Adhoc Networks: A Survey," IEEE Comm. Survey and Tutorials, vol. 14, no. 2, 2012, pp. 279-298.

[19] D. Korzhyk, V. Conitzer, and R. Parr, "Solving Stackelberg Games with Uncertain Observability," 10th International Conference on Autonomous Agents and Multiagent Systems, Taipei, Taiwan, May 2010, pp. 1013-1020.

[20] J. Wang and I. R. Chen, "Trust-based Data Fusion Mechanism Design in Cognitive Radio Networks," IEEE Conference on Communications and Network Security (CNS), San Francisco, Oct. 2014, pp. 53-59.

[21] I. R. Chen and F. B. Bastani,, "Effect of Artificial-Intelligence Planning-Procedures on System Reliability," IEEE Trans-actions on Reliability, vol. 40, no. 3, 1991, pp. 364–369.

[22] I. R. Chen, F. B. Bastani, and T. W. Tsao, "On the Reliability of AI Planning Software in Real-time Applications," IEEE Transactions on Knowledge and Data Engineering, vol. 7, no. 1, 1995, pp. 4–13.

[23] F. B. Bastani, I. R. Chen, and T. W. Tsao, "Reliability of Systems with Fuzzy-Failure Criterion," Annual Reliability and Maintainability Symposium, Anaheim, California, USA, 1994, pp. 442–448.

 ISBN: 978-1-61208-429-9

[24] I. R. Chen and D. C. Wang, “Analyzing Dynamic Voting using Petri Nets,” 15th IEEE Symposium on Reliable Distributed Systems, Niagara Falls, Canada, 1996, pp. 44-53.

[25] I. R. Chen and D. C. Wang, “Analysis of Replicated Data with Repair Dependency,” The Computer Journal, vol. 39, no. 9, 1996, pp. 767-779.

[26] R. Mitchell and I. R. Chen, “Behavior Rule Specification-based Intrusion Detection for Safety Critical Medical Cyber Physical Systems,” IEEE Transactions on Dependable and Secure Computing, vol. 12, no. 1, 2015, pp. 16-30.

[27] I. R. Chen, T. M. Chen, and C. Lee, “Performance Evaluation of Forwarding Strategies for Location Management in Mobile Networks,” The Computer Journal, vol. 41, no. 4, 1998, pp. 243–253.

[28] B. Gu and I. R. Chen, “Performance Analysis of Location-Aware Mobile Service Proxies for Reducing Network Cost in Personal Communication Systems,” Mobile Networks and Applications, vol. 10, no. 4, 2005, pp. 453–463.

[29] O. Yilmaz and I. R. Chen, “Utilizing Call Admission Control for Pricing Optimization of Multiple Service Classes in Wireless Cellular Networks,” Computer Communications, vol. 32, 2009, pp. 317-323.

[30] I. R. Chen and T. H. Hsi, “Performance Analysis of Admission Control Algorithms based on Reward Optimization for Real-Time Multimedia Servers,” Performance Evaluation, vol. 33, no. 2, pp. 89-112, 1998.

[31] I. R. Chen, T. M. Chen, and C. Lee, “Agent-based forwarding strategies for reducing location management cost in mobile networks,” Mobile Networks and Applications, vol. 6, no. 2, 2001, pp. 105-115.

[32] S. T. Cheng, C. M. Chen, and I. R. Chen, “Dynamic Quota-based Admission Control with Sub-Rating in Multimedia Servers,” Multimedia Systems, vol. 8, no. 2, 2000, pp. 83-91.

[33] I. R. Chen, O. Yilmaz, and I. L. Yen, “Admission Control Algorithms for Revenue Optimization with QoS Guarantees in Mobile Wireless Networks,” Wireless Personal Communications, vol. 38, no. 3, 2006, pp. 357-376.

[34] Y. Li and I. R. Chen, “Design and Performance Analysis of Mobility Management Schemes Based on Pointer Forwarding for Wireless Mesh Networks,” IEEE Transactions on Mobile Computing, vol. 10, no. 3, 2011, pp. 349-361.

[35] H. Zhu, S. Du, Z. Gao, M. Dong, and Z. Cao, “A Probabilistic Misbehavior Detection Scheme Towards Efficient Trust Establishment in Delay-Tolerant Networks,” IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 1, 2014, pp. 22-32.

[36] L. Xiong and L. Liu, “PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities,” IEEE Trans. on Knowledge and Data Engineering, vol. 16, no. 7, 2004, pp. 843-857.

[37] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, “The EigenTrust Algorithm for Reputation Management in P2P Networks,” 12th International Conference on World Wide Web, Budapest, Hungary, May 2003, pp. 640-651.

[38] Z. Su et al., “ServiceTrust: Trust Management in Service Provision Networks,” IEEE International Conference on Services Computing, Santa Clara, CA, 2013, pp. 272-279.

[39] J. H. Cho, I. R. Chen, and P. G. Feng, "Effect of Intrusion Detection on Reliability of Mission-Oriented Mobile Group Systems in Mobile Ad Hoc Networks," IEEE Transactions on Reliability, vol. 59, no. 1, 2010, pp. 231-241.

[40] H. Al-Hamadi and I. R. Chen, “Adaptive Network Defense Management for Countering Smart Attack and Selective Capture in Wireless Sensor Networks,” IEEE Transactions on Network and Service Management, 2015, in press.

[41] R. Mitchell and I. R. Chen, “Modeling and Analysis of Attacks and Counter Defense Mechanisms for Cyber Physical Systems,” IEEE Transactions on Reliability, 2015, in preess.

[42] R. Mitchell and I. R. Chen, “Behavior Rule Based Intrusion Detection Systems for Safety Critical Smart Grid Applications,” IEEE Transactions on Smart Grid, vol. 4, no. 3, Sept. 2013, pp. 1254 – 1263.

 ISBN: 978-1-61208-429-9