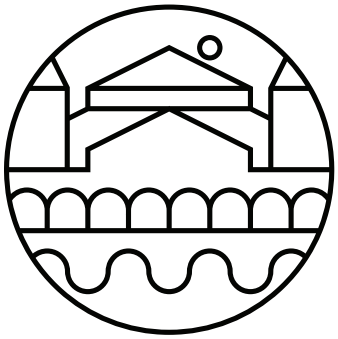
XXXV
SIMPÓSIO BRASILEIRO DE
REDES DE COMPUTADORES
E SISTEMAS DISTRIBUÍDOS

15 a 19 de maio de 2017
Belém - Pará

Anais XVIII WTF 2017



XXXV
SIMPÓSIO BRASILEIRO DE
REDES DE COMPUTADORES
E SISTEMAS DISTRIBUÍDOS

15 a 19 de maio de 2017
Belém - Pará

Anais do XVIII WTF 2017
Workshop de Testes e Tolerância a Falhas

Editora
Sociedade Brasileira de Computação (SBC)

Organização
Cátia Mesquita Brasil Khouri (UESB)
Ronaldo Alves Ferreira (UFMS)
Antônio Jorge Gomes Abelém (UFPA)
Eduardo Coelho Cerqueira (UFPA)

Realização
Sociedade Brasileira de Computação (SBC)
Universidade Federal do Pará (UFPA)
Laboratório Nacional de Redes de Computadores (LARC)

Capa: Catarina Nefertari (PCT-UFPA)
Produção Editorial: Denis Lima do Rosário (UFPA)

Cópias Adicionais:
Sociedade Brasileira de Computação (SBC)
Av. Bento Gonçalves, 9500- Setor 4 - Prédio 43.412 - Sala 219
Bairro Agronomia - CEP 91.509-900 - Porto Alegre - RS
Fone: (51) 3308-6835
E-mail: sbc@sbc.org.br

XVIII Workshop de Testes e Tolerância a Falhas (18: 2017: Belém, Pa).

Anais / XVIII Workshop de Testes e Tolerância a Falhas – WTF; organizado por Antônio Jorge Gomes Abelém, Eduardo Coelho Cerqueira, Ronaldo Alves Ferreira, Cátia Mesquita Brasil Khouri - Porto Alegre: SBC, 2017

111 p. il. 21 cm.
Vários autores
Inclui bibliografias

1. Redes de Computadores. 2. Sistemas Distribuídos. I. Abelém, Antônio Jorge Gomes II. Cerqueira, Eduardo Coelho III. Ferreira, Ronaldo Alves IV. Khouri, Cátia Mesquita Brasil V. Título.

Sociedade Brasileira da Computação

Contato
Av. Bento Gonçalves, 9500
Setor 4 - Prédio 43.412 - Sala 219
Bairro Agronomia
91.509-900 – Porto Alegre RS
CNPJ: 29.532.264/0001-78
http://www.sbrc.org.br

Laboratório Nacional de Redes de Computadores (LARC)

Diretora do Conselho Técnico-Científico
Rossana Maria de C. Andrade (UFC)

Vice-Diretor do Conselho Técnico-Científico
Ronaldo Alves Ferreira (UFMS)

Diretor Executivo
Paulo André da Silva Gonçalves (UFPE)

Vice-Diretor Executivo
Elias P. Duarte Jr. (UFPR)

Membros Institucionais
SESU/MEC, INPE/MCT, UFRGS, UFMG, UFPE, UFCG (ex-UFPB Campus Campina Grande), UFRJ, USP, PUC-Rio, UNICAMP, LNCC, IME, UFSC, UTFPR, UFC, UFF, UFSCar, IFCE (CEFET-CE), UFRN, UFES, UFBA, UNIFACS, UECE, UFPR, UFPA, UFAM, UFABC, PUCPR, UFMS, UnB, PUC-RS, PUCMG, UNIRIO, UFS e UFU.

Contato
Universidade Federal de Pernambuco - UFPE
Centro de Informática - CIn
Av. Jornalista Anibal Fernandes, s/n
Cidade Universitária
50.740-560 - Recife - PE
http://www.larc.org.br

Organização do SBRC 2017

Coordenadores Gerais
Antônio Jorge Gomes Abelém (UFPA)
Eduardo Coelho Cerqueira (UFPA)

Coordenadores do Comitê de Programa
Edmundo Roberto Mauro Madeira (UNICAMP)
Michele Nogueira Lima (UFPR)

Coordenador de Palestras e Tutoriais
Edmundo Souza e Silva (UFRJ)

Coordenador de Painéis e Debates
Luciano Paschoal Gaspary (UFRGS)

Coordenadores de Minicursos
Heitor Soares Ramos (UFAL)
Stênio Flávio de Lacerda Fernandes (UFPE)

Coordenadora de Workshops
Ronaldo Alves Ferreira (UFMS)

Coordenador do Salão de Ferramentas
Fabio Luciano Verdi (UFSCar)

Comitê de Organização Local
Adailton Lima (UFPA)
Alessandra Natasha (CESUPA)
Davis Oliveria (SERPRO)
Denis Rosário (UFPA)
Elisangela Aguiar (SERPRO)
João Santana (UFRA)
Josivaldo Araújo (UFPA)
Marcos Seruffo (UFPA)
Paulo Henrique Bezerra (IFPA)
Rômulo Pinheiro (UNAMA)
Ronedo Ferreira (META)
Thiêgo Nunes (IFPA)
Vagner Nascimento (UNAMA)

Comite Consultivo
Allan Edgard Silva Freitas (IFBA)
Antonio Alfredo Ferreira Loureiro (UFMG)
Christian Esteve Rothenberg (UNICAMP)
Fabíola Gonçalves Pereira Greve (UFBA)
Frank Augusto Siqueira (UFSC)
Jussara Marques de Almeida (UFMG)
Magnos Martinello (UFES)
Antonio Marinho Pilla Barcellos (UFRGS)
Moisés Renato Nunes Ribeiro (UFES)
Rossana Maria de Castro Andrade (UFC)

Mensagem dos Coordenadores Gerais

Sejam bem-vindos ao 35o Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2017) e a acolhedora cidade das mangueiras - Belém / Pará.

Organizar uma edição do SBRC pela segunda vez no Norte do Brasil é um desafio e um privilégio por poder contribuir com a comunidade de Redes de Computadores e Sistemas Distribuídos do Brasil e do exterior. O SBRC se destaca como um importante celeiro para a discussão, troca de conhecimento e apresentação de trabalhos científicos de qualidade.

A programação do SBRC 2017 está diversificada e discute temas relevantes no cenário nacional e internacional. A contribuição da comunidade científica brasileira foi de fundamental importância para manter a qualidade técnica dos trabalhos e fortalecer a ciência, tecnologia e inovação no Brasil.

Após um cuidadoso processo de avaliação, foram selecionados 77 artigos completos organizados em 26 sessões técnicas e 10 ferramentas para apresentação durante o Salão de Ferramentas. Além disso, o evento contou com 3 palestras e 3 tutoriais proferidos por pesquisadores internacionalmente renomados, 3 painéis de discussões e debates, todos sobre temas super atuais, 6 minicursos envolvendo Big Data, sistemas de transportes inteligentes, rádios definidos por software, fiscalização e neutralidade da rede, mecanismos de autenticação e autorização para nuvens computacionais e comunicação por luz visível, bem como 10 workshops.

O prêmio “Destaque da SBRC” e uma série de homenagens foram prestadas para personalidades que contribuíram e contribuem com a área. O apoio incondicional da SBC, do LARC, do Comitê Consultivo da SBRC e da Comissão Especial de Redes de Computadores e Sistemas Distribuídos da SBC foram determinantes para o sucesso do evento. A realização do evento também contou com o importante apoio do Comitê Gestor da Internet no Brasil (CGI.br), do CNPq, da CAPES, do Parque de Ciência e Tecnologia Guamá, da Connecta Networking, da Dantec Telecom, da RNP e do Google. Nosso especial agradecimento à Universidade Federal do Pará (UFPA) e ao Instituto Federal do Pará (IFPA) pelo indispensável suporte à realização do evento.

Nosso agradecimento também para os competentes e incansáveis coordenadores do comitê do programa (Michele Nogueira/UFRP – Edmundo Madeira/UNICAMP), aos coordenadores dos minicursos (Stênio Fernandes/UFPE – Heitor Ramos/UFAL), ao coordenador dos workshops (Ronaldo Ferreira/UFMS), ao coordenador de painéis e debates (Luciano Gaspary/UFRGS), ao coordenador do Salão de Ferramentas (Fabio Verdi/UFSCar) e ao coordenador de palestras e tutoriais (Edmundo Souza e Silva/UFRJ). Destacamos o excelente trabalho do comitê de organização local coordenado por Denis do Rosário.

Por fim, desejamos a todos uma produtiva semana em Belém.

Antônio Abelém e Eduardo Cerqueira

Coordenadores Gerais do SBRC 2017

Mensagem do Coordenador de Workshops

É com grande prazer que os convido a prestigiar os workshops do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC) nos dias 15, 16 e 19 de maio de 2017. Tradicionalmente, os workshops abrem e fecham a semana do SBRC e são responsáveis por atrair uma parcela expressiva de participantes para o Simpósio. Como coordenador de workshops, dividi com os coordenadores gerais do SBRC a nobre tarefa de selecionar os workshops que melhor representam a comunidade e que fortaleçam novas linhas de pesquisa ou mantenham em evidência linhas de pesquisa tradicionais.

Em resposta à chamada aberta de workshops, recebemos dez propostas de alta qualidade, das quais nove foram selecionadas. Além disso, mantivemos a longa colaboração com a RNP por meio da organização do WRNP, que já é uma tradição na segunda e terça-feira da semana do SBRC. Dentre as propostas aceitas, sete são reedições de workshops tradicionais do SBRC que já são considerados parte do circuito nacional de divulgação científica nas várias subáreas de Redes de Computadores e Sistemas Distribuídos, como o WGRS (Workshop de Gerência e Operação de Redes e Serviços), o WTF (Workshop de Testes e Tolerância a Falhas), o WCGA (Workshop em Clouds, Grids e Aplicações), o WP2P+ (Workshop de Redes P2P, Dinâmicas, Sociais e Orientadas a Conteúdo), o WPEIF (Workshop de Pesquisa Experimental da Internet do Futuro), o WoSiDA (Workshop de Sistemas Distribuídos Autonômicos) e o WoCCES (Workshop de Comunicação de Sistemas Embarcados Críticos). Como novidade, teremos dois novos workshops com programação diversificada e grande apelo social, o CoUrb (Workshop de Computação Urbana) e o WTICp/D (Workshop de TIC para Desenvolvimento).

Temos certeza que 2017 será mais um ano de sucesso para os workshops do SBRC pelo importante papel de agregação que eles exercem na comunidade científica de Redes de Computadores e Sistemas Distribuídos no Brasil.

Aproveitamos para agradecer o apoio recebido de diversos membros da comunidade e, em particular, a cada coordenador de workshop, pelo brilhante trabalho. Como coordenador dos workshops, agradeço imensamente o apoio recebido da Organização Geral do SBRC 2017.

Esperamos que vocês aproveitem não somente os workshops, mas também todo o SBRC e as inúmeras atrações de Belém.

Ronaldo Alves Ferreira

Coordenador de Workshops do SBRC 2017

Mensagem da Coordenadora do XVIII WTF 2017

Sejam bem vindos ao XVIII Workshop de Testes e Tolerância a Falhas (WTF 2017)! Através dos anos, o WTF tem proporcionado um ambiente de integração da comunidade acadêmica e da indústria que favorece a discussão, a troca de experiências e ideias que visam a construção de sistemas computacionais confiáveis e altamente disponíveis. Nesse ambiente são apresentados trabalhos teóricos e práticos que versam sobre temas como testes, detecção e tolerância a falhas e intrusões, confiabilidade e robustez.

Promovido pela Comissão Especial de Sistemas Tolerantes a Falhas (CE-TF) da Sociedade Brasileira de Computação (SBC), o XVIII WTF acontece, como tradicionalmente, junto com o XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC). Nesta edição serão apresentados 8 artigos completos, cuidadosamente revisados por 31 especialistas do Comitê de Programa, organizados nas seguintes sessões: Segurança e Confiabilidade em Nuvem; Técnicas de Tolerância a Falhas; e Estratégias de Teste.

Agradeço à CE-TF pelo convite para coordenar o XVIII WTF. Também agradeço aos membros do Comitê de Programa pelo forte apoio e colaboração na divulgação do evento, indicação de novos membros, revisão de artigos e coordenação das sessões técnicas. Em especial, aos professores Eduardo Alchieri, Elias Duarte e Fabíola Greve pelas orientações e sugestões. Por fim, agradeço também a todos os envolvidos na realização do XXXV SBRC, que nos acolhe, nas pessoas dos professroes Antônio Abelém e Eduardo Cerqueira, coordenadores gerais; e Ronaldo A. Ferreira, coordenador de Workshops.

Desejo a todos um agradável dia de boas discussões e interatividade.

Cátia Khouri

Coordenadora do WTF 2017

Comitê de Programa

- Alcides Calsavara (PUC-PR)
- Aldelir Fernando Luiz (IFC)
- Allyson Bessani (Universidade de Lisboa)
- Avelino Zorzo (PUC-RS)
- Cátia Mesquita Brasil Khouri (UESB)
- Daniel Batista (USP)
- Daniel Cordeiro (USP)
- Daniel Fernandes Macedo (UFMG)
- Eduardo Alchieri (UNB)
- Eduardo Bezerra (UFSC)
- Eliane Martins (UNICAMP)
- Elias P. Duarte Jr. (UFPR)
- Fabíola Greve (UFBA)
- Francisco Maia (HASLab-INESC TEC/U. Minho)
- Frank Siqueira (UFSC)
- Irineu Sotoma (UFMS)
- João Marynowski (UFPR)
- João Paulo (INESC TEC/University of Minho)
- Luciana Arantes (UPMC/LIP6)
- Luiz Antonio Rodrigues (UNIOESTE)
- Luiz Eduardo Buzato (UNICAMP)
- Marco Vieira (Universidade de Coimbra)
- Miguel Correia (Instituto Superior Técnico)
- Miguel Matos (INESC TEC/Universidade do Minho)
- Raimundo Barreto (UFAM)
- Raul Ceretta Nunes (UFSM)
- Regina Moraes (UNICAMP)
- Sergio Gorender (UFBA)
- Sérgio Luis Cechin (UFRGS)
- Taisy Weber (UFRGS)
- Udo Fritzke Jr. (PUC-MG-Poços de Caldas)

Sumário

Sessão Técnica 1 – Segurança e Confiabilidade em Nuvem 1

PRIVA: a policy-based anonymization library for cloud and big data platform 1
Andre Ferreira (UNICAMP), Tania Basso (UNICAMP), Hebert Silva (UNICAMP) e Regina Moraes (UNICAMP)

Provendo Privacidade e Tolerância a Falhas em Cloud of Things 13
Luis Alberto B. Pacheco (UnB), Eduardo A. P. Alchieri (UnB) e Priscila Solis (UnB)

Armazenamento Seguro em Nuvem com Detecção de Falhas Bizantinas 27
Clésio Matos (UFBA) e Fabíola Greve (UFBA)

Sessão Técnica 2 – Técnicas de Tolerância a Falhas 41

Replicação de Dados no VCube Baseada em DHT de Salto Único 42
Alexandre B. Neto (UNIOESTE), Luiz A. Rodrigues (UNIOESTE) e Elias P. Duarte Jr. (UFPR)

SEU Mitigation for SRAM FPGAs: A comparison via Probabilistic Model Checking 56
Viny Cesar Pereira (INPE), Valdivino Alexandre de Santiago Júnior (INPE) e Silvio Manea (INPE)

Arquitetura de hardware de uma estação remota de entrada analógica em Con-formidade com a IEC 61508 70
Taisy S. Weber (UFRGS), Sérgio Cechin (UFRGS), João de Moraes (UFRGS) e João C. Netto (UFRGS)

Sessão Técnica 3 – Estratégias de Teste 83

Um injetor de falhas de comunicação para implementações do protocolo EtherCAT 84
Luiz Gustavo A. Gomes (UFRGS), João de Moraes (UFRGS), Taisy S. Weber (UFRGS), Sérgio L. Cechin (UFRGS) e João Netto (UFRGS)

Análise e avaliação de Teste de Intrusão para a estratégia de recomendações Tramonto 98
Daniel Dalalana Bertoglio (PUCRS) e Avelino Francisco Zorzo (PUCRS)

XVIII Workshop de Testes e Tolerância a Falhas

SBRC 2017
Sessão Técnica 1
Segurança e Confiabilidade em Nuvem

PRIVA: a policy-based anonymization library for cloud and big data platform

André Ferreira, Tania Basso, Hebert Silva, Regina Moraes

[1]School of Technology – University of Campinas (UNICAMP)
Limeira – SP – Brazil

{andre.victorf,hebert.oliveiras}@gmail.com, {taniabasso,regina}@ft.unicamp.br

Abstract. *Big Data and Cloud Computing are related technologies and allow users to access data from any device. Preserving individual privacy is one of the major issues in this context, as while handling huge volumes of data it is possible that sensitive or personally identifiable information ends up disclosed. This paper presents the PRIVA, a policy-based anonymization library that aims to help providing higher reliability in cloud computing and big data platforms through privacy protection.*

Resumo. *Big data e computação em nuvem são tecnologias que se complementam e permitem que usuários acessem informações a partir de qualquer dispositivo. Preservar a privacidade dos dados neste contexto é um grande desafio pois, ao manipular uma grande quantidade de informação, é possível que dados sensíveis ou de identificação pessoal sejam divulgados sem autorização. Este artigo apresenta a PRIVA, uma biblioteca de anonimização baseada em políticas que tem como objetivo auxiliar na proteção de privacidade, provendo maior confiabilidade para plataformas de big data e computação em nuvem.*

1. Introduction

Big Data and Cloud Computing are technologies that can be combined to generate benefits and advantages for organizations. Cloud Computing enables computing resources to be provided as IT services in a pay-as-you-go fashion with high efficiency and effectiveness. Cloud-based platforms are increasingly utilized as potential hosts for Big Data, allowing the integration and analysis of large volumes of data with heterogeneous formats from different sources. Big Data analytics support the derivation of properties and correlations among data and are considered by companies a key asset to make business decisions.

Despite the numerous benefits and advantages provided by these both technologies, the analyzed data through big data analytics often include personal and sensitive information and this implies threats to privacy. So, organizations are very cautious when adopting these technologies, as there is a concern about data privacy. Usually, the reliability of these platforms are related to privacy protection, i.e., the more a platform protects the privacy of its users, clients, customers and business partners, the more credibility it gets.

One of the possible solutions to address this issue is the use of data anonymization strategies. Data Anonymization, also known as de-identification, consists of techniques that can be applied to prevent the recovery of individual information. It is mainly used to reduce the leakage of information about particular individuals while data are shared and

disclosed to public. The Anonymization process is carried out to change the data before its being disclosed. Anonymization policies, similar to privacy policies (which is a legal document that discloses the conditions under which a party can gather, use, disclose, and manage personally identifiable information), specifies how the anonymization must be performed, minimizing the risk of data re-identification. These policies may be based on privacy principles and laws, as well as data source owners specifications.

There are some anonymization tools available as, for example, ARX [Prasser and Kohlmayer 2015], SDCMicro [Templ et al. 2015], μ-ARGUS [Hundepool et al. 2005]. However, these tools perform the data anonymization based only in the knowledge of their users, requiring them to be privacy specialists and having knowledge about different data from different contexts (e.g., medical data, public transportation data, organizational and financial data, etc.). The library we present in this paper, PRIVA, is based on anonymization polices, i.e., it performs the data anonymization enforcing the rules specified in the policies. This allows improving the privacy laws compliance and data source owners privacy requirements compliance throughout the anonymization process. In addition, the fact that it is policy-based means that the proposed tool does not require its users to have such advanced knowledge in privacy, relaxing them from this task.

This paper is organized as follow: Section II presents a background and related work regarding the existent anonymization tools. Section III presents our policy-based anonymization tool. Section IV describes a case study where data from urban mobility context involving bus transportation is anonymized. Finally, Section V presents the conclusions.

2. Related Work

There are some anonymization tools available. SDCMicro [Templ et al. 2015] is a free, R-based open-source package for the generation of protected microdata for researchers and public use. This package can be used for the generation of anonymized confidential micro-data sets, i.e. for the creation of public and scientific-use files. It includes all popular disclosure risk and perturbation methods (such as global recoding, local suppression, post-randomization, micro-aggregation, adding correlated noise, shuffling and others). It also includes some risk estimation methods. The associated package sdcMicroGUI includes a graphical user interface for various methods in the sdcMicro package.

In the same software product line, SDCTAble [Meindl 2011] is a free and open source R-package to protect tabular data. It provides methods to generate instances of multidimensional, hierarchical table structures, identify primary sensitive table cells within such objects and protect primary sensitive table cells by solving the secondary cell suppression problem. This problem consists of determining which additional cells should be suppressed so that a data intruder, despite knowing the additive relationships of the tables, will not be able to estimate the sensitive cells too precisely [Massell 2001].

Similarly to SDC-family, The Argus software has the μ-ARGUS [Hundepool et al. 2005], which is a software package for the disclosure control of microdata and the τ-ARGUS [Hundepool et al. 2004] for tabular data. The package has been developed using Visual C++ language and runs under Windows versions from Windows 2000 and later. μ-ARGUS implements anonymization techniques as like

global recoding (grouping of categories), local suppression, PostRAndomisation Method (PRAM), adding noise and microaggretation. It implements a methodology for individual risk estimation based on the sampling weight. τ-ARGUS also deals with the secondary cell suppression problem [Hundepool 2004].

Other important anonymization tool for structured data is the ARX [Prasser and Kohlmayer 2015]. It supports methods for statistical disclosure control by providing: anonymization techniques such as generalization, suppression and microaggregation; privacy models such as κ-anonymity, ℓ-diversity, τ-closeness and ℓ-presence; models for analyzing reidentification risks; methods for evaluation of data utility. This tool allows anonymizing datasets with several millions of records and offers a comprehensive graphical user interface with wizards and visualizations that guide users through different aspects of the anonymization process.

Besides the previous presented anonymization tools (SDCMicro, SDCTable, μ-ARGUS, τ-ARGUS, ARX) implement lots of features, they are not policy-based and depend only of their users experience, requiring them to have a high privacy knowledge, including about privacy principles and laws. Furthermore, these tools, except ARX, are not stand-alone and work based on specifics data analytics platforms (e.g., R [Ripley 2001]). The advantage of the library proposed in this paper, that we called PRIVA, is to be based on anonymization polices. It is only needed to inform the policy and the library automatically performs the data anonymization, enforcing the rules specified in it. Furthermore, PRIVA can work stand-alone or be included in different data analytics platforms.

3. PRIVA

Anonymization, roughly speaking, is the act of removing personal identifiers from data, for example, by converting personally identifiable information into aggregated data. Anonymized data can no longer be associated with an individual [Higher Education Information Security Council (HEISC) 2015]. *Anonymization techniques* have been used to provide a balance between the beneficial use of data and the individual privacy. *Anonymization policies* can help in the anonymization process. There are regulations and guidelines to standardize the use of anonymization techniques and algorithms and the implementation of such policies are considered an important progress in organizations that want to protect their customer personal data.

PRIVA is a library developed to perform data anonymization in order to protect sensible information that is processed by data analytics algorithms and, consequently, provide higher reliability in cloud computing and big data platforms. The library is based on anonymization policies and implements the most common anonymization techniques (e.g. generalization, suppression, masking, encryption). It was developed as part of the approach presented in the work of Basso et al. [Basso et al. 2017] and the idea is that it can be integrated to that approach to perform anonymizaton in ETL process and in the two anonymization proposed phases (called *Anonymization1* for the first phase and *Anonymization2* for the second phase). PRIVA is a free and open source tool, developed in Java language, and suitable to be integrated in cloud and big data analytics platforms. Fig. 1 overviews PRIVA.

The *Privacy Library* is a Java library that has as input the data set to be anonymized and the anonymization policy that should be used to guide the anonymization

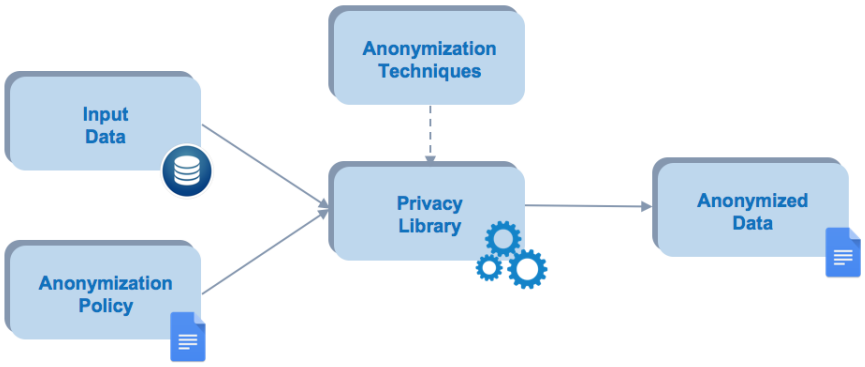


Figure 1. Library PRIVA overview

process. It applies the implemented anonymization techniques according to the policy and provides, as output, the anonymized data set.

The *Input Data* is the data from database tables and, for PRIVA, they must be loaded from files (JSON files). The fields are identified and must be anonymized according to the *Anonymization Policy*.

The *Anonymization Policy* is the guideline to the anonymization process. Basically, this policy must specify the fields related to personally identifiable information and the anonymization techniques that must be applied to each of these fields. These policies may be defined by privacy specialists, based on privacy principles and laws, as well as data source owners specifications. After defined, they can be standardized and reused by users with not so advanced privacy knowledge, facilitating the anonymization process.

The *Anonymization Techniques* refers to anonymization techniques that can be applied on data in order to protect the privacy of individual. Some of the existing and most used techniques are [Basso et al. 2016]:

- *Generalization*: attribute values are generalized to a range in order to reduce the granularity of representation. That is, the date of birth may be generalized to a range such as year of birth, so as to reduce the risk of identification;
- *Suppression*: the key attributes or the quasi-identifiers are removed completely to form the anonymized table, providing only summaries of the table data instead of individual data;
- *Encryption*: it uses cryptographic schemes to replace key-attributes, quasi-identifiers and sensitive attributes for encrypted data;
- *Perturbation (Masking)*: it consists of the replacement of the actual data values for dummy data. The idea is to randomly change the data to mask sensitive information. There are some masking techniques: (i) replacement (random replacement for similar content, but with no relation to the real data); (ii) shuffling (random replacement for data that is derived from the table column); (iii) blurring (applied to numerical data and dates. It changes the value of the data for some percentage of their random real value); (iv) reduction/nulling (replaces sensitive data with null values).

Still in Fig. 1, the *Anonymized Data* represents the resulting data set after

anonymization process. Similar to input, this output may be through JSON files.

Currently, the first release of PRIVA implements the generalization, suppression, masking (replacement) and encryption techniques. The team is working on a second release, where PRIVA will implement more techniques including the other variations of masking.

As a proof of concept, a case study was developed to show the anonymization performed by PRIVA.

4. Case Study

As a case study we performed data anonymization for the EUBra-BIGSEA project [EUBra-BIGSEA 2017]. This project address a cloud and big data platform that handles real data from transportation systems, more specifically from the Curitiba city, in Brazil. Three Big Data sources are used for the dynamic aspects of the system: the GPS location of all buses and user cards in the city (*dynamic spatial data*), data output by fine grained models of weather (*environmental data*), and historical and real-time data posted on social media associated with the city and its locations (*social network data*). This heterogeneous and large volume of data is integrated and continuously processed to detect patterns, trends and outliers in the behavior of the transportation system. The idea is to investigate speed, vehicle flux, traffic disruptions, main origin-destination routes for citizens (according to each day, time, and region), and sentiments and topics historically or recently associated with places, stress caused by traffic, landmarks, weather conditions and the effects of all of these on the perception one has from a trip in the city [EUBra-BIGSEA 2017].

This use case primarily targets two groups of end-users: citizens and urban planners. Citizen can query for the state of the route options available for a given trip. The system provides multiple route options that maximize different criteria in addition to travel time, such as likely stress, pleasantness, interestingness and liveliness of the routes (according to parameters such as day of the week, hour and location). Urban planners can obtain a descriptive view on the state of the mobility in the city as a whole, identifying its status, trends and the impacts on relevant events.

In this scenario, our goal is to provide privacy to the bus passengers. The *dynamic spatial data* is composed by (i) *user cards* data, which represents information comprising trip data per user card: the vehicle id, line code, the user card number, and the date of the trip; (ii) *vehicles and respective bus lines* data, which represents the daily itinerary, having as data the vehicle id, line code, the date time, and the latitude and longitude. With these information it is possible to perform statistical analysis in the database and track a specific user from the user card identification (e.g., it is possible to find out which bus lines were used by a specific passenger and in which localities he/she was);(iii) *personal data*, which represents the information about passengers that have an user card, as name, address, birth data, telephone, etc. When a passenger acquires his/her user card they must provide their personal information. t is important to mention that the real passenger personal information is not available in the platform and, for this case study, we generated a fake database to replace this information. Details are given in the next subsection.

4.1. The input data

Part of the input data to be anonymized in this case study are real data from public transportation from Curitiba city (Brazil). These real data refers to one working day of use of the transport system of Curitiba, in the year 2015. These data were provided for case studies in the context of the EUBra-BIGSEA project. However, the real personal information of the users, that must be associated to the bus card, were not provided, precisely for the preservation of privacy. So, to have these information, we created fictitious records.

The fictitious data was created using the *Fakenamegenerator* tool [Works 2011]. We want the fictitious user profile to be as close as possible to the actual profile. So, the fictitious data were generated respecting the proportionality of the population of Curitiba, according to the last IBGE (*Instituto Brasileiro de Geografia e Estatística* - Brazilian Institute of Geography and Statistics) census [Censo 2010], exemplified in the Age Pyramid of Figure 2.

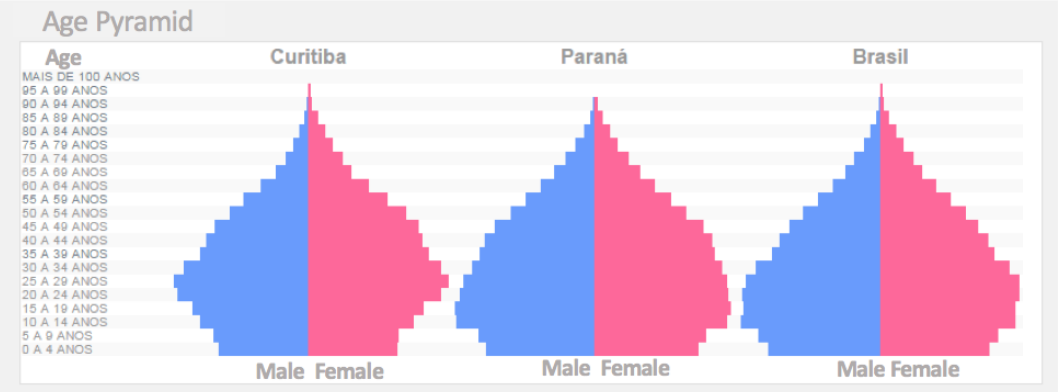


Figure 2. Curitiba's distribution of population by sex (male/female), according to age groups [Censo 2010]

In Figure 2 we can observe that the population of Curitiba city has similar proportions than Paraná State and the general population in Brazil, except for a difference in the 10 to 14 years old and 15 to 19 years old. To be as realistic as possible, we adopted the data from Curitiba.

The data correspond to one working day of use of the transport system has more than 480 thousand transactions of bus user cards. Each card is used, in average, twice a day. So, it is necessary to create more than 245 thousand user registers. We generate a table with the following fields: *Number, Gender, NameSet, Title, GivenName, MiddleInitial, Surname, StreetAddress, City, State , StateFull, ZipCode, Country, CountryFull, EmailAddress, BrowserUserAgent, TelephoneNumber, TelephoneCountryCode, Birthday, TropicalZodiac, CCType, CVV2, CCExpires, NationalID, Color, Occupation, Company, Vehicle, Domain, BloodType, Pounds, Kilograms, FeetInches, Centimeters, GUID*. The records were generated according to the data proportion from Curitiba. Then, a relationship between the tables *BusCardData*(which has data about the card, such as bus line code, bus line name, bus code, user card number, date of use) and *UserData* (which has the personal data with the fields described above) was created in order to associate each user card with a user.

4.2. The anonymization policy

We adopted the policy created for EUBra-BIGSEA [Matsunaga et al. 2017] because it is based on the existing regulations and guidelines for data anonymization found in the literature (European Data Protection Directive, GDPR (General Data Protection Regulation), PIPEDA (The Personal Information Protection and Electronic Documents Act), HIPAA (Health Insurance Portability and Accountability Act), PCI-DSS (Payment Card Industry Data Security Standard)) and built on their strength.

This policy specifies the fields related to Personally Identifiable Information and the classification of these fields into three categories, in light of the disclosure risks: (i) key attributes (attributes that uniquely identify individuals, e.g., ID, name, social security number); (ii) quasi-identifiers (attributes that can be combined with external information to expose some individuals, or to reduce uncertainty about their identities, e.g., birth date, ZIP code, position, job, blood type); (iii) sensitive attributes (attributes that contain sensitive information about individuals, e.g., salary, medical exams, credit card releases). It is a generic policy and does not include all data type that can be stored and managed by a data analytics platform, but it includes the majority of the personally identifiable information (PII) that can be held. Moreover, it can be easily extended to include other data types.

We identified, among the fields of the fictitious database, the fields that must be anonymized according to the policy. Table 1 shows the fields and respective techniques that must be applied for this case study.

Table 1. Anonymization Policy for cloud and big data-based project EUBra-BIGSEA (adapted from [Matsunaga et al. 2017])

	Field Table	Data Type	Technique	Base policy
1	Name	Key Att.	Supression	Safe Harbor method- HIPAA
2	Gender	-	Keep the data	No HIPAA guideline
3	Birth Date	Quasi-ident.	Generalization	Safe Harbor method - HIPAA
4	ID Document	Key Att.	Supression	Safe Harbor method - HIPAA
5	Address	Quasi-ident.	Supression	Safe Harbor method - HIPAA
6	City	Quasi-ident.	Supression	Safe Harbor method - HIPAA
7	State	-	Keep the data	Safe Harbor method - HIPAA
8	Country	-	Keep the data	Safe Harbor method - HIPAA
9	Postal Code	Quasi-ident.	Generalization	Safe Harbor method - HIPAA
10	Telephone Number	Quasi-ident.	Supression	Safe Harbor method - HIPAA
11	Vehicle	Quasi-ident.	Supression	Safe Harbor method - HIPAA
12	E-mail	Key Att.	Supression	Safe Harbor method - HIPAA
13	Color	-	Keep the data	No HIPAA guideline
14	CCType	-	Keep the data	Requirement 3 from PCI-DSS
15	Expiration Data	-	Keep the data	Requirement 3 from PCI-DSS
16	CVV	Sensitive Att.	Supression	Requirement 3 from PCI-DSS
17	Occupation	Quasi-ident.	Keep the data	Safe Harbor method - HIPAA
18	Company Name	Quasi-ident.	Supression	Safe Harbor method - HIPAA
19	User Card Id	Key attribute	Encryption - Hash Function	-

Besides the protection of personally identifiable information, the authors in [Matsunaga et al. 2017] present an extension of the policy to address the key attribute *user card identification*, which must be anonymized using *encryption* technique. We will also use this extension. It is represented in the latest line of Table 1, where *User Card Id*

is the number of the bus card.

Still in Table 1, the *CVV* is Card Verification Value code (CVV2 for Visa, CVC2 for Master Card and CID for AMEX), i.e., the three or four digit number located either on the front or back of a credit or debit card. *CCtype* is the Credit card type. We will use the field *Name* as the whole name, not dividing it into *GivenName, MiddleInitial, Surname*

4.3. The anonymization process

The library we propose will anonymize the input data according to the policy. It receives the both files (input data and anonymization policy) in a JSON format and generates a new JSON file with the anonymized data. Figure 3 shows the anonymization policy enforcement through PRIVA library.

Figure 3. Enforcement of anonymization policy in real bus data from Curitiba's city

In Figure 3, the *Anonymization Policy* guides the anonymization process. It describes, for each field (*FIELD_NAME*), the anonymizaton technique to be applied (*TYPE*, *DETAIL*). For exemplification, we highlighted in Figure 3 the last three fields that requires different techniques: *Company*, which requires the suppression technique (*SUP*), where the company name must be replaced by the character *; *BirthYear*, which requires a generalization (*GEN*) technique where age ranges were defined (who was born from 1900 to 1956 is classified as old; from 1957 to 1999 is adult; from 2000 to 2005 is teenager;

from 2006 to 2017 is child); *BusIdCard*, which requires the *Encryption* technique ((*ENC*), implemented as a Hash function. The other fields requires the suppression technique.

The*Database (Json) file* is the data set to be anonymized, i.e., the real bus data from Curitiba's city and the respective fictitious personal data. In this file, *busIdCard* is the field that contains the bus card identification. The other fields are the ones previously described in Section 4.1. For sake of organization, we present here just a sample of the records and highlighted the same as specified in the policy, just for comparison.

The *privacy library* applies the existing anonymization techniques and algorithms according to the policy and provides, as output, the *Anonymized Database (Json) file*, with the anonymized data set. It is possible to observe that the highlighted fields (as well as the other ones specified in the policy) are anonymized, increasing the privacy protection of these sensible information.

5. Conclusions and Future Work

In this paper we presented PRIVA, a policy-based anonymization library that helps providing higher reliability in cloud computing and big data platforms through privacy protection. Although PRIVA does not implement several resources as other available anonymization tools (e.g., data disclosure risk estimation and evaluation of data utility), the fact of being able to enforce anonymization policies makes this library a first step on (i) allowing the anonymization process be more privacy regulation compliant; (ii) standardizing and reusing anonymization policies; (iii) not requiring users to have so advanced privacy knowledge. Furthermore, PRIVA is generic, open source and can be integrated in different data analytics platforms.

The case study showed an example, as a proof of concept, of standalone use of PRIVA, enforcing a predefined anonymization policy on data from public transportation from Curitiba city. It helped us to identify errors and limitations in the library, which have been solved in order to improve it.

As future work we intend to implement more functionalities in this tool as, for example, the anonymization models (κ-anonymity, ℓ-diversity, τ-closeness). This would help minimizing the risk of data re-identification. Also, we intend to integrate the PRIVA in a data analytics platform. Once anonymization in these platforms can be performed not only on the ETL process, but also at runtime, performance evaluation shall be done and, if necessary, actions to improve the performance would be taken.

Acknowledgment

This work has been partially supported by the project **EUBra-BIGSEA** (www.eubra-bigsea.eu), funded by the Brazilian Ministry of Science, Technology and Innovation (Project 23614 - MCTI/RNP 3rd Coordinated Call) and by the European Commission under the Cooperation Programme, Horizon 2020 grant agreement no `690116`.

Also, it is supported by the project **DEVASSES** (www.devasses.eu), funded by the European Union's FP7 for research, technological development and demonstration under grant agreement no `PIRSES-GA-2013-612569`.

References

[Basso et al. 2016] Basso, T., Matsunaga, R., Moraes, R., and Antunes, N. (2016). Challenges on anonymity, privacy, and big data. In *Dependable Computing (LADC), 2016 Seventh Latin-American Symposium on*, pages 164–171. IEEE.

[Basso et al. 2017] Basso, T., Moraes, R., Antunes, N., Vieira, M., Santos, W., and Meira Jr, W. (2017). Privaaas: privacy approach for a distributed cloud-based data analytics platforms. In *International Workshop On Assured Cloud Computing And QoS Aware Big Data*, pages 1–9. IEEE.

[Censo 2010] Censo, I. (2010). Disponível em:¡ http://www. censo2010. ibge. gov. br/¿. *Acesso em*, 23.

[EUBra-BIGSEA 2017] EUBra-BIGSEA (2017). Eubra-bigsea. europe - brazil collaboration of big data scientific research through cloud-centric applications. `http://www.eubra-bigsea.eu/`.

[Higher Education Information Security Council (HEISC) 2015] Higher Education Information Security Council (HEISC) (2015). Guidelines for data de-identification or anonymization. `https://spaces.internet2.edu/display/2014infosecurityguide/Guidelines+for+Data+De-Identification+or+Anonymization`.

[Hundepool 2004] Hundepool, A. (2004). The argus-software. *Monographs of official statistics*, page 347.

[Hundepool et al. 2004] Hundepool, A., van de Wetering, A., Ramaswamy, R., de Wolf, P.-P., Giessing, S., Fischetti, M., Salazar, J.-J., Castro, J., and Lowthian, P. (2004). Userâs manual.

[Hundepool et al. 2005] Hundepool, A., Van de Wetering, A., Ramaswamy, R., Franconi, L., Capobianchi, A., DeWolf, P., Domingo-Ferrer, J., Torra, V., Brand, R., and Giessing, S. (2005). μ-argus version 4.0 software and userâs manual. *Statistics Netherlands, Voorburg NL*.

[Massell 2001] Massell, P. B. (2001). Using linear programming for cell suppression in statistical tables: Theory and practice. In *Proceedings of the Annual Meeting of the American Statistical Association*.

[Matsunaga et al. 2017] Matsunaga, R., Basso, T., Ricarte, I., and Moraes, R. (2017). Towards an ontology-based definition of data anonymization policy for cloud computing and big data. In *Manuscript submitted for publication*.

[Meindl 2011] Meindl, B. (2011). A computational framework to protect tabular data-r-package sdctable.

[Prasser and Kohlmayer 2015] Prasser, F. and Kohlmayer, F. (2015). Putting statistical disclosure control into practice: The arx data anonymization tool. In *Medical Data Privacy Handbook*, pages 111–148. Springer.

[Ripley 2001] Ripley, B. D. (2001). The r project in statistical computing. *MSOR Connections. The newsletter of the LTSN Maths, Stats & OR Network*, 1(1):23–25.

[Templ et al. 2015] Templ, M., Kowarik, A., and Meindl, B. (2015). Statistical disclosure control for micro-data using the r package sdcmicro. *Journal of Statistical Software*, 67(1):1–36.

[Works 2011] Works, C. (2011). Fakenamegenerator. `www.fakenamegenerator.com/`.

Provendo Privacidade e Tolerância a Falhas em Cloud of Things

Luis Alberto B. Pacheco[1], **Eduardo A. P. Alchieri**[1], **Priscila Solis**[1]

[1]Departamento de Ciência da Computação
Universidade de Brasília (UnB)
Brasília – DF – Brasil

luisbelem@aluno.unb.br,{alchieri,pris}@unb.br

Abstract. *Through the Internet of Things (IoT) a large number of devices are connected to the internet, resulting in a huge amount of produced data. Cloud computing is currently adopted to store, process and access control to these data, this integration is called Cloud of Things - CoT. This approach is useful in personal networks, like residential automation and health care, since it facilitates the access to the information. Although this integration brings benefits to the users, it introduces a federation problem, the information leaves the user control and is housed at the cloud providers. In order for these technologies to be adopted, the users privacy must be ensured. This paper proposes a fault-tolerant architecture to privacy in Cloud of Things, which allows the users to control the data generated by the devices of their IoT networks.*

Resumo. *Através da Internet das Coisas (IoT), uma imensa quantidade de dispositivos são conectados à internet, gerando uma grande quantidade de dados. Atualmente propõe-se a utilização de computação em nuvem para o armazenamento, processamento, apresentação e controle de acesso a esses dados, a essa integração chama-se Cloud of Things - CoT. Essa abordagem é especialmente útil para redes domésticas e pessoais, tais como automação residencial e assistência médica, pois facilita o acesso a informação pelos indivíduos. Apesar de trazer benefícios aos usuários, a integração desses dois conceitos tecnológicos introduz um problema de acesso a informação, que sai da esfera de controle do usuário ao ser enviado para a nuvem. Para que haja uma adoção em massa dessa tecnologia é importante que a privacidade dos dados dos usuários seja mantida. Este trabalho propõe uma arquitetura tolerante a falhas e que provê privacidade em Cloud of Things, permitindo ao usuário o controle do acesso aos dados gerados pelos dispositivos de suas redes IoT.*

1. Introdução

Os avanços nas tecnologias de miniaturização de componentes eletrônicos e nas tecnologias de comunicação sem fio possibilitaram o advento da Internet das Coisas (IoT), onde os mais diversos itens do nosso cotidiano terão acesso à internet, trazendo uma enorme gama de benefícios à população [Chui et al. 2010]. São previstas bilhões de "coisas" conectando-se a internet para prover os mais diferentes tipos de informações aos usuários [Sundmaeker et al. 2010].

Os dispositivos de IoT podem estar presentes nos mais variados meios, mas comumente uma rede IoT é implementada através de sensores. Redes de sensores sem fio

(RSSF) são compostas por dispositivos que possuem uma unidade de processamento, um sensor que proporciona a interação com o mundo físico e uma antena para comunicação sem fio [Akyildiz and Vuran 2010]. A IoT será formada por milhares de RSSFs. Os dispositivos das RSSFs constantemente possuem tamanho e fonte de energia limitados, dessa forma a pilha de rede precisa possuir um baixo custo de processamento. Muitos protocolos em todas as camadas da pilha foram propostos com tal finalidade. Nas camadas Física e de Enlace o padrão IEEE 802.15.4 [Group 2006] foi lançado em 2006 e possui diversas atualizações, sendo atualmente utilizado por grande parte das redes de sensores. Já protocolos das demais camadas possuem uma grande fragmentação, com diferentes propostas para diferentes aplicações, gerando a necessidade da realização de uma tradução quando a comunicação deve ocorrer com a Internet, que é efetuada através de um *gateway*.

A comunicação direta da rede de sensores com a Internet, sem a necessidade de um *gateway*, traz grande benefício para a implementação da IoT, pois facilita o acesso direto à informação advinda do sensor pelo usuário [Granjal et al. 2015]. Existem vários trabalhos em andamento com o objetivo de criar adaptações de protocolos já utilizados na Internet para serem utilizados em redes de sensores. Na camada de rede, o 6LoWPAN [Kushalnagar et al. 2007] efetua compressão e encapsulamento de cabeçalhos IPv6 [Deering 1998] para que caibam em quadros do protocolo IEEE 802.15.4. O IPv6 possui grande capacidade de endereçamento, o que o faz um forte candidato à utilização na Internet das Coisas, que possuirá bilhões de dispositivos conectados. Na camada de transporte, é indicada a utilização do protocolo UDP [Postel 1980], que apesar de não oferecer funcionalidades de entrega confiável, ordenada e com checagem de erros como o TCP [Postel 1981], apresenta um *overhead* muito menor. Apesar dos sensores poderem se comunicar diretamente com outros dispositivos da internet, um *gateway* ainda pode ser utilizado para efetuar operações muito custosas para os pequenos dispositivos.

As limitações tecnológicas da IoT (armazenamento, processamento, comunicação) podem ser mitigadas com a utilização de Computação em Nuvem. Estes modelos são complementares uma vez que a IoT produz uma imensa quantidade de dados, enquanto que a Computação em Nuvem é capaz de fornecer mecanismos para armazenamento e processamento destes dados. De fato, aplicações podem utilizar os recursos virtualmente ilimitados de plataformas na nuvem para processar e apresentar as informações aos usuários. Outras características da computação em nuvem importantes para IoT são: escalabilidade inerente da tecnologia; aumento da segurança, pois o controle de acesso à informação é realizado na nuvem, a qual dispõe de poder computacional adequado para essa tarefa; maior possibilidade de alcance; dentre outros. Esta integração é chamada de *Cloud of Things* (CoT) e atualmente está sendo amplamente discutida [Botta et al. 2016].

Apesar dos imensos benefícios desta integração, é importante que as soluções assegurem as propriedades de segurança e, principalmente, a privacidade dos dados dos usuários gerados pelos dispositivos de IoT, uma vez que estes saem da esfera de controle do usuário ao serem enviados para a nuvem. Neste sentido, este trabalho propõe uma arquitetura tolerante a falhas que provê privacidade em *Cloud of Things*, permitindo que o usuário controle o acesso aos dados gerados pelos dispositivos que estão em seu controle. A arquitetura proposta permite um controle de acesso de granularidade fina sobre os dados, uma vez que os protocolos e controles são executados nos dispositivos de IoT

e não na borda da rede por um *gateway*, o qual é também um único ponto de falha que pode quebrar a disponibilidade ou as propriedades de segurança das aplicações caso seja comprometido.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 discute os principais trabalhos relacionados com a arquitetura proposta, que é detalhada na Seção 3. Uma ampla discussão relacionada com as soluções propostas é apresentada na Seção 4. Finalmente, as conclusões e trabalhos futuros são discutidos na Seção 5.

2. Trabalhos Relacionados

O modelo de serviços utilizado na computação em nuvem pode ser estendido para a integração com a IoT, de forma que as funcionalidades advindas da IoT possam ser oferecidas como serviço na nuvem. Por exemplo, sensores de temperatura espalhados pela cidade podem enviar seus dados para a nuvem, onde um serviço fornece, de maneira ubíqua e segura, acesso à essa informações para usuários em diversas plataformas.

Diversos trabalhos implementam a integração de IoT e computação em nuvem utilizando esta técnica. O IoTCloud [Fox et al. 2012] é uma plataforma de código aberto que tem por objetivo integrar os dispositivos IoT com a nuvem para o gerenciamento dos mesmos. Já o Nimbits [Sautner 2017] oferece uma solução a ser utilizada em plataformas de nuvem onde é possível coletar e processar dados de sensores.

O conceito de *Cloud of Things* envolve a integração de vários elementos, aumentando a complexidade das soluções de segurança que precisam ser adotadas [Roman et al. 2013]. Em redes IoT domésticas, o armazenamento dos dados do usuário na nuvem gera um problema relacionado a privacidade, pois a partir deste momento os dados estão sob controle de uma entidade diferente, e portanto podem ser utilizados para fins que o usuário não aprova. Para atender os requisitos de privacidade do usuário é preciso que apenas entidades previamente autorizadas manuseiem seus dados. Neste sentido, o SensorCloud [Eggert et al. 2014] propõe uma plataforma baseada na nuvem que integra redes de sensores e a Internet. Uma arquitetura baseada em camadas é implementada onde RSSFs conectam-se à nuvem por meio de pontos de confiança (dispositivos de borda) que são responsáveis pela *(i)* comunicação com a nuvem e *(ii)* aplicação de segurança. A arquitetura envolve apenas a comunicação a partir do ponto de confiança, o funcionamento da rede de sensores não é abordado.

O *User-driven Privacy Enforcement for Cloud-based Services in the IoT* (UPECSI) [Henze et al. 2016] estende o SensorCloud, implementando uma solução voltada a privacidade que abrange desde o processo de desenvolvimento de um serviço em nuvem até o usuário. Uma Linguagem de Desenvolvimento de Privacidade (do inglês *Privacy Development Language* (PDL)) foi desenvolvida para facilitar o desenvolvimento de serviços de nuvem com privacidade. A utilização da PDL permite ao desenvolvedor fornecer informações detalhadas sobre quais dados e como eles são utilizados pela aplicação. Esta funcionalidade é então utilizada pelo usuário para habilitar (ou não) certas funções do serviço de acordo com suas preferências. A segurança entre as redes IoT do usuário e a nuvem é realizada por um Ponto de Aplicação de Privacidade (do inglês *Privacy Enforcement Points* (PEP)). O PEP é um dispositivo na borda da rede de sensores que possui capacidade computacional e fonte de energia ilimitada, dessa forma é capaz de executar as tarefas necessárias para garantir a segurança e privacidade dos dados do usuário. Esta

arquitetura permite ao usuário alterar a Política de Privacidade de acordo com suas preferências, fornecendo um alto nível de controle sobre os dados armazenados na nuvem.

O Ponto de Aplicação de Privacidade é a ultima entidade sob o controle do usuário, como tal dispositivo não possui capacidades limitadas, é possível aplicar mecanismos de segurança robustos, como por exemplo a utilização do TLS na camada de transporte. De acordo com a Política de Privacidade configurada pelo usuário, o PEP determina quais dados podem sair da sua esfera de controle, e para quais serviços tais dados devem ser enviados. Os dados são armazenados na nuvem em forma cifrada, e a chave utilizada nesta operação é alterada periodicamente, possibilitando a remoção do acesso de um determinado serviço de nuvem quando desejado. Infraestrutura de Chave Pública (PKI) é utilizada para cifrar a chave simétrica que cifra os dados, a chave simétrica é cifrada uma vez para cada serviço que possui autorização sob os dados. Este mecanismo possibilita que apenas os serviços tenham acesso aos dados, nem mesmo o provedor de nuvem (que fornece o armazenamento) tem condições de acessar os dados do usuário.

Não apenas o acesso aos dados é requisito de privacidade, a maneira como o provedor e os serviços de nuvem manuseiam os dados do usuário também devem ser levados em conta. Por exemplo, deve ser possível estipular um tempo de vida para o dado armazenado, ou até mesmo a localização de armazenamento do dado [Henze et al. 2013a]. A arquitetura UPECSI utiliza Anotações de Manuseio de Dados (do inglês *Data Handling Annotations*) [Henze et al. 2013b] para aplicar os requisitos de privacidade, tais anotações são enviadas com os dados para o provedor de nuvem, indicando como eles devem ser manuseados. Como mencionado anteriormente, a PDL também gera dados que possibilitam auditoria e monitoramento dos serviços de nuvem.

3. Arquitetura para Privacidade em *Cloud of Things*

Seguindo a tendência atual em que os dispositivos de uma rede IoT devem possuir capacidades de comunicação direta com a Internet, este trabalho propõe a remoção do Ponto de Aplicação de Privacidade. Os próprios dispositivos devem possuir funcionalidades de segurança e privacidade, e para tanto as funcionalidades do PEP foram categorizadas em protocolos com propósitos equivalentes, mas com implementação para dispositivos de baixo poder computacional. A arquitetura UPECSI envolve vários aspectos de um sistema de *Cloud of Things*, desde o desenvolvimento de serviços da nuvem até os protocolos de comunicação entre as redes IoT e o provedor de nuvem. O foco deste trabalho é a comunicação entre os dispositivos IoT e a nuvem, portanto os mecanismos definidos pela UPECSI que não são referentes a este aspecto não são mencionados na proposta, tais como a PDL citada na Seção 2.

A abordagem proposta apresenta pelo menos as seguintes vantagens: *(1)* melhora a tolerância a falhas da rede pois remove o dispositivo de borda, o qual propicia um ponto único de falha da arquitetura; *(2)* otimiza a segurança da rede pois remove um componente responsável por todas as tarefas relacionadas à segurança e, consequentemente, pode quebrar as propriedades de segurança do sistema uma vez que seja comprometido por um ataque bem sucedido; e *(3)* a execução da aplicação de mecanismos de segurança e privacidade nos dispositivos possibilita um controle fino sob os dados, pois cada dispositivo pode adotar uma política de segurança diferente.

A Figura 1 apresenta uma visão geral da arquitetura proposta, onde um usuário

possui várias redes IoT sob seu controle. O usuário vincula os dispositivos de suas redes ao provedor de nuvem, assim sendo os dados gerados por eles são armazenado na nuvem. Os serviços de nuvem com autorização podem então processar os dados e apresentá-los ao usuário. Uma Terceira Parte Confiável (TPC) é responsável por auditar e monitorar os serviços de nuvem, prover políticas de privacidade padrão (para usuários iniciantes) e participar de alguns mecanismos de segurança e privacidade envolvendo a comunicação dos dispositivos com a nuvem.

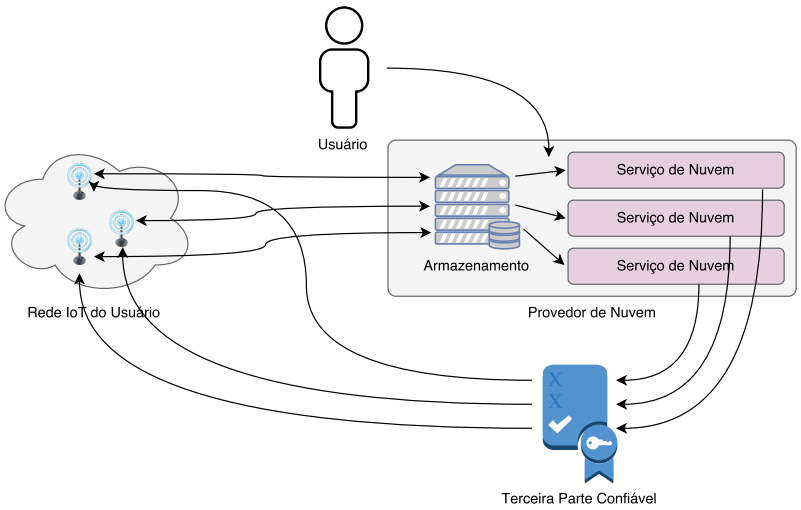


Figura 1. Arquitetura para Privacidade em *Cloud of Things*

Os dispositivos IoT armazenam os dados por eles gerados na nuvem de acordo com a Política de Privacidade. A aplicação da política possibilita que dados: *(1)* não sejam enviados para a nuvem, *(2)* sejam enviados cifrados, ou até mesmo *(3)* sejam enviados sem criptografia. Após o envio os requisitos de privacidade são de responsabilidade dos serviços de nuvem, pois os mesmos possuem as chaves necessárias para acessar os dados.

3.1. Vinculação dos Dispositivos ao Provedor de Nuvem

Assim como geralmente ocorre em provedores de nuvem privados, primeiramente o usuário precisa cadastrar uma conta para então ter acesso aos serviços. Após o processo de registro os usuários podem vincular seus dispositivos para então armazenar os dados gerados. O processo de vinculação é realizado com a utilização do protocolo OAuth 2.0 [Hardt 2012], um protocolo de código aberto para autorização segura. Após o processo de vinculação, os dispositivos podem enviar os dados gerados para a nuvem sem possuir as credenciais do usuário.

O protocolo OAuth 2.0 exige que a comunicação entre as partes seja segura para que o mesmo proporcione autorização segura. Na Internet essa exigência é cumprida com a utilização do procolo TLS, utilizado em conjunto com o TCP. Devido às características limitadas dos dispositivos IoT, ambos os protocolos (TCP e TLS) apresentam um custo muito alto para serem implementados nos dispositivos, dessa forma outra solução precisa ser utilizada. Este assunto atualmente está sendo estudado de forma ativa pela comunidade acadêmica, e já existem diversas propostas para prover canais seguros em redes IoT. Particularmente interessante, uma adaptação do Datagram Transport Layer

Security (DTLS)[McGrew and Rescorla 2010] está sendo estudada pelo grupo de trabalho *DTLS In Constrained Environments* (DICE) [Tschofenig and Fossati 2016]. O protocolo de vinculação é agnóstico ao protocolo da camada de transporte (que deve prover comunicação segura), portanto este trabalho não impõe a utilização de um protocolo específico.

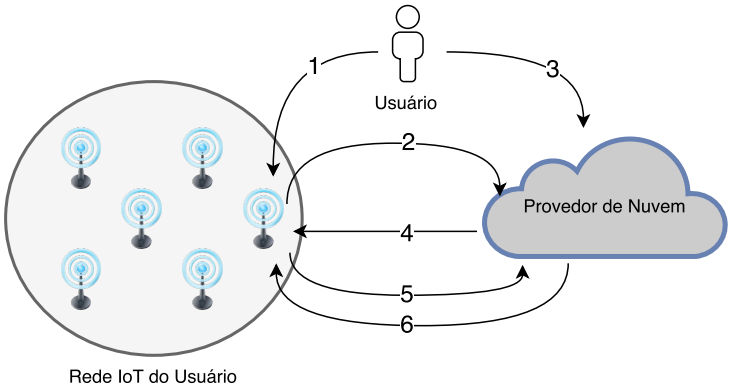


Figura 2. Processo de vinculação.

O usuário pode vincular um único dispositivo ou múltiplos dispositivos por vez. Neste trabalho mostramos como vincular um único dispositivo por vez, pois o processo de vinculação de múltiplos dispositivos pode ser realizado apenas com a utilização de um dispositivo de borda, de forma que todos os dispositivos IoT compartilham a mesma identificação e o usuário não será capaz de aplicar uma política de privacidade diferente a cada dispositivo. A vinculação é realizada utilizando o processo definido como Concessão de Código de Autorização (do inglês *Authorization Code Grant*) do protocolo OAuth, este tipo de permissão necessita da intervenção do usuário apenas na fase de configuração. Após receber a autorização, os dispositivos podem enviar dados para a nuvem sem utilizar as credenciais do usuário.

A Figura 2 apresenta o processo de vinculação para um dispositivo, o qual possui os seguintes passos:

1. O usuário inicia o processo como o *Resource Owner* acessando a página web de vinculação do dispositivo;
2. O dispositivo solicita o Código de Autorização para a nuvem;
3. O usuário é então redirecionado a uma página para inserir suas credenciais;
4. Após receber as credenciais do usuário a plataforma de nuvem envia o Código de Autorização ao dispositivo;
5. De posse do Código de Autorização, o dispositivo solicita um Token de Acesso;
6. A plataforma de nuvem envia o Token de Acesso.

Ao finalizar o processo o dispositivo possui um Token de Acesso que precisa ser anexado a todas as mensagens enviadas à nuvem. O provedor de nuvem autoriza o recebimento e armazenamento dos dados de acordo com o Token de Acesso. O processo de vinculação para múltiplos dispositivos é o mesmo, mas é executado pelo dispositivo de borda, que, ao receber o Token de Acesso, o envia a todos os dispositivos da rede. Como todos os dispositivos possuem o mesmo token, não há distinção de dispositivos pelo provedor de nuvem. Após vincular todos os dispositivos o usuário também pode criar redes lógicas para facilitar o gerenciamento.

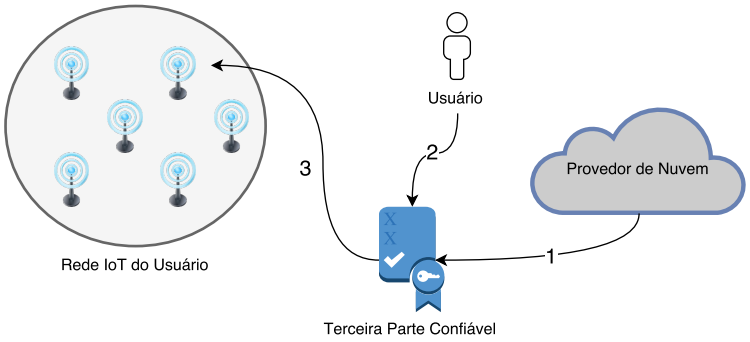


Figura 3. Atualização da Política de Privacidade.

3.2. Aplicação da Política de Privacidade

Uma Política de Privacidade (PP) define se um dado pode ser armazenado na nuvem, como ele será armazenado (cifrado ou não) e o que pode ser feito com este dado por um serviço de nuvem. Diferente de políticas de privacidade comuns, onde o usuário tem apenas a opção de aceitar ou rejeitar ela por completo, na arquitetura proposta o usuário tem o poder de alterar a sua política de privacidade com o tempo. Esta funcionalidade permite ao usuário assumir controle real sob os dados que envia para a nuvem. Serviços de nuvem fornecem ao usuário uma interface onde é possível analisar quais são os dados utilizados pelo serviço, e para quais propósitos, os usuários então podem habilitar ou desabilitar funções específicas do serviço de modo a restringir o acesso a determinados dados de seus dispositivos. Por exemplo, um usuário pode desabilitar a utilização de dados de localização por um determinado serviço se assim desejar.

A Política de Privacidade é aplicada toda vez que um dispositivo IoT envia dados para a nuvem, tal aplicação é realizada pelo próprio dispositivo. A Figura 3 apresenta o processo de atualização de uma PP:

1. O serviço de nuvem envia a Política de Privacidade à Terceira Parte Confiável (TPC) para auditoria. Ao receber a Política de Privacidade, a TPC audita a PP e gera uma Configuração de Privacidade (CP);
2. O usuário acessa a CP por meio de uma interface (navegador, *smartphone*, etc) e realiza as configurações desejadas;
3. A TPC envia aos dispositivos a CP revisada pelo usuário.

3.3. Controle de Acesso dos Dados

Após vincular os dispositivos ao provedor de nuvem e configurar as Políticas de Privacidade referentes aos serviços utilizados, é necessário garantir que os dados vindo dos dispositivos do usuário são armazenados e acessados apenas por entidades autorizadas.

O controle de acesso é implementado utilizando mecanismos baseados em criptografia. Assim como no processo de vinculação, a comunicação entre os dispositivos IoT e a nuvem deve ser segura, o que é garantido por um protocolo da camada de transporte. Os dados sensíveis vindo dos dispositivos são armazenados cifrados na nuvem (dados que não são sensíveis podem ser armazenados sem criptografia), dessa forma é prevenido o acesso ao dado até mesmo pelo provedor de nuvem. As chaves utilizadas para decifrar os dados são armazenadas em um repositório de chaves (também na nuvem).

Antes do envio do dado, o dispositivo IoT deve filtrá-lo de acordo com a Configuração de Privacidade, dessa forma decidindo se o dado deve deixar a esfera de controle do usuário. O dado é então cifrado com um algoritmo de criptografia simétrica. A chave utilizada por tal algoritmo é então cifrada com a chave pública dos serviços autorizado a acessar o respectivo dado. Caso haja mais de um serviço utilizando o mesmo dado a chave simétrica deve ser cifrada uma vez para cada serviço. O dado é armazenado apenas uma vez no provedor, independente do número de serviços que o acessarão.

A chave simétrica é atualizada periodicamente, prevenindo que novos serviços acessem dados armazenados previamente à sua autorização, ou que serviços com autorização cancelada continuem a acessar os dados do usuário. As chaves antigas continuam no repositório de chaves, possibilitando o acesso a dados já armazenados por novos serviços.

Como pode ser visto, os dispositivos IoT executam operações criptográficas. No caso de dispositivos limitados, com fonte de energia restrita, operações criptográficas podem diminuir significativamente o tempo de vida da rede. Duas abordagens são propostas para mitigar este problema, as chaves criptográficas são gerenciadas pelos dispositivos IoT (Seção 3.3.1) ou pela Terceira Parte Confiável (Seção 3.3.2). A seguir tais abordagens são apresentadas e suas vantagens e desvantagens discutidas.

3.3.1. Gerenciamento das Chaves pelos Dispositivos IoT

A Figura 4 apresenta a abordagem onde o próprio dispositivo é responsável por criar as chaves simétricas, cifrá-las com as chaves públicas dos serviços e enviá-las para o repositório de chaves no provedor de nuvem.

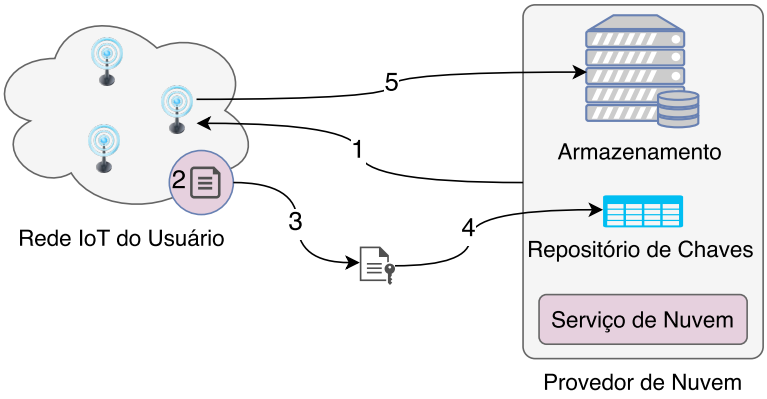


Figura 4. Gerenciamento das Chaves pelos Dispositivos IoT.

Esta abordagem é proposta para dispositivos que possuam capacidade computacional e energética suficientes para executar as seguintes tarefas (Figura 4) sem impactar significativamente seu tempo de vida:

1. Receber as chaves públicas dos serviços pelo provedor de nuvem;
2. Criar periodicamente chaves simétricas;
3. Cifrar as chaves simétricas com as chaves públicas dos serviços autorizados;
4. Enviar as chaves simétricas cifradas ao repositório de chaves;
5. Enviar dados cifrados ao provedor de nuvem.

3.3.2. Gerenciamento das Chaves por uma Terceira Parte Confiável

A Figura 5 apresenta a segunda abordagem, onde uma Terceira Parte Confiável é responsável pela maioria das tarefas relacionadas ao controle de acesso dos dados, atenuando a carga nos dispositivos IoT. A TPC cria as chaves simétricas, recebe as chaves públicas dos serviços, cifra as chaves simétricas e as envia ao repositório de chaves. As chaves simétricas também são enviadas aos dispositivos.

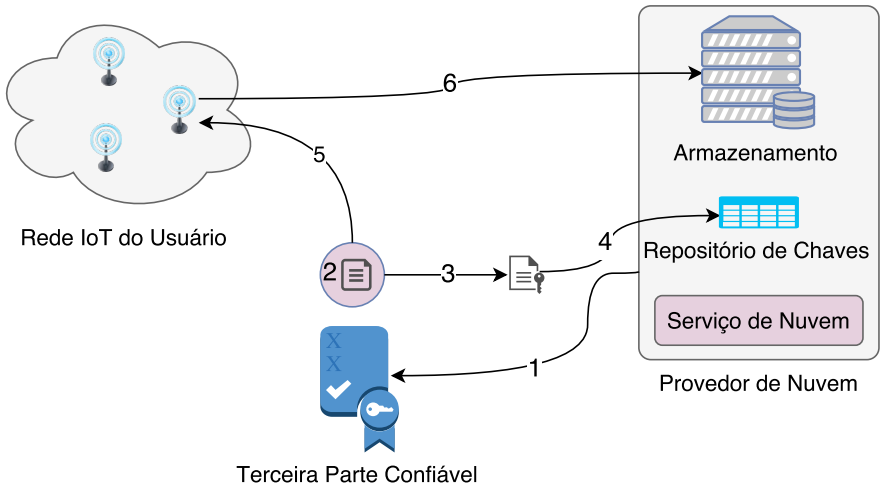


Figura 5. Gerenciamento das Chaves por uma TPC.

Esta abordagem é vantajosa para dispositivos limitados, onde as tarefas migradas à TPC diminuiriam o atraso na comunicação e aumentariam a expectativa de vida do aparelho. Nesta abordagem são executados os seguintes passos:

1. **TPC**: Recebe as chaves públicas dos serviços pelo provedor de nuvem;
2. **TPC**: Cria periodicamente chaves simétricas;
3. **TPC**: Cifra as chaves simétricas com as chaves públicas dos serviços autorizados;
4. **TPC**: Envia as chaves simétricas cifradas ao repositório de chaves;
5. **TPC/Dispositivo IoT**: Envia (de forma segura) as chaves simétricas aos dispositivos IoT;
6. **Dispositivo IoT**: Envia dados cifrados ao provedor de nuvem.

Em ambas as abordagens os dados são cifrados duas vezes, uma vez na camada de transporte, pelo protocolo que garante um canal seguro, e outra vez na camada de aplicação, pelo controle de acesso aos dados. A criptografia efetuada na camada de aplicação é sempre necessária, pois o dado é armazenado em forma cifrada. Consequentemente, para prevenir que o dado seja cifrado duas vezes é necessário remover a cifragem realizada na camada de transporte. A mensagem enviada pelos dispositivos contem os dados cifrados e o Token de Acesso, que então precisa ser protegido na camada de aplicação. Isso pode ser alcançado simplesmente compartilhando uma chave entre o provedor de nuvem e o dispositivo de forma segura. Este processo pode ser realizado na etapa de vinculação, onde é utilizado um canal seguro, desta forma os dispositivos podem cifrar o Token de Acesso com a chave compartilhada e enviar todas as informações de forma segura.

3.4. Políticas de Privacidade Flexíveis

O controle de acesso aos dados fornecido pela arquitetura proposta permite apenas o acesso de agentes previamente autorizados. Entretanto, dependendo de certas situações, seria conveniente diminuir a privacidade do usuário, como por exemplo em emergências médicas. Em situação normal os dados do usuário iriam apenas para o médico vinculado, mas em uma emergência qualquer médico disponível teria acesso aos dados do usuário para poder trata-lo.

Para tratar situações como a descrita o usuário deve criar Políticas de Privacidade alternativas. Tais políticas são acionadas por eventos internos, definidos por valores coletados que ultrapassam um limiar, ou externos, comumente gerados pelos serviços. O processo de criação e atualização da PP Flexível é idêntico ao da PP comum (ver Seção 3.2).

Dados que serão liberados quando políticas flexíveis forem acionadas passam por um processo de criptografia semelhante aos dados comuns, no entanto a chave simétrica utilizada não é criptografada com a chave pública dos serviços e sim com uma chave pública criada pela TPC. Quando eventos indicam o acionamento da PP Flexível, o dispositivo aciona a TPC, que por sua vez envia a chave privada (referente a chave pública criada) aos serviços que devem ter acesso aos dados.

Para prevenir acesso a dados gerados quando PP Flexíveis deixam de estar em vigor, o par de chaves pública/privada deve ser reconfigurado, i.e., a chave privada antiga deve ser revogada e uma nova chave privada deve ser obtida. Esse mecanismo, diferente da aplicação de privacidade em situações comuns, requer a utilização da TPC, pois a mesma apresenta maior confiabilidade que os dispositivos IoT.

4. Discussões

Esta seção apresenta uma ampla discussão a respeito de aspectos relacionados com a arquitetura apresentada na seção anterior. Primeiramente é discutida a abordagem de transferência das funcionalidades do *gateway* para os dispositivos, principalmente em relação a segurança da comunicação fim-a-fim. Em seguida é discutida a proposta desse trabalho, indicando suas limitações, vantagens e desvantagens.

4.1. Comunicação Direta dos Dispositivos com a Internet

Um aspecto importante da arquitetura proposta é a utilização de protocolos equivalentes aos da Internet nos dispositivos IoT, proporcionando a habilidade de comunicação direta entre os dispositivos e a nuvem. Baseado no UPECSI [Henze et al. 2016], este trabalho transfere as funcionalidades do Ponto de Aplicação de Privacidade para os dispositivos IoT, adaptando tais funcionalidades de acordo com a natureza limitada dos dispositivos. Apesar dos dispositivos se comunicarem diretamente com a nuvem, *gateways* podem ser utilizados para tradução de protocolos. Por exemplo, o IEEE 802.15.4 define um pacote de no máximo 127 bytes, mas muitos protocolos da Internet possuem cabeçalhos e mensagens que utilizam a maior parte deste pacote, deixando pouco espaço para dados da aplicação (por exemplo, o cabeçalho do IPv6 possui no mínimo 40 bytes).

O UPECSI define mecanismos de comunicação entre o *gateway* e a nuvem, este trabalho propõe uma arquitetura segura entre os dispositivos IoT e a nuvem, relativos apenas à camada de aplicação. Contudo, uma discussão a respeito da segurança na camada

de transporte é pertinente, pois é um tópico sendo amplamente estudado. Segurança nas camadas de Enlace e de Rede na Internet das Coisas, apesar de apresentar muitos desafios, pode ser considerada mais madura, já que o IEEE 802.15.4, o 6LoWPAN e o RPL [Winter 2012] são protocolos consolidados.

O grande dificultador da implementação de mecanismos de segurança na Internet das Coisas é o custo adicional causado pela quantidade de processamento e pelas mensagens extras necessárias para realização de operações de criptografia. Os protocolos da Internet utilizam a Criptografia de Chave Pública [Salomaa 2013], ou criptografia assimétrica, onde uma chave pública é utilizada para cifrar e uma chave privada para decifrar a mensagem. A criação e troca de chaves entre os pares são especialmente custosas, e por isso muitos trabalhos assumem o carregamento prévio das chaves nos dispositivos. Neste trabalho é assumida a utilização de um protocolo de camada de transporte que assegure um canal seguro e confiável. Apesar de várias propostas atuais com este fim [Sethi et al. 2012, Hummen et al. 2014, Tschofenig and Fossati 2016], o IETF atualmente trabalha para definir um protocolo padrão para utilização em redes de dispositivos limitados.

4.2. Arquitetura Proposta

Esta seção traz uma discussão acerca de aspectos relacionados com os protocolos implementados nos dispositivos de IoT.

4.2.1. Vinculação

O processo de vinculação é uma aplicação direta do protocolo OAuth 2.0. Visto que esse processo acontece apenas uma vez, a carga adicionada aos dispositivos pode ser negligenciada. Esse processo também pode ser realizado por um *gateway*, como definido no UPECSI, dessa forma todos os dispositivos partilham de um mesmo *Access Token*. Esse recurso agiliza a etapa de vinculação da rede, mas diminui a granularidade de controle do usuário, pois todos os dispositivos serão obrigatoriamente vinculados ao mesmo provedor.

O protocolo OAuth 2.0 presume a utilização de um canal seguro entre o dispositivo e o provedor, pois não dispõe de recursos para manter a confidencialidade das mensagens trocadas. Caso o processo de vinculação seja realizado por cada dispositivo, é necessário que eles disponham de uma interface web. Levando em conta sua natureza limitada, é importante que a implementação dessa etapa seja realizada de forma otimizada.

4.2.2. Aplicação da Política de Privacidade

A aplicação da Política de Privacidade é realizada da mesma forma que no UPECSI, transferindo o processo do *gateway* ao dispositivo. A alteração da PP é realizada através do serviço de nuvem, por meio de uma interface *Web*. O dispositivo IoT recebe a nova informação assim que gerada, dessa forma o processo de atualização da PP não ocasiona uma carga considerável aos dispositivos, já que não deve acontecer com frequência. O procedimento de aplicação da PP é realizado por cada dispositivo, o qual de posse da PP deve filtrar todas as mensagens a serem enviadas ao provedor (previamente ao envio).

Como esta filtragem não exige grande processamento, sua implementação também não deve ocasionar aumento significativo da carga no dispositivo.

4.2.3. Controle de Acesso aos Dados

A privacidade do usuário é garantida por meio de criptografia. O dado é armazenado no provedor de forma cifrada, e apenas os serviços autorizados possuem a chave para decriptá-lo. Este mecanismo permite que apenas entidades previamente autorizadas tenham acesso ao dado. Como já discutido, operações criptográficas podem ser custosas para dispositivos IoT. Embora os dispositivos IoT precisem cifrar os dados, algumas operações podem ser transferidas para a Terceira Parte Confiável, alivando a carga nos dispositivos. Nesta abordagem o gerenciamento das chaves é realizado pela TPC, e os dispositivos precisam apenas receber as chaves e cifrar os dados. Esta abordagem é recomendada a redes IoT onde os dispositivos possuam recursos limitados.

Os próprios dispositivos podem ser responsáveis por criar as chaves simétricas periodicamente e enviá-las aos serviços autorizados, sem o intermédio da Terceira Parte Confiável. Este método introduz maior carga aos dispositivos, que ficam responsáveis por tarefas de alto custo computacional: criação de chaves simétricas, encriptação das chaves simétricas com as chaves públicas dos serviços e envio das chaves encriptadas aos serviços. Vale notar que a encriptação e o envio é realizada uma vez para cada serviço sendo utilizado, e esse processo repete-se periodicamente, deve-se ter cautela ao utilizar tal método, pois caso o período de renovação das chaves seja muito pequeno os dispositivos podem ter uma sobrecarga. A vantagem desta abordagem é a remoção da Terceira Parte Confiável do processo, tornando o protocolo mais seguro pois diminui a superfície de ataque. Este método é indicado para redes IoT de moderada capacidade computacional e de grande disponibilidade de energia.

Em ambas as abordagens o dado será cifrado duas vezes, uma na camada de transporte, por um protocolo que garanta um canal seguro, e outra na camada de aplicação, pela arquitetura proposta neste trabalho. A cifra realizada pela camada de aplicação será sempre necessária, pois é nesta forma que o dado é armazenado na nuvem. Para remover a criptografia da camada de transporte é preciso assegurar o envio confidencial da mensagem na camada de aplicação. A mensagem enviada pelos dispositivos possui os dados cifrados e o *Access Token*, que agora precisa ser protegido, o que pode ser alcançado compartilhando uma chave entre a plataforma de nuvem e o dispositivo anteriormente ao envio dos dados. Este compartilhamento pode ser realizado ao final do processo de vinculação, o qual utiliza um canal seguro. Desta forma, os dispositivos IoT podem cifrar o *Access Token* e os dados gerados.

4.2.4. Políticas de Privacidade Flexíveis

Políticas de Privacidade Flexíveis foram propostas para aumentar a flexibilidade da arquitetura. Este protocolo possibilita ao usuário definir limites, para que serviços de nuvem sejam habilitados ou desabilitados. Este mecanismo é especialmente útil em situações excepcionais, onde pode ser mais vantajoso abdicar da privacidade.

Uma vez que o principal propósito das Políticas de Privacidade Flexíveis são situações excepcionais, pode ser o caso em que o próprio dispositivo IoT deixe de funcionar, desta forma o acionamento da PPF é possível apenas por um agente externo. Para assegurar o correto funcionamento em uma situação como esta a TPC é responsável pela aplicação da PPF, gerenciando as chaves e autorizando os serviços quando necessário. Este método torna o protocolo mais confiável e introduz pouca carga aos dispositivos, pois apenas uma mensagem é enviada do dispositivo à TPC para ativação da PPF.

5. Conclusões

A integração de Internet das Coisas com Computação em Nuvem traz muitos benefícios para as aplicações, pois enquanto a primeira produz uma grande quantidade de dados, a segunda possui poder suficiente para armazenar e processar estas informações, incluindo soluções para *big data*. Um aspecto extremamente relevante nesta integração é a segurança, principalmente relacionada com a privacidade dos dados dos usuários, visto que os mesmos saem da esfera de controle do usuário ao serem enviados para a nuvem.

Visando contornar ou pelo menos mitigar este problema, este trabalho propõe uma arquitetura para que os usuários possam controlar o acesso aos dados gerados pelos dispositivos de IoT em seu controle. A solução proposta possibilita uma granularidade fina neste controle, i.e., o controle pode ser realizado por dispositivo. Como trabalhos futuros, pretende-se definir cenários e analisar o desempenho das soluções propostas.

Referências

Akyildiz, I. F. and Vuran, M. C. (2010). *Wireless sensor networks*, volume 4. John Wiley & Sons.

Botta, A., de Donato, W., Persico, V., and Pescapé, A. (2016). Integration of cloud computing and internet of things: a survey. *Future Generation Computer Systems*, 56:684–700.

Chui, M., Löffler, M., and Roberts, R. (2010). The internet of things. *McKinsey Quarterly*, 2(2010):1–9.

Deering, S. E. (1998). Internet protocol, version 6 (ipv6) specification. RFC 2460.

Eggert, M., Häußling, R., Henze, M., Hermerschmidt, L., Hummen, R., Kerpen, D., Pérez, A. N., Rumpe, B., Thißen, D., and Wehrle, K. (2014). Sensorcloud: Towards the interdisciplinary development of a trustworthy platform for globally interconnected sensors and actuators. In *Trusted Cloud Computing*, pages 203–218. Springer.

Fox, G. C., Kamburugamuve, S., and Hartman, R. D. (2012). Architecture and measured characteristics of a cloud based internet of things. In *Collaboration Technologies and Systems (CTS), 2012 International Conference on*, pages 6–12. IEEE.

Granjal, J., Monteiro, E., and Silva, J. S. (2015). Security in the integration of low-power wireless sensor networks with the internet: A survey. *Ad Hoc Networks*, 24:264–287.

Group, W. W. P. A. N. W. W. (2006). *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*. IEEE Standard for Information Technology.

Hardt, D. (2012). The oauth 2.0 authorization framework. RFC 6749.

Henze, M., Großfengels, M., Koprowski, M., and Wehrle, K. (2013a). Towards data handling requirements-aware cloud computing. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 2, pages 266–269. IEEE.

Henze, M., Hermerschmidt, L., Kerpen, D., Häußling, R., Rumpe, B., and Wehrle, K. (2016). A comprehensive approach to privacy in the cloud-based internet of things. *Future Generation Computer Systems*, 56:701–718.

Henze, M., Hummen, R., and Wehrle, K. (2013b). The cloud needs cross-layer data handling annotations. In *Security and Privacy Workshops (SPW), 2013 IEEE*, pages 18–22. IEEE.

Hummen, R., Shafagh, H., Raza, S., Voig, T., and Wehrle, K. (2014). Delegation-based authentication and authorization for the ip-based internet of things. In *Annual IEEE International Conference on Sensing, Communication, and Networking*.

Kushalnagar, N., Montenegro, G., and Schumacher, C. (2007). Ipv6 over low-power wireless personal area networks (6lowpans): overview, assumptions, problem statement, and goals. Technical report.

McGrew, D. and Rescorla, E. (2010). Datagram transport layer security (dtls) extension to establish keys for secure real-time transport protocol (srtp).

Postel, J. (1980). User datagram protocol (udp). RFC 768.

Postel, J. (1981). Transmission control protocol (tcp). RFC 793.

Roman, R., Zhou, J., and Lopez, J. (2013). On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, 57(10):2266–2279.

Salomaa, A. (2013). *Public-key cryptography*. Springer Science & Business Media.

Sautner, B. (2017). Nimbits. `http://bsautner.github.io/com.nimbits/`, Acessado em Março de 2017.

Sethi, M., Arkko, J., and Keränen, A. (2012). End-to-end security for sleepy smart object networks. In *IEEE Conference on Local Computer Networks Workshops*.

Sundmaeker, H., Guillemin, P., Friess, P., and Woelfflé, S. (2010). Vision and challenges for realising the internet of things. *Cluster of European Research Projects on the Internet of Things, European Commision*.

Tschofenig, H. and Fossati, T. (2016). Transport layer security (tls)/datagram transport layer security (dtls) profiles for the internet of things. RFC 7925.

Winter, T. (2012). Rpl: Ipv6 routing protocol for low-power and lossy networks. RFC 6550.

Armazenamento Seguro em Nuvem com Detecção de Falhas Bizantinas

Clésio Matos[1], Fabíola Greve[1]

[1]Departamento de Ciência da Computação – Universidade Federal da Bahia (UFBA)
Av. Adhemar de Barros, s/n – CEP 40.170-110 – Salvador – BA– Brasil

{clesiormatos,fabiola}@dcc.ufba.br

Abstract. *Failure detectors are essential services for designing fault tolerant applications as they provide information about system process failures. In modern distributed systems that have inherently dynamic behavior, such as cloud computing, the design of these services is a challenge, especially in the treatment of byzantine failures. Thus, this paper aims at proposing a secure implementation for detecting byzantine faults in a dynamic and asynchronous environment, applying it in the cloud storage and data replication. The algorithm brings the interesting feature to propose weak time constraints of an asynchronous system to the cloud, not using time limits for failure detection. Simulation experiments have validated the presented approach.*

Resumo. *Detectores de falhas são serviços essenciais para o projeto de aplicações tolerantes a falhas, pois provêm informações sobre falhas de processos do sistema. Nos sistemas distribuídos modernos que possuem comportamento inerentemente dinâmico, dentre eles a computação em nuvem, a concepção destes serviços é um desafio, principalmente no tratamento de falhas bizantinas. Por tanto, este artigo tem por objetivo propor uma implementação segura para a detecção de falhas bizantinas num ambiente dinâmico e assíncrono, aplicando-o no armazenamento e replicação de dados da nuvem. O algoritmo possui a característica interessante de propor as fracas restrições de tempo de um sistema assíncrono para a nuvem, não utilizando limites temporais para detecção de falhas. A solução foi implementada e os resultados de simulação são apresentados como forma de validar a abordagem apresentada.*

1. Introdução

A computação em nuvem é um dos grandes cenários da computação distribuída moderna que evolui para integração de sistemas dinâmicos e auto-organizáveis. Ela propõe um novo paradigma para a gestão de infraestrutura, oferecendo possibilidades para implantar diversos recursos em ambientes distribuídos através de um modelo de *utility computing*. Este gerenciamento de nuvem permite o provisionamento de recursos computacionais sob demanda, operado através de máquinas virtuais em *data centers* de provedores distribuídos [Ganesh et al. 2014].

Neste contexto, novos recursos computacionais são criados em tempo de execução com base em solicitações existentes, derivando redes de filiação dinâmicas ao longo do tempo. Como a arquitetura em nuvem tipicamente combina um grande número

de módulos heterogêneos e dispersos geograficamente, ligados através da internet, a confiabilidade da aplicação não depende só do sistema em si, mas também do nó que esteja implantada e da internet imprevisível [Ganesh et al. 2014]. Assim, os protocolos distribuídos clássicos tornam-se inadequados para esse novo contexto, uma vez que eles fazem a suposição de que todo o sistema é estático e sua composição é previamente conhecida. Além disso, a computação em nuvem está crescendo suportada por recurso e serviço *on-line*, o que torna mais provável falhas maliciosas e consequências mais graves [Garraghan et al. 2011].

Uma forma de manter a confiabilidade desses sistemas distribuídos é dispor de de mecanismos que os tornem tolerantes a falhas, com recursos que identifiquem e implementem formas de contorno. Na resolução destes problemas a principal dificuldade encontrada é a complexidade de se detectar as falhas existentes no sistema. Assim, o detector de falhas (*FD*) é um serviço fundamental capaz de ajudar no desenvolvimento de sistemas distribuídos tolerantes a falhas, trabalhando com um oráculo distribuído que fornece periodicamente uma lista de processos suspeitos de terem falhado. Neste trabalho, apresenta-se o interesse especial nos detectores de falhas da classe não confiável, denotado por ◊S, que implementa os recursos mínimos para detecção em ambiente assíncrono. Esses *FD*s podem cometer um número arbitrário de erros, sendo que após um tempo, um processo correto nunca é suspeito e todos os processos falhos serão suspeitos permanentemente [Chandra and Toueg 1996].

O serviço de armazenamento (*storage)* é um dos principais recursos da nuvem e está associado a um alto custo de recuperação de falhas, pois todas as réplicas devem conter a mesma informação, ordenada, obtida através de protocolos avançados, tais como a sincronia de visões e o acordo [Greve 2005]. Assim, identificar os processos falhos e evitar sua utilização é essencial para garantir o desempenho do sistema de nuvem.

A replicação de dados na nuvem pode ser feita de forma estática, com um grupo de processos corretos predeterminados na rede, ou dinâmica, onde o grupo de processos é determinado a partir da solicitação e com base em critérios adotados [Greve 2005]. Estes dois modelos utilizam tipicamente um FD para controlar os nós disponíveis para replicação. Neste contexto, encontra-se sistemas de *storage* distribuídos largamente utilizados como o Dynamo da empresa Amazon [DeCandia et al. 2007] e o Cassandra da empresa Facebook [Lakshman and Malik 2009]. Este último, emprega um sistema chamado Zookeeper [Hunt et al. 2010] para controlar a vivacidade dos nós na distribuição do armazenamento.

Grande parte das implementações de detectores de falhas são baseadas numa comunicação entre todos os nós, em que é monitorada a atividade dos nós através da troca periódica de mensagens *heartbeats* e do *timeout* da comunicação. [Hunt et al. 2010]. Porém, avançando neste contexto, alguns trabalhos foram propostos para lidar com dinamismo da rede adotando uma abordagem livre de tempo (não utiliza *timeout*) para efetuar a detecção de falhas em sistemas dinâmicos, assumindo um ambiente assíncrono. Estes artigos se destacam ainda por trabalharem com detecção de falhas malignas (bizantinas) no sistema [Greve et al. 2012, Matos and Greve 2015].

Neste contexto, este trabalho tem o intuito de contribuir com o estudo sobre a detecção de falhas bizantinas para a computação em nuvem com a aplicação, implementação e validação de um algoritmo livre de tempo como detector de falhas no

serviço de *storage* na nuvem. Ele é uma continuidade da pesquisa desenvolvida em [Matos and Greve 2015] e visa apresentar uma simulação da solução proposta em ambiente de nuvem. O FD é utilizado como um oráculo que informa um grupo de processos falhos na rede, incapazes de atender as requisições de armazenamento e replicação com segurança. Para efeito de avaliação, utilizamos o serviço Zookeeper [Hunt et al. 2010] como padrão de comparação para o algoritmo proposto. A comparação tem o duplo propósito de comparar o desempenho das abordagens, além de confrontar duas categorias de detectores: o FD livre de tempo (aqui proposto) e o FD baseado em timeouts (integrado ao Zookeeper).

As principais contribuições deste trabalho são a implementação, simulação e validação de um novo algoritmo de detecção de falhas proposto em [Matos and Greve 2015], com sua subsequente aplicação para o armazenamento confiável da computação em nuvem.

O restante deste artigo está organizado da seguinte forma: A Seção 2 discute os principais trabalhos relacionados. A Seção 3 apresenta o modelo de ambiente do detector de falhas. Na Seção 4 é apresentado o algoritmo. A Seção 5 apresenta a avaliação da proposta. Por fim, as conclusões e trabalhos futuros na Seção 6.

2. Trabalhos Relacionados

Na literatura, encontra-se trabalhos relacionados a detectores de falhas aplicados a varias áreas da computação, na sua maioria, implementações ligadas à falhas benignas e ambiente síncronos. Outro seguimento de trabalhos é focado nas falhas malignas, ou bizantinas, que propõe soluções baseadas no envio de *heartbeats*, através controle do *timeout* da comunicação [Ramasamy and Cachin 2005]. Neste sentido, tem-se a proposta de Sivakami and Nawaz (2011) que apresenta um protocolo de roteamento tolerante a falhas bizantinas. Ela implementa uma busca de melhor caminho através da utilização de um detector de falhas malignas, operando em cada nó do sistema.

Ainda no contexto de falhas bizantinas, encontra-se um novo grupo de trabalhos que aplica tolerância a falhas no ambiente de computação em nuvem. Este é o caso de Fan et. al (2012), que propõe um modelo de organização de processos na nuvem capaz de detectar falhas em componentes. Destaca-se também o trabalho de Garraghan (2011) que desenvolveu um framework para a criação de sistemas tolerantes a falhas bizantinas aplicadas à nuvens federadas, demonstrando os resultados iniciais desta abordagem. Avançando neste contexto, tem-se um grupo de trabalhos que propõe a adoção de uma abordagem livre de tempo para efetuar a detecção de falhas bizantinas em sistemas dinâmicos aplicada em redes móveis [Greve et al. 2012].

As contribuições citadas anteriormente são de grande valia para a evolução da pesquisa de falhas bizantinas, buscando soluções diversas para essa classe de problemas. Entretanto, não fazem a aplicação real da detecção de falhas em ambientes verdadeiramente assíncronos (isentos de limites temporais) com garantias de segurança de comunicação.

Em [Matos and Greve 2015], apresentamos um novo algoritmo de detecção de falhas para o armazenamento da computação em nuvem, que tem como características inovadoras: (i) a adoção de uma estratégia de detecção de falhas sem a utilização de mecanismos de tempo (*time free*), baseado em ambiente assíncrono, modelando assim, a

dinamicidade dos sistemas em nuvem; *(ii)* detecção de falhas na presença de agentes maliciosos, o que constitui um desafio em sistemas distribuídos.

O presente trabalho dá continuidade à proposta trazida por [Matos and Greve 2015] e apresenta os resultados da aplicação e simulação da detecção de falhas bizantinas na computação em nuvem, assumindo as fracas prerrogativas temporais de um sistema assíncrono.

3. Modelo para Detector de Falhas Bizantinas no Armazenamento em nuvem

Nesta seção é discorrido sobre como detector de falhas proposto está inserido no módulo de armazenamento da nuvem, bem como detalhes do ambiente e de seu comportamento.

3.1. Modelo de Nuvem Assíncrono

A computação em nuvem é composta de *data centers* dispersos geograficamente que efetuam a computação e o armazenamento dos dados, uma realidade ainda mais evidenciada assumindo a utilização de nuvens federadas ou nuvens *Mashups* [Garraghan 2011]. A quantidade e qualidade dos processos que compõem cada módulo são dinâmicas, pois a cada momento novos recursos podem ser provisionados para atender novas requisições, ou ainda, requisições antigas podem ser migradas para outros processos servidores, a fim de obter melhores respostas do sistema. Este dinamismo da computação em nuvem provê as seguintes propriedades ao modelo proposto: *i*) população dinâmica de nós, *ii*) conhecimento parcial sobre o conjunto de processos, *iii*) não existem suposições sobre a velocidade relativa dos processos [Fan et. al 2012].

Na computação em nuvem a comunicação entre todos os módulos heterogêneos que compõem o sistema é dada pela troca de pacotes através da internet, estando sujeita a atrasos arbitrários e perdas [Ganesh et al. 2014]. Isto provê mais duas propriedades ao modelo: *iv*) não existem limites temporais para a transferência da mensagem[1], *v*) não existe um relógio global conhecido para a coordenação dos processos dispersos. A partir desta caracterização é possível determinar o modelo de ambiente dinâmico e assíncrono, foco deste trabalho [Greve et al. 2012].

Neste ambiente complexo podem ocorrer falhas em processos e na comunicação por vários motivos: porque um processo parou de funcionar (falha por parada), porque deu lugar a outro processo e esta informação ainda não está presente nos demais nós da rede, ou ainda, porque um processo executou ações não planejadas dentro do sistema (falhas bizantinas) [Fan et. al 2012]. Nesta última está concentrado o interesse deste trabalho. É razoável pensar que dentro de *data centers* as falhas malignas não ocorrem pelo nível de controle implementado nesse ambiente. Contudo, estas falhas podem ser oriundas de incidentes (ex. erros de programação, falhas de hardware) ou por meio de intrusões [Verissimo et al. 2003] (ataques de segurança bem sucedidos, que conduzem o sistema ao comportamento de falha).

[1] Ressalta-se que o algoritmo proposto está inserido dentro da arquitetura da nuvem, e ao invés de se contrapor com as garantias de SLA (*service level agreement*) da nuvem, torna-se um suporte interno para assegurar a funcionalidade destas garantias.

3.2. Arquitetura do Sistema

Na computação em nuvem grande parte do armazenamento de dados é feito com replicação no intuito de manter a disponibilidade dos dados e melhorar o desempenho da recuperação de informações. O sistema conta com nós primários que recebem as solicitações e escolhem, de forma sequencial ou aleatória, quais nós secundários irão replicar os dados [Greve 2005]. Essa característica pode ser observada na maioria dos sistemas de armazenamento utilizados em larga escala. Como exemplo tem-se o banco de dados distribuído Cassandra [Lakshman and Malik 2009], que propaga o dado armazenado em um nó, em N réplicas no sistema. As réplicas podem ser escolhidas de forma sequencial dentro do grupo de nós, ou serem designadas por um componente externo chamado Zookeeper [Hunt et al. 2010], que para detecção de falhas efetua uma comunicação par a par com todos os nós do sistema, e assim define a escolha. Ainda neste contexto tem-se o sistema de armazenamento Dynamo [DeCandia et al. 2007] que trabalha de forma semelhante ao sistema citado anteriormente, mas para controle do grupo de nós do sistema implementa um detector de falhas baseado em *heartbeats* através de um protocolo *gossip*, obtendo informações sobre os nós ativos no sistema. Outro exemplo é o banco de dados distribuído Bigtable [Chang et al. 2006], criado pela empresa Google para armazenar distribuidamente uma grande quantidade de dados. Internamente o sistema utiliza o protocolo Paxos para coordenar integridade das informações na presença de falhas de processo. Este protocolo tem como base um detector de falhas não confiável para identificar a situação de cada nó da rede.

Partindo deste contexto, é proposto que o detector de falhas implementado neste trabalho seja utilizado como um componente interno do sistema armazenamento que aponta diretrizes de escolha dos nós a serem utilizados na replicação. Isto se dá da seguinte maneira: a partir da chegada de uma requisição de armazenamento no nó primário, será efetuado uma consulta ao detector de falhas local que irá determinar os nós falhos que não devem ser integrados no processamento da replicação. O FD é um componente que está presente em todos os nós do sistema, fazendo com que cada processo tenha uma visão adequada dos demais. A Figura 1, abaixo, demonstra a aplicação do FD proposto na nuvem e a visão que cada nó tem do sistema.

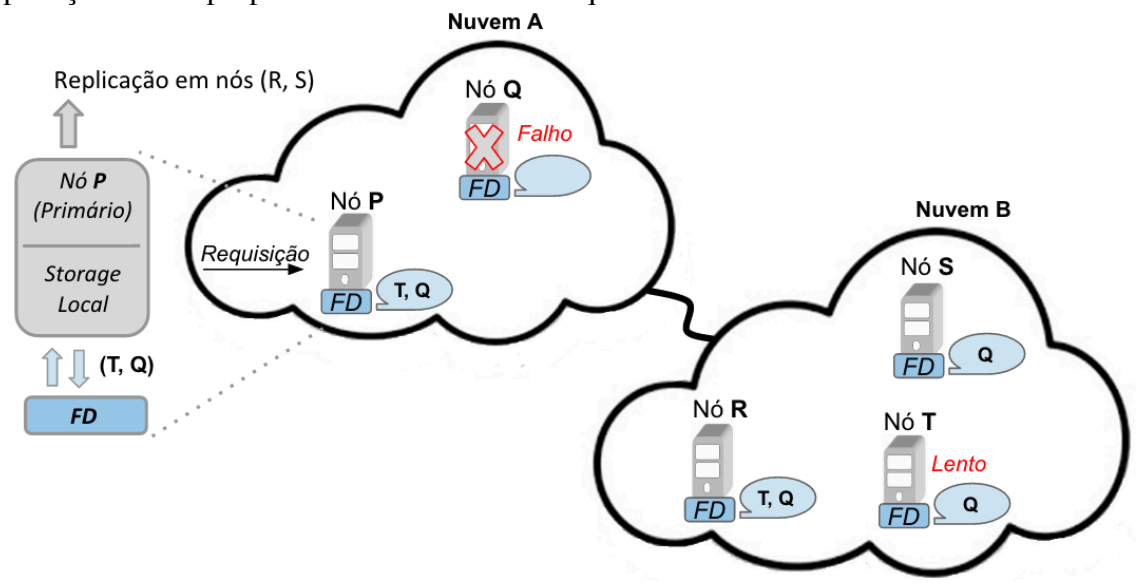


Figura 1. Aplicação do FD no componente de *storage* da nuvem

3.3. Modelo do Sistema

Considera-se um sistema distribuído dinâmico constituído por um conjunto finito de processos $\Pi = \{p1, \ldots, pn\}$, com $|\Pi| = n$, e $n > 3$, em cada execução. Este modelo expressa adequadamente as redes dinâmicas onde os nós (máquinas virtuais) entram e deixam a rede livremente [Fan et. al 2012]. Não existe um relógio global conhecido para todos os processos, mas para simplificar a apresentação, assume-se o intervalo T de c*lock* como um conjunto dos números naturais.

Na transmissão de mensagens entre processos, o protocolo de *broadcast* garante que a difusão ocorre para todos os nós do sistema, conjunto Π, ainda que este valor possa variar entre cada execução. Mesmo os nós recém chegados devem receber a mensagens enviadas por *broadcast*. Portanto, a primitiva *broadcast(m)* invocada pelo processo p, correto, envia uma cópia da mensagem m através do *link* de p para q, para cada $q \in \Pi$ [Greve 2005].

Para o ambiente de computação em nuvem proposto, adota-se a comunicação do tipo *fair-lossy*. Uma mensagem m enviada por um processo correto p um número infinito de vezes, é recebido pelo processo q um número infinito de vezes, ou q é bizantino. Como garantia da chegada de mensagens são empregados mecanismos de confirmação e retransmissão. Além disso, processos bizantinos não podem interferir na transmissão de mensagens de processos corretos, sendo vetado a duplicação e modificação de mensagens. Processos maliciosos podem tentar criar mensagens falsas, identificando-as como de outro processo. Este problema será assegurado pela autenticação das mensagens [Greve 2005].

3.4. Modelo de Falhas dos Processos

Os processos estão sujeitos a falhas bizantinas [Lamport et al. 1982], isto é, podem apresentar comportamento arbitrário desviando da execução do algoritmo especificado, podendo parar, omitir o envio e recepção de mensagens, enviar mensagens espúrias, além de trabalhar em conluio com outros processos maliciosos visando corromper o comportamento do sistema. Um processo p que não segue a especificação do algoritmo é dito bizantino, caso contrário, ele está correto, neste caso, a função *correct(p)* é verdadeira. Os conjuntos de processos corretos e falhos formam uma partição.

Na literatura, as falhas bizantinas são divididas em duas classes distintas: quando o comportamento externo de um processo fornece evidências da falha, é chamada de *detectável*, caso contrário, *não detectável* [Kihlstrom et. al. 2003]. Este trabalho lida com falhas detectáveis. Elas são classificadas em falhas de omissão (*progress*), onde o processo defeituoso não envia as mensagens exigidas pela especificação; e, falhas de segurança (*security*), que violam as propriedades invariantes que os processos devem obedecer de acordo com o algoritmo em execução.

Alguns requisitos devem ser atendidos em sistemas que admitem falhas bizantinas [Kihlstrom et. al. 2003]: *i*) processos corretos devem identificar de forma consistente as mensagens enviadas por cada processo; *ii*) processos corretos devem ser capazes de verificar a validade da mensagem de acordo com os requisitos do algoritmo especificado. Neste trabalho, o primeiro requisito foi satisfeito com a utilização de canais autenticados, e o segundo através da criptografia, como detalhado a seguir.

Canais autenticados: Assume-se o modelo de criptografia de chave pública e privada, em que cada processo *p* tem um chave privada *K,* com a qual assina suas mensagens, (a exemplo de RSA [Rivest et al. 1978]) e cada outro processo *q* no sistema pode obter a chave pública de *p*, a fim de autenticar o remetente de uma mensagem assinada. Processos bizantinos não podem subverter as primitivas criptográficas.

Validação de mensagens: O algoritmo efetua a validação do conteúdo das mensagens através do modelo de criptografia *hash* (como exemplo, MD5 [Rivest 1992]), evitando falhas arbitrárias. É computacionalmente simples obter o valor *hash* para qualquer mensagem de entrada, além de ser impossível encontrar duas mensagens distintas com o mesmo valor *hash*. Aqui, utiliza-se uma criptografia *hash*, chamado doravante de **H**, com as seguintes propriedades de segurança:

- **H** aceita um dado de tamanho arbitrário como entrada;
- **H** produz uma saída de tamanho fixo, independentemente do tamanho da entrada;
- Dado um valor *hash h*, é computacionalmente difícil de encontrar uma mensagem *M* tal que $h = M$.
- É computacionalmente impossível encontrar qualquer par de mensagens distintas (*M1, M2*) de tal modo que **H**(*M1*) = **H**(*M2*).

Limite de falhas de processos: Para garantir conectividades dos processos, deve-se delimitar um limite de tolerância de falhas bizantinas nos processos. Isto garante que as informações de um processo *p* vão ser recebidas/enviadas para um mínimo de processos corretos na rede. Este limite é de, pelo menos, $f + 1$ processos corretos que podem se comunicar com *p*. Caso o processo *p* esteja falho, depois de um período, pelo menos $f + 1$ processos corretos vão suspeitar de *p* e podem espalhar a suspeita para o restante do sistema. O conjunto de processos do sistema contendo $n > 2f$ processos é suficiente e necessário para a execução correta do protocolo de detecção de falhas bizantinas.

4. Detector de Falhas Bizantinas

Esta Seção descreve o algoritmo (*A*), proposto em [Matos and Greve 2015], que utiliza o detector de falhas como módulo interno para efetuar a identificação de falhas bizantinas, satisfazendo o modelo descrito na Seção anterior.

O algoritmo é executado constantemente, enviando por difusão uma mensagem de solicitação, Q*uery*. O intervalo de tempo entre envios consecutivos é finito e arbitrário. Quando o processo *j* recebe uma mensagem Q*uery* de um processo *p*, *j* lhe confirma a recepção com uma mensagem *Response*. As mensagens *Query* e *Response* determinam aqui o padrão de mensagens trocadas pela aplicação de armazenamento. O algoritmo é composto por três comportamentos principais, são eles:

Geração de suspeita de falha: as suspeitas geradas por falhas de omissão estão ligadas a mensagens requeridas pelo algoritmo *A*. Assim, o processo *j* é suspeito de falhar pelo processo *p*, se *j* não enviar as mensagens que deveria para *p* de acordo com *A*. Para tanto, cada mensagem deve ter um identificador distinto. Além das suspeitas locais, um processo também adota suspeitas se receber mensagens de *Suspicion* devidamente validadas de, pelo menos, $f + 1$ remetentes diferentes. Esta exigência não permite que processos bizantinos criem suspeitas sobre processos corretos.

Geração de erro de suspeita: Seja *j* um processo suspeito de não enviar uma mensagem *m* requerida por *A*. Se, após um tempo, um processo *p* recebe *m* de *j* devidamente assinada e validada, *p* irá declarar o erro de suspeita e espalhar uma mensagem *Suspicion* para que os demais processos possam fazer o mesmo.

Identificação de falha bizantina: Para assegurar a correção da detecção de falhas, deve-se estabelecer um padrão para as mensagens de *A*. Estas podem ser do tipo *Response* (resposta a uma mensagem inicial *Query*) ou *Suspicion* (mensagem de atualização de suspeitas internas). Como todas as mensagens do algoritmos são assinadas através de criptografia de chave pública e seu conteúdo é validado através de criptografia *hash*, é perfeitamente possível garantir que uma mensagem *m* esteja formatada corretamente, e que seu remente é autêntico. Dessa maneira, *m* é válida se: *i) m* possui conteúdo correto, ou seja, computado o seu valor *hash* é igual ao valor *hash* recebido; *ii) m* segue adequadamente um padrão especificado pelo algoritmo. Seguindo estes princípios, se um processo correto detecta a invalidade de uma mensagem recebida, ele irá suspeitar permanentemente do remetente, como processo bizantino, e irá espalhar uma mensagem para os processos restantes.

4.1. Descrição do Algoritmo de Detecção de Falhas Bizantinas

A seguinte notação é utilizada na implementação do algoritmo:

- *output:* guarda a saída do FD;
- *know:* conjunto de processos conhecidos pelo FD;
- *mistake:* array que guarda os processos com erro de suspeita;
- *inter_susp:* conjunto de processos suspeitos internamente;
- *exter_susp*: array que armazena as suspeitas externas (geradas pelos outros nós);
- *byzantine:* array que guarda os processos considerados bizantinos;
- *received:* conjunto de processos dos quais recebeu mensagens *Response*;
- *Broadcast(m):* transmite uma mensagem *m* para todos os processos da rede*;*
- *Send(m, j):* transmite a mensagem *m* para o processo *j*;
- $H(m)$*:* função de criptografia *Hash* que dá como saída o valor *hash* da mensagem *m*;
- $K_{Priv(p)}$*:* função de criptografia de chave privada do processo *p;*
- $K_{Pub(p)}$*:* função de criptografia de chave pública do processo *p;*

O algoritmo ainda utiliza os seguinte procedimentos auxiliares:

- *InternalSusp(j):* adiciona as suspeitas por omissão geradas pelo FD (linhas 34-36);
- *Mistake(j, m):* insere erros de suspeitas de omissão anteriores (linhas 38-44);
- *Byzantine(j, m):* adiciona as suspeitas de processos bizantinos (linhas 45-47);
- *ValidateReceived(j, m):* decodifica e valida a mensagem *m* recebida (linhas 49-59);
- *UpdateState(j, m):* atualiza o estado do FD local (linhas 61-73);
- *ValidateHash (j, m):* valida a segurança da mensagem *m* recebida (linhas 75-90).

A implementação do algoritmo *A* é composta por quatro tarefas executadas paralelamente por cada processo: a tarefa T1, realiza o envio de solicitações iniciais (linhas 5-6); a tarefa T2, realiza o envio de respostas (linhas 8-10); a tarefa T3 é responsável pela geração inicial de suspeitas de falhas (linhas 12-24); e por fim a tarefa T4 efetua o recebimento de mensagens para a atualização do estado do detector (linhas 26-31).

As provas de correção deste algoritmo, bem como suas especificações de acordo

com a classificação de detectores de falhas não confiáveis proposta Chandra e Toueg (1996), Kihlstrom et al. (2003) podem ser obtidas em Matos and Greve (2015).

Algoritmo *A* – Detector de Falhas Bizantinas Assíncrono

```
1: Init:
2: output ←known ←inter_susp ←∅
3: exter_susp ←mistake ←byzantine ←[ ]
4:
5: Task T1:  /* Solicitação inicial - Query */
6:  Broadcast(Q)
7:
8: Task T2:  /* Envio de Response para j */
9:  CR = K_Priv(p){H(Response) + Response}
10:  Send(CR, j)
11:
12: Task T3:  /* Gera novas suspeitas */
13:  when (requires messages Response) do
14:  wait until receive m from α distinct
     processes j
15:    received ←j
16:  for all j ∈ ( known \ received ) do
17:     InternalSusp(j)
18:  end for
19:  for all m received from j do
20:     ValidateReceived(j, m)
21:  end for
22:  S ←SUSPICION(byzantine, mistake,
     inter_susp)
23:  CS ←K_Priv(p){ H(S) + S }
24:  Broadcast(CS)
25:
26: Task T4:  /* Recebe mensagens Suspicion
     ou Response de processos lentos*/
27:  upon receipt of m from j do
28:     ValidateReceived(j, m)
29:  S ←SUSPICION(byzantine, mistake,
     inter_susp)
30:  CS ←K_Priv(p){ H(S) + S }
31:  Broadcast(CS)
32:
33: /* PROCEDIMENTO AUXILIARES */
34: procedure InternalSusp( j ):
35:  inter_susp ←inter_susp ∪ { j }
36:  output ←output ∪ { j }
37:
38: procedure Mistake( j, m ):
39:  mistake[ j ]  ←mistake[ j ] ∪ { m }
40:  inter_susp ←inter_susp \ { j }
41:  exter_susp ←exter_susp[ ] \ { j }
42:  if  byzantine[ j ] = ∅ then
43:     output ←output \ { j }
44:  end if
45: procedure Byzantine( j, m ):
46:  byzantine[ j ] ←byzantine[ j ] ∪ { m }
47:  output ←output ∪ { j }
48:
49: procedure ValidateReceived( j, m ):
50:  mr ←ValidateHash( j, m )
51:  if mr != ∅ then
52:     known ←known ∪ { j }
53:     if j ∈ inter_susp then
54:        Mistake( j, m )
55:     end if
56:     if mr is SUSPICION then
57:        UpdateState( j, mr)
58:     end if
59:  end if
60:
61: procedure UpdateState( j, m ):
62:  for all ( j_x, m_x ) ∈ m.byzantine do
63:     ValidateReceived( j_x, m_x )
64:  end for
65:  for all ( j_x, m_x ) ∈ m.mistake do
66:     ValidateReceived( j_x, m_x )
67:  end for
68:  for all ( j_x ) ∈ m.inter_susp do
69:     exter_susp[ j ] ←exter_susp[ j ] ∪ { j_x }
70:     if l ( j_x ) ∈ exter_susp[ ] l > f then
71:        InternalSusp(j_x)
72:    end if
73:  end for
74:
75: function ValidateHash( j, m):
76:  md ←K_Pub(j) { m }
77:  if md is [ Hmsg, msg ] then  /* remetente */
78:     Hmc ←H( md.msg )
79:     if Hmc != md.Hmsg  then  /* conteúdo */
80:        Byzantine( j, m )
81:        return ∅
82:     else
83:        if (md.msg = Response) or  /* padrão */
             (md.msg = Suspicion) then
84:           return md.msg
85:        else
86:           Byzantine( j, m)
87:           return ∅
88:        end if
89:     end if
90:  end if
```

5. Implementação e Avaliação de Desempenho do Detector de Falhas

Nessa Seção é apresentado um estudo de avaliação de desempenho da implementação do detector de falhas assíncrono proposto. Os resultados obtidos são comparados com os apresentados pelo detector existente no Zookeeper [Hunt et al. 2010]. O intuito dessa análise comparativa é fazer um contraponto entre o detector de falhas assíncrono criado e um detector baseado em temporização largamente utilizado em aplicações reais existentes. Em seguida é avaliado o detector de falhas na presença de nós bizantinos.

5.1. Modelo de Simulação

Esta simulação busca avaliar a qualidade de serviço dos detectores de falhas através de algumas métricas baseadas em tempo sugeridas por Chen and Toueg (1996). No ambiente de *storage* em nuvem proposto, o detector de falhas deve estar pronto a responder às solicitações das réplicas primárias e informar os nós falhos da rede não indicados para replicação. Assim, quanto mais rápido e correto forem identificados os nós falhos, melhores serão as respostas emitidas pelo detector, consequentemente ocasionando um serviço de replicação mais seguro.

Vale salientar que o ambiente simulado foi escolhido para eliminar da avaliação as possíveis interferências dos ambientes reais, com o proposito de obter resultados mais precisos sobre o modelo avaliado. Todos os experimentos foram realizados em um computador Intel Core i5 3.2GHz com 8GB de RAM e sistema operacional Ubuntu 14.04 64bits. Os detectores de falhas foram implementados na linguagem Java utilizando o pacote de *sockets* para a troca de mensagens no algoritmo assíncrono proposto, e o pacote Apache Zookeeper[2] versão 3.4.9 na implementação do Zookeeper. Cada cenário do teste de desempenho foi repetido por 10 vezes obtendo uma média dos dados das execuções como resultado final. Neste contexto, cada processo equivale a um nó da rede.

Detector de Falhas Zookeeper: Este detector é baseado numa conexão cliente servidor, onde é armazenada em cada servidor uma variável de conexão de cada nó cliente chamada de zNode. Assim, cada servidor periodicamente envia uma mensagem de *heartbeat* a seus clientes verificando a atividade destes nós. Todos os nós implementam o comportamento de servidor para obter informações sobre os nós da rede, e também de cliente para responder a solicitações dos outros nós. Este detector trabalha unicamente com falhas benignas. Para questões de implementação de detecção de falhas foi utilizado modo *standalone* com zNodes tipo *ephemeral*.

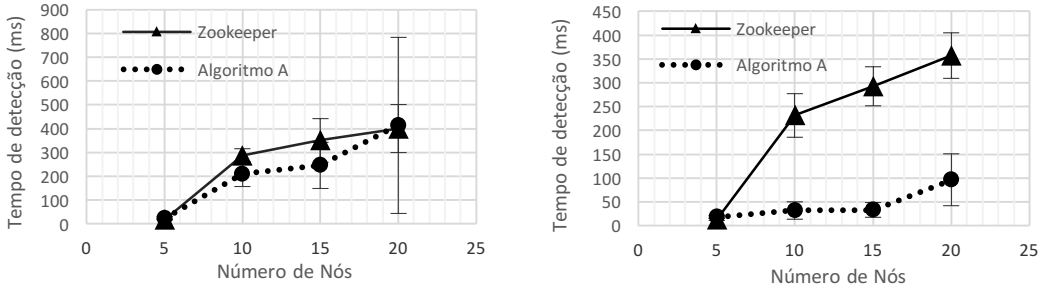
Detector de Falhas Assíncrono: Na implementação deste detector foi utilizado como tempo de reenvio de mensagens tipo *Query*, executado pela tarefa **T1**, o tempo de sessão padrão do Zookeeper, *TickTime* com 3 segundos, permitindo que os dois detectores trabalhem com os mesmos parâmetros. Com o intuito de evitar sobrecarga na rede, ao receber uma mensagem tipo *Suspicion,* a mensagem a ser repassada em *broadcast* não é reenviada ao remetente. Também é implementado número de sequência de mensagens para validar a sua atualidade. Vale ressaltar que essas otimizações não interferem na correção do algoritmo.

[2] Disponível em http://zookeeper.apache.org

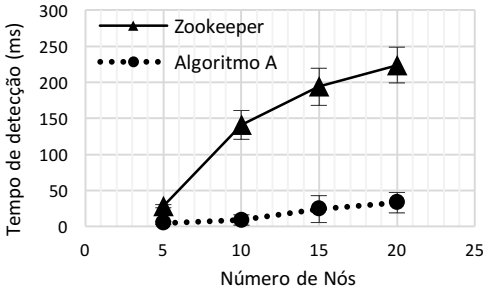
5.2. Avaliação da Detecção de Falhas Benignas

Para avaliar a qualidade de serviço de ambos os detectores, foi mensurado o impacto da variação da densidade de nós na rede e os respectivos tempos de detecção de falha dos dois detectores (Figura 2). O tempo de detecção de falha consiste no intervalo entre o instante *t* em que a falha aconteceu, e o instante t' > t em que ela é permanentemente detectada por todos os processos corretos. Para a avaliação foram criados três cenários com diferentes números de nós falhos: *i*) rede sem nós falhos, *ii)* rede com *f*/2 nós falhos, e *iii)* rede com *f* nós falhos, sendo *f* o limite de tolerância de falhas de acordo com a quantidade de nós da rede N, $f = (N/2) - 1$. As falhas são inseridas em nós aleatórios, sendo que em cada execução existe um numero fixo de nós falhos.

No cenário com apenas nós corretos, o tempo de detecção foi considerado como o momento em que todos os nós reconhecem a não existência de falhas na rede.



(a) Sem nós falhos (b) *f/2 nós falhos*



(c) *f* nós falhos

Figura 2. Tempo de detecção de falhas benignas

De acordo com a Figura 2 é possível perceber que quando não existe falha na rede, os dois detectores se comportam de maneira semelhante, com o crescimento do tempo de detecção associado à densidade de nós da rede. Já no momento que são inseridos nós falhos, (b) e (c), o algoritmo proposto consegue imprimir uma redução considerável no tempo de detecção, diferenciando-se do detector Zookeeper. Isto se justifica porque o detector de falhas assíncrono utiliza o princípio de suspeitar de todos os processos que não estiverem entre as primeiras α respostas recebidas (α corresponde a quantidade mínima de processos corretos). Assim independente de um tempo de espera de respostas, estes processos já são tidos como falhos para o detector. Erros de suspeitas para processos com respostas atrasadas são tratados posteriormente com a chegada de cada mensagem. Outro fator que influi para o baixo tempo de detecção são as mensagens do tipo *Suspicion,* que espalham na rede as informações sobre os

processos falhos descoberto por cada nó. Estas mensagens proporcionam, além de uma melhoria no tempo detecção, uma garantia para que toda a rede suspeite dos nós falhos.

5.3. Avaliação da Recuperação de Falsas Suspeitas

Como o detector proposto permite gerar falsas suspeitas sobre nós da rede e efetua correções posteriores, faz-se necessário validar o impacto deste mecanismo na detecção e falhas. Nesta avaliação foi verificado o tempo de recuperação de falsas suspeitas, sendo ele o período entre o tempo *t,* onde o detector suspeita erroneamente de um processo, e o tempo t' > t, onde esta suspeita é eliminada. Utilizou-se os mesmos três cenários de falhas da avaliação anterior.

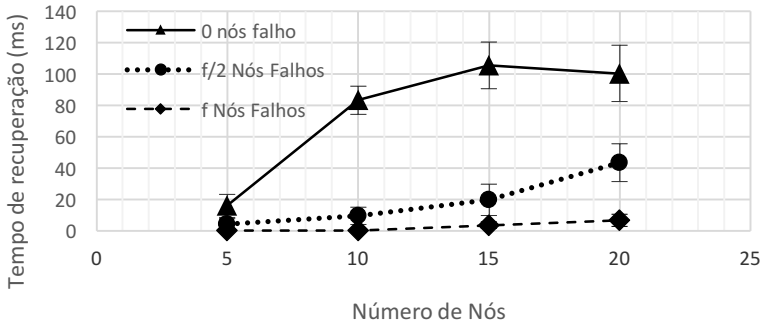


Figura 3. Tempo de recuperação a falsas suspeitas do detector proposto

A Figura 3 mostra que na rede sem nós falhos o tempo de recuperação de falsas suspeitas é consideravelmente maior, pois são geradas várias suspeitas de nós corretos mais lentos, que são corrigidas posteriormente. Entretanto nos cenários com nós falhos, a recuperação é efetuada de forma rápida, seja pela chegada das respostas dos nós suspeitos, reavaliando o processo como ativo, ou pela chegada de mensagens do tipo *Suspicion,* que confirmam a situação de falhas nos processos da rede. Enfim, nos três casos avaliados, o tempo de recuperação de falhas não é fator limitante para a qualidade da detecção de falhas, pois equivale apenas cerca de 30% do tempo total de detecção, apresentado na Figura 2.

Destaca-se ainda que mesmo não trabalhando com *timeouts* de conexão, o detector proposto nunca irá suspeitar de um grupo de processos com comunicação mais rápida na rede, sempre que suas mensagens estiverem dentro das primeiras α respostas recebidas.

5.4. Avaliação da Detecção de Falhas Bizantinas

Nesta avaliação os processos bizantinos agem de forma randômica entre três comportamentos: como um processo correto, emitindo mensagens corretas; como um processo falho, com omissão de mensagens; ou de forma arbitrária, se desviando do padrão de mensagens exigida pelo algoritmo. O detector de falhas proposto é capaz de identificar os três comportamentos do processo, entretanto é garantido que um processo uma vez identificado como bizantino, sempre seguirá com esta classificação, mesmo que este volte a se comportar como um processo correto.

Para esta avaliação do detector, foi mensurado o impacto no tempo de detecção dos nós bizantinos com a variação da densidade da rede (Figura 4). O tempo de

detecção bizantina consiste no intervalo entre o instante *t* em que um nó bizantino emitiu sua primeira mensagem bizantina, e o instante t' > t em que este nó é permanentemente detectado por todos os processos corretos. Na avalição foi utilizado apenas os dois cenários com falhas, pois a rede com ausência de falhas já foi avaliada anteriormente.

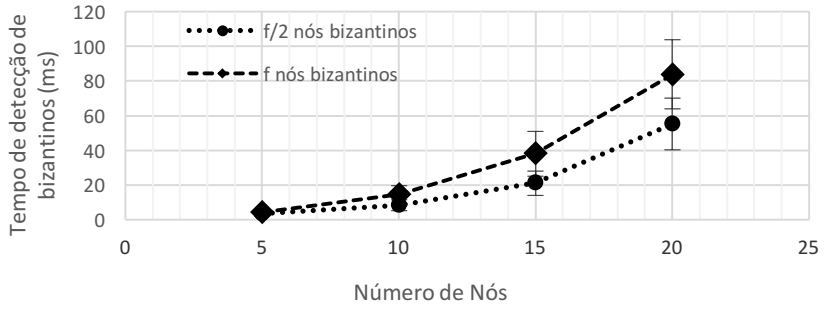


Figura 4. Tempo de detecção de falhas bizantinas

A partir da Figura 4 é possível perceber que após emissão de uma mensagem bizantina, a identificação e propagação desta informação pela rede é feita de forma rápida pelas mensagens tipo *Suspicion.* Salienta-se que para uma informação ser tida como verdadeira, deve ser recebida de pelo menos $f+1$ processos, para que a união de nós bizantinos não possa resultar numa subversão da rede.

Fazendo contraponto entre as Figuras 2 e 4, avalia-se que a detecção de falhas bizantinas acontece, em alguns casos, em menor tempo do que detecção de falhas benignas. Isto se justifica porque a suspeita de falhas bizantinas pode ocorrer na primeira mensagem recebida, desde que essa seja emitida por um nó bizantino, já as suspeitas de falhas benignas seguem o padrão do algoritmo de surgirem após α respostas recebidas.

6. Conclusões e Trabalhos Futuros

Neste artigo é apresentado uma proposta de armazenamento seguro na nuvem utilizando como coordenador de replicação um detector de falhas bizantinas assíncrono. O detector traz as seguintes características inovadoras aplicadas à computação em nuvem: *i)* adequado a redes dinâmicas, de forma a promover a escalabilidade e capacidade de adaptação; *ii)* trata o sistema com assíncrono, retirando as primitivas temporais na detecção de falhas, modelando a realidade de nuvens federadas e *Mashups*. Os resultados da avalição de desempenho puderam validar a eficácia do detector proposto. Por fim, como continuação deste trabalho pretende-se ampliar o protocolo para realidade de outros módulos da computação em nuvem.

Referências

Chandra T. and Toueg, S. (1996) "Unreliable failure detectors for reliable distributed systems". J Journal of ACM. *43*, pp . 225–267.

Chang, F., Dean, J., Ghemawat, S. et. Al. (2006) "Bigtable: a distribuited storage system for structured data". In Porc of 7h Conf. USENIX. Berkeley, CA.

DeCandia G., Hastorun D., Jampani M., et. Al. (2007) “Dynamo: amazon’s highly available key-value store”. *In Proccessing of 21th ACM SIGOPS symposium on Operating systems principles*. pp. 205–220..

Fan G., Yu H., Chen L. and Liu D. (2012) “Model Based Byzantine Fault Detection Technique for Cloud Computing” In *IEEE Services Computing Conference (APSCC)*, Guilin, Asia-Pacific, pp. 249-256.

Ganesh, A., Sandhya, M. and Shankar, S. (2014) “A study on fault tolerance methods in Cloud Computing”, In *Advance Computing Conference (IACC), IEEE,* pp. 844 - 849

Garraghan P., Tounend P. and Jie X. (2011) “Byzantine fault-tolerance in federated cloud computing”. In *Proccessing of 6th International Symposium on Service Oriented System Engineering (SOSE 2011). IEEE Computer Society*, pp. 280-285.

Greve, F. (2005) “Protocolos Fundamentais para o desenvolvimento de Aplicações Robustas” SBC-SBRC, Brasil, pp. 330-398.

Greve, F., Lima M., Arantes L. and Sens P. (2012) “A Time-Free Byzantine Failure Detector for Dynamic Networks” In *IEEE Dependable Computing Conference (EDCC),* Sibiu, Ninth European. pp.191 – 202.

Hunt, P., Konar, M., Junqueira, F. and Reed B., (2010) “Zookeeper: Wait-free coordination for internet-scale systems In *Proc. 10th USENIX Annual Technical Conference (USENIXATC)*, Boston, USA.

Kihlstrom K. P., Moser L. E. and Melliar-Smith P. M. (2003) “Byzantine Fault Detectors for Solving Consensus,” The Computer Journal, vol. 46, no. 1, pp. 16–35

Lakshman A. and Malik P. (2009) “Cassandra – A Decentralized Structured Storage System”

Lamport L., Shostak R. and Pease M., (1982) “The Byzantine generals problem” ACM Transactions on Programming Languages and Systems, vol. 4, pp. 382–401.

Matos C. and Greve F., (2015) “Um Detector de Falhas Bizantinas Assíncrono Aplicada à Computação em Nuvem” XVI WTF-SBRC, Vitória, ES, Brasil. SBC.

Ramasamy H. and Cachin. C. (2005) “Parsimonious Asynchronous Byzantine-Fault-Tolerant Atomic Broadcast” In *Proc. 9th International Conference on Principles of Distributed Systems*, Berlin. Germany

Rivest R., Shamir A. and Adleman L, (1978) “A method for obtaining digital signatures and public-key cryptosystems,” Commun. ACM, vol. 21, pp. 120–126.

Rivest R. (1992), “The MD5 Message-DigestAlgorithm”. MIT Laboratory for Computer Science and RSA DataSecurity.

Sivakami, R. and Nawaz G. (2011) “Reliable communication for MANETS in military through identification and removal of byzantine faults” In *IEEE 3rd Inter Conference on Electronics Computer Technology (ICECT)*. Kanyakumari, Indian, pp. 377-381.

Verissimo, P., Neves, N. F., et. al. (2003) “Intrusion-Tolerant Architectures: Concepts and Design”, Lemos, R., Gacek, C., Romanovsky, A. (eds), Architecting Dependable Systems, v. 2677, LNCS, Springer-Verlag.

XVIII Workshop de Testes e Tolerância a Falhas

SBRC 2017
Sessão Técnica 2
Técnicas de Tolerância a Falhas

Replicação de Dados no VCube Baseada em DHT de Salto Único

Alexandre B. Neto[1], **Luiz A. Rodrigues**[1] **e Elias P. Duarte Jr.**[2]

[1] Ciência da Computação – Universidade Estadual do Oeste do Paraná (UNIOESTE)
Cascavel – PR – Brasil

[2]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

alexandrebn01@gmail.com, luiz.rodrigues@unioeste.br, elias@inf.ufpr.br

Abstract. *This paper presents a strategy to replicate files on a VCube, a virtual topology based on a hypercube that presents several logarithmic properties. The proposed strategy is based on a single-hop Distributed Hash Table (DHT), a data structure that provides an efficient and scalable solution to locate information in a P2P (peer-to-peer) network. The system uses a replication technique that fragments file data across system participants in order to improve availability. Simulation results confirm the efficiency of the proposed solution in scenarios with and without faults.*

Resumo. *Este trabalho investiga a replicação de dados utilizando o VCube, uma topologia virtual baseada em hipercubo com diversas propriedades logarítmicas. A estratégia proposta é baseada em tabela hash distribuída (DHT* - Distributed Hash Table*) de salto único, estrutura de dados que oferece uma solução eficiente e escalável para a localização de informações em redes peer-to-peer. O sistema proposto utiliza uma técnica de replicação que fragmenta os arquivos entre os participantes do sistema, buscando garantir a disponibilidade dos dados. Resultados de simulação confirmam a eficiência da solução proposta em cenários com e sem falhas de nodos.*

1. Introdução

Atualmente uma variedade de organizações necessitam distribuir informações e proporcionam serviços para milhares de usuários. A necessidade de desenvolver tecnologias que garantam a escalabilidade dos serviços cresce constantemente. Para um sistema ser escalável, deseja-se que este não possua dependências de pontos centralizados e que seja auto-organizável, prevendo a alteração dinâmica da sua composição em tempo de execução. Outro ponto importante é a disponibilidade, especialmente levando em conta que a medida que o número de componentes aumenta, maior é a probabilidade da ocorrência de falhas.

Uma estrutura de dados que possui grande parte das propriedades citadas é a tabela hash distribuída (DHT - *Distributed Hash Table*). Esta estruturas são comuns em redes *peer-to-peer* (P2P), que são sistemas distribuídos que possuem a característica de se auto-organizar, com o objetivo de realizar o compartilhamento de recursos. As redes P2P surgiram originalmente como sendo uma alternativa ao modelo cliente/servidor. Em um

modelo de sistema P2P puro não há a noção de que um par (*peer*) seja apenas cliente ou servidor, um *peer* é, ao mesmo tempo, cliente e servidor. Além disso, uma rede P2P pode ser classificada em dois tipos, não-estruturada e estruturada. Na primeira, a disposição dos arquivos não possui vínculo com a topologia rede, na segunda, os arquivos são alocados em posições específicas da rede e não de forma aleatória. Desse modo, as buscas são realizadas de forma eficiente e escalável [Androutsellis-Theotokis e Spinellis 2004].

As DHTs são estruturas de dados que armazenam informações distribuídas sobre múltiplos *peers* do sistema. Para o armazenamento, elas utilizam o conceito de chave/valor, mais conhecido por sua utilização em tabelas *hash* tradicionais. Além disso, usam uma estrutura chamada tabela de roteamento para encaminhar uma consulta a partir do *peer* solicitante até o responsável pela chave procurada. Cada participante do sistema possui uma tabela. Ela é composta por um subconjunto de entradas que referenciam outros pares, associando um par à sua chave e a seu respectivo endereço na rede [Ghodsi 2006].

Algumas implementações utilizam tabelas de roteamento completas, onde cada *peer* conhece todos os demais participantes do sistema. As DHTs que usam essa abordagem são definidas como sistemas de salto único, pois dependem de um único salto para a consulta. Em outras implementações, cada *peer* faz uso de tabelas de roteamento parciais e necessitam que uma consulta seja roteada por diversos *peers* do sistema. As DHTs que fazem uso dessa abordagem são definidas como sistemas de múltiplos saltos.

Neste trabalho é apresentada uma estratégia de replicação de dados em uma topologia virtual baseada em hipercubo conhecida como VCube [Duarte et al. 2014] para uma solução de DHT de salto único denominada VCube-DHT, também proposta neste trabalho. O VCube possui diversas propriedades logarítmicas, escaláveis por definição. A estratégia de replicação proposta é baseada na fragmentação dos dados dos arquivos mantidos entre os participantes do sistema. Esta é uma diferença importante da técnica proposta para diversas outras existentes que fazem replicação total, isto é, os dados são replicados por completo entre os participantes da rede. Entre tais estratégias está a HyperDHT [Koppe et al. 2014], que também é uma DHT baseada em hipercubo virtual, mas prevê replicação total, além de não ser baseada no VCube. Este trabalho apresenta dados experimentais de comparação da estratégia proposta com a HyperDHT.

O restante do texto está organizado da seguinte forma. A Seção 2 apresenta uma abordagem de DHTs de único salto. A Seção 3 exibe as principais características referentes ao modelo do sistema. A Seção 4 aborda as principais características do VCubeDHT. A Seção 5 apresenta os resultados obtidos em relação as simulações realizadas. Por fim, a Seção 6 exibe as principais conclusões sobre o trabalho elaborado, bem como sugestões de trabalhos futuros a serem desenvolvidos.

2. Trabalhos Relacionados

Enquanto algumas DHTs mantêm tabelas de roteamento com informações sobre todos os nodos do sistema, outras optam pela utilização de tabelas parciais que são distribuídas entre os nodos da rede. As soluções que usam essa abordagem visam minimizar o tráfego de manutenção das tabelas de roteamento, com prejuízo para a latência das consultas.

CAN (*Content-Addressable Network*) [Ratnasamy et al. 2001] é um sistema completamente distribuído que não necessita de controle centralizado. Este possui uma estru-

tura baseada em um espaço cartesiano virtual multi-dimensional de coordenadas, sendo completamente lógico, não possuindo qualquer relação com a estrutura física do sistema. Além disso, este espaço é particionado de forma dinâmica entre os participantes do sistema. Estes mantêm informações acerca de um pequeno número de nodos adjacentes, armazenando o endereço IP e as coordenadas desses. O número de saltos necessários em uma consulta geralmente será maior que naqueles sistemas no qual os participantes possuem tabela de roteamento completa.

Chord [Ratnasamy et al. 2001] é um modelo de sistema que proporciona rápido mapeamento de chaves entre os nodos do sistema utilizando uma variante de *hashing* consistente. O identificador do nodo é selecionado através do *hash* de seu endereço IP, e o identificador da chave é escolhido baseado no *hash* da mesma. O uso de uma função *hash* distribuída espalha as chaves sobre os nodos de maneira ordenada e balanceada em um anel, conhecido como *Chord Ring* de tamanho 2^m, sendo *m* o número de bits dos identificadores. Além disso, mantém o equilíbrio do sistema, resolvendo os problemas relativos ao balanceamento de carga. O Chord é considerado de múltiplos saltos pois cada nodo possui uma tabela de roteamento parcial, denominada *finger table*, usada sobretudo, para reduzir a latência das consultas.

O HyperDHT [Koppe et al. 2014] propõe uma solução na qual o procedimento de localização de uma informação ou um objeto distribuído na rede seja realizado de forma rápida, eficiente e escalável. Esse sistema utiliza uma infra-estrutura baseada em um hipercubo virtual disponibilizada pelo algoritmo de diagnóstico distribuído DiVHA[Bona et al. 2006], que provém serviços na qual caracterizam-na como sistema de salto único. O sistema implementa mecanismos para o posicionamento e busca dos objetos na rede P2P, em que associa chaves aos nodos e objetos da rede. Além disso, realiza o particionamento e mapeamento das chaves da tabela *hash* entre os nodos utilizando técnicas de *hash* consistente. O sistema também implementa protocolos de entrada de novos integrantes da rede, que determina, de acordo com a topologia atual da rede, uma posição para inserção do novo nodo onde ele é mais necessário. Por fim, o HyperDHT implementa funções para replicar as informações ou objetos dispostos na rede.

3. Modelo do Sistema

Este trabalho considera um sistema distribuído como um conjunto finito Π com $n > 1$ processos independentes, onde $\Pi = \{p_0, p_1, ..., p_{n-1}\}$. Estes se comunicam e coordenam suas ações através de trocas de mensagens [Raynal 2013]. O modelo do presente sistema é associado a um grafo cuja topologia formada é baseada em um hipercubo virtual. Cada vértice representa uma posição na rede de sobreposição que pode ou não ser ocupada. Estes vértices estão estabelecidos em *clusters* e o número de vértices pertencentes a um *cluster* é definido de acordo com a dimensão *d* do hipercubo e determinado por 2^d.

Assim como no HyperDHT, o sistema proposto permite que os *peers* possam tanto sair como entrar da rede. Sendo assim, um vértice pode ser considerado *ocupado* se ele contém um *peer*, caso contrário é classificado como *vazio*. Assim como os vértices, os participantes da rede também podem assumir dois estados, sendo eles *disponíveis* ou *indisponíveis*. Um *peer* considerado disponível é aquele que realiza suas ações de forma esperada, respondendo corretamente às requisições dos demais participantes feitas a ele. De modo contrário, um *peer* indisponível é aquele que, por algum motivo, se comporta

de maneira inesperada, não possuindo comunicação com os demais participantes da rede.

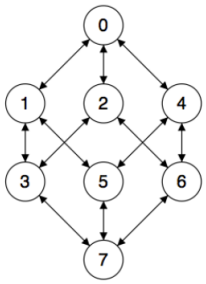
Os *peers* notificam-se uns aos outros sobre a ocorrência de um *evento* na rede por meio de trocas de mensagens. Um evento é uma mudança de estado, seja de um vértice ou de um *peer*. O processo que verifica a ocorrência de um evento é conhecido como teste. Os testes são realizados em rodadas por meio de mensagens trocadas entre os *peers* adjacentes de forma a classificá-los como disponíveis ou indisponíveis. A fim de realizar o diagnóstico do sistema, bem como a construção da rede de sobreposição, este trabalho utiliza o algoritmo de diagnóstico distribuído VCube [Ruoso 2013].

3.1. O Algoritmo VCube

O VCube é uma solução de diagnóstico distribuído que constrói e mantém os processos do sistema com base em uma topologia virtual baseada em um hipercubo. O hipercubo virtual é criado e mantido de acordo com as informações de diagnósticos obtidas por meio de um sistema de monitoramento de processos, descrito em [Duarte et al. 2014]. A cada execução do VCube, cada processo é capaz de testar outros processos no sistema para verificar se estão corretos ou falhos. Um processo é correto se a resposta ao teste realizado for recebida corretamente dentro de um intervalo de tempo.

Os processos são estabelecidos em *clusters*. Em cada rodada de testes um processo *i* testa o primeiro processo sem falha *j* na lista de processos de cada *cluster s* e obtém informação sobre os processos naquele *cluster*. Os membros de cada *cluster s*, bem como a ordem em que eles serão testados por um processo *i* são dadas pela função $C_{i,s}$. Ela determina que o processo *i* testa o processo $j \in C_{i,s}$, somente se o processo *i* é o primeiro correto em $C_{j,s}$. Logo, todo processo é testado uma única vez em cada rodada, garantindo uma latência de diagnóstico de $log_2 N$ rodadas no caso médio e $\log_2^2 N$ no pior caso.

A Figura 1 ilustra a organização lógica de um hipercubo de três dimensões. Em $C_{(0,s)}$, o processo 0 testa o processo 1 quando *s* é 1, depois testa o processo 2 quando *s* é 2, e por fim, quando *s* é 3, testa o processo 4 e é notificado com informações atualizadas sobre o estado dos demais processos daquele *cluster*.



s	$c_{0,s}$	$c_{1,s}$	$c_{2,s}$	$c_{3,s}$	$c_{4,s}$	$c_{5,s}$	$c_{6,s}$	$c_{7,s}$
1	1	0	3	2	5	4	7	6
2	2 3	3 2	0 1	1 0	6 7	7 6	4 5	5 4
3	4 5 6 7	5 4 7 6	6 7 4 5	7 6 5 4	0 1 2 3	1 0 3 2	2 3 0 1	3 2 1 0

Figura 1. Organização Hierárquica do VCube de $d = 3$ dimensões.

4. VCubeDHT

O sistema utilizado como base para a elaboração do VCubeDHT foi o HyperDHT [Koppe et al. 2014], descrito na Seção 2. A principal motivação para elaboração deste sistema foi a implementação de uma estratégia de replicação de dados mais eficiente que a usada no HyperDHT. Além disso, uma nova solução de diagnóstico foi utilizada, o VCube [Duarte et al. 2014].

As principais estratégias usadas no HyperDHT, como mecanismos para posicionamento e busca de objetos na rede, particionamento e distribuição consistente das chaves e também protocolos de entrada e saída, foram mantidas na elaboração do VCubeDHT. Para garantir a disponibilidade dos arquivos armazenados, foi implementada, no sistema proposto, uma abordagem de replicação ativa, explorando uma nova alternativa para propagação das réplicas entre os *peers* do sistema.

4.1. Determinação dos Vizinhos

A fim de determinar os vizinhos que irão receber as réplicas de um *peer*, a presente proposta utiliza a mesma estratégia adotada pelo HyperDHT. Ela define que cada participante do sistema realiza um cálculo baseado em três parâmetros, sendo eles, um identificador i do vértice no qual deseja-se verificar seus vizinhos, a dimensão d na qual determina que os vértices a serem testados são aqueles pertencentes ao *cluster* d, e um contador z, que representa a distância mínima entre dois vértices. O cálculo é feito com o uso da operação de $\oplus$ (ou exclusivo) entre o identificador i e o contador z, identificando os vértices mais próximos daquele representado pelo identificador i. A função utilizada é definida em [Koppe et al. 2014] e representada por: $C_{i,d} = i \oplus z,\ \forall z \in \mathbb{Z}_{\geq 0} : z \leq 2^d - 1$.

Na tabela da Figura 1 são exibidos os resultados após o término da execução da função $C_{i,d}$ feita por cada *peer* em um sistema de dimensão $d = 3$. Nela é possível notar, por exemplo, que o método identifica o primeiro vértice mais próximo ao vértice 0 sendo o vértice 1, o segundo mais próximo sendo o vértice 2, e o terceiro, sendo o vértice 3, e assim sucessivamente. No entanto, se o vértice analisado não estiver ocupado, ele não poderá fazer parte do grupo de replicação, dessa forma, o próximo da lista é examinado.

4.2. Replicação de Dados no VCubeDHT

O VCubeDHT é uma proposta de propagação de dados que replica os fragmentos em nível de *byte*. O pseudo-código responsável pelo processo de propagação das réplicas é apresentado no Algoritmo 1. A função MAKE_REPLICAS recebe como entrada dois parâmetros. O primeiro, denotado por k, caracteriza o fator de replicação escolhido, isto é, o número de vizinhos nos quais os dados serão replicados. Já o segundo, identificado como $frag$, é o fator de fragmentação, que define a quantidade de blocos nos quais cada arquivo será divido.

A replicação faz uso das informações de estado de cada *peer* (ocupado ou não-ocupado), armazenadas em $vertices$, e da lista de arquivos mantidos no *peer* i que está executando a replicação, referenciada por $files$. Na linha 3 os blocos de bytes de cada arquivo que devem ser enviados para cada um dos k *peers* vizinhos de i é calculada pela função CONFIGUREBLOCKS e armazenado em $blocks_k$. O cálculo consiste na criação de uma tabela verdade de tamanho 2^{frag} e na seleção das linhas com $frag - 1$ valores 1.

Em seguida, caso o vértice j sendo examinado esteja ocupado (linha 7), o *peer* responsável é considerado como pertencente à cadeia de replicação. Cada arquivo será fragmentado em $frag$ fragmentos. Após isso, os blocos serão enviados para os *peers* identificados como pertencentes à cadeia de replicação de i (linha 8).

A Figura 2 apresenta a execução do método de replicação feita pelo *peer* 0, cujos fatores de replicação e fragmentação são iguais a 3. Nesse exemplo, o arquivo ***R.txt*** será fragmentado em 3 blocos: BC, AC e AB. Assim, o vizinho mais próximo do *peer* 0 irá

Algoritmo 1 Replicação do arquivos do *peer* i (p_i)

Entrada:

$files$ ▷ Conjunto dos arquivos alocados ao *peer* i

$vertices[0, .., n-1]$ ▷ Informação sobre os vértices ocupados e não-ocupados

1: **procedure** MAKE_REPLICAS(k, $frag$)
2: $z = 0$
3: $\forall b = 1, .., k : blocks_b \leftarrow$CONFIGUREBLOCKS($b$, k, $frag$)
4: **while** $k > 0$ **and** $z < n$ **do** ▷ Envia os respectivos blocos para as k réplicas
5: $z++$
6: $j = i \oplus z$
7: **if** $vertices[j]$ **is** $occupied$ **then**
8: SEND($blocks_k$) **to** p_j
9: $k--$

10: **function** CONFIGUREBLOCKS(b, k, $frag$)
11: $blocks_b \leftarrow \emptyset$
12: **for all** $file \in files$ **do**
13: $blocks_b \leftarrow$ os fragmentos do arquivo para a réplica b com base em k e $frag$
14: **return** $blocks_b$

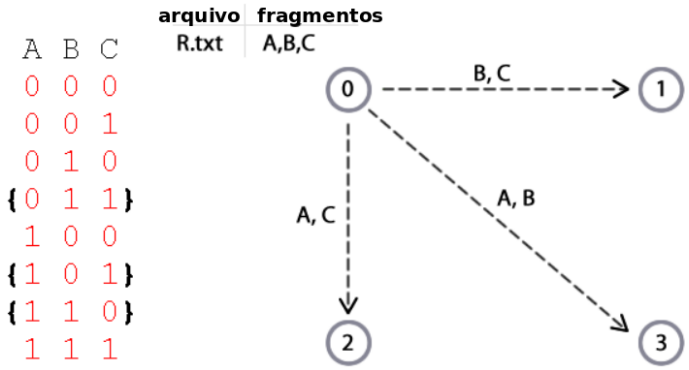


Figura 2. Exemplo de replicação a partir do método MAKE_REPLICAS**.**

receber os blocos B e C, o segundo mais próximo receberá os blocos A e C e, por fim, o terceiro mais próximo receberá os blocos A e B.

4.3. Balanceamento de Réplicas

Com o intuito de realizar o diagnóstico dos participantes da rede, após um período de tempo que pode ser ajustado pela necessidade da aplicação, o VCubeDHT faz com que todos os *peers* executem o algoritmo de diagnóstico distribuído VCube. Se após o diagnóstico for identificado um *peer* falho, além de identificar o novo *peer* responsável pelo vértice deixado pelo *peer* falho, há a necessidade da redistribuição dos blocos de dados entre os *peers* que possuem o *peer* falho como pertencente a seu grupo de replicação.

Após o término do diagnóstico, cada participante da rede determina, através de seu vetor de *timestamp* (que indica o seu conhecimento sobre o estado de cada *peer* do

sistema), a ocorrência de um *peer* falho. Depois dessa identificação, é verificado se o *peer* que está executando o algoritmo faz parte da cadeia de replicação daquele *peer* diagnosticado como falho. Na sequência, o mesmo *peer* replicará os blocos de dados referentes a seus arquivos ao seu novo vizinho, substituindo o *peer* falho em sua cadeia de replicação. Após esse procedimento, é identificado o novo *peer* responsável pelo vértice em que se encontrava o *peer* falho e, por fim, caso o *peer* que esteja executando o algoritmo não for responsável pelo vértice deixado pelo *peer* falho, este deverá enviar todos os blocos referentes ao *peer* falho ao novo responsável pelo vértice desocupado. A Figura 3 apresenta esse processo. Os vértices disponíveis são identificados pelos círculos com a cor branca, e, o indisponível, representado pelo círculo com maior destaque. Cada *peer* armazena os blocos de dados referentes a cada réplica dos arquivos que possui.

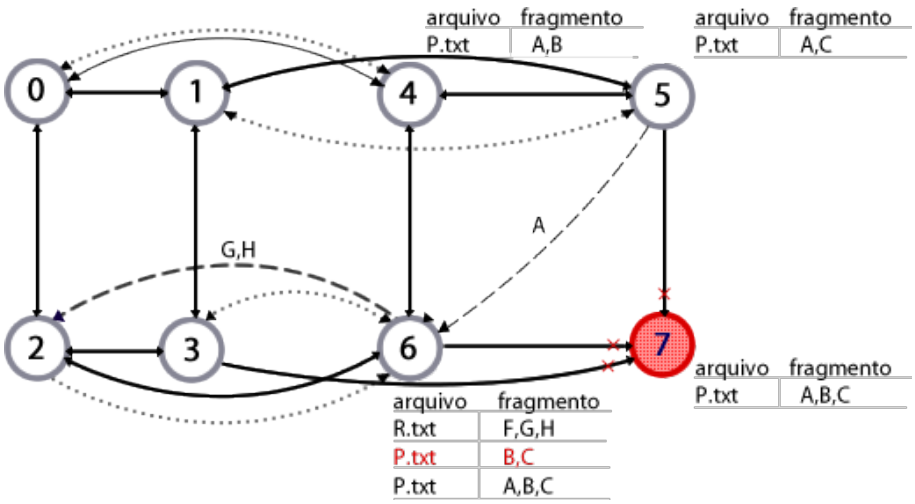


Figura 3. Exemplo da redistribuição dos blocos pelos *peers* do sistema.

Após o diagnóstico do sistema, todos os participantes possuem conhecimento do estado dos demais, porém, os que iniciam o procedimento de reorganização do sistema são aqueles que possuíam o *peer* falho como membro de seu grupo de replicação. Desse modo, o *peer* 6 busca um novo vizinho, substituindo o *peer* 7 e enviando a réplica de seu arquivo (em blocos) a ele. Nota-se que nesse processo, os blocos de dados do *peer* 6 (G e H) referentes ao arquivo R.txt são enviados ao *peer* 2. O *peer* 5 é responsável por enviar o bloco de dados referentes ao *peer* falho. Portanto, o *peer* 5 envia o bloco A ao novo responsável, sendo este o *peer* 6. Esse procedimento é representado pelas conexões tracejadas. Após esse processo, o *peer* 6 possuirá todos os blocos de dados do arquivo referente ao *peer* 7. Da mesma forma que o *peer* 6 identificou seu novo vizinho, os demais *peers* também terão de fazê-lo. Tal ação é representada pelas conexões pontilhadas.

5. Avaliação Experimental

Com o intuito de simular um cenário próximo a uma rede de computadores baseada na Internet, foram construídas diferentes redes constituídas de *peers*, *roteadores* e *links* de comunicação. Foram criados cenários com 8, 16, 32, 64, 128, 256 e 512 participantes.

O número de roteadores foi definido por um valor logarítmico, representado por $\log_2 N$, sendo N o número de participantes da rede. Consequentemente, o número de

roteadores presentes em uma rede com 8 participantes foi limitado a 3. Em um cenário composto por 16 participantes foram definidos 4 roteadores, e assim sucessivamente. As conexões entre os roteadores também foram definidas aleatoriamente, de modo que um roteador não pode se conectar a todos os demais, mas sim com $R - 2$ vizinhos, sendo R o número de roteadores. Foi definido que cada roteador terá no mínimo 1 e no máximo $N - (R - 1)$ participante(s) da rede conectado(s) a ele. A largura de banda de cada *link* de conexão também foi aleatorizada, possuindo valores entre 20 a 40 megabits por segundo. Além disso, a latência de cada *link* foi aleatorizada com valores entre 10 a 40 milissegundos.

Para todos os testes, os fatores de replicação e fragmentação foram ajustados em $log_2 N$, sendo N o número de *peers* do sistema. Portanto, em um sistema com 8 *peers*, cada *peer* terá 3 vizinhos e, no VCubeDHT, cada arquivo será fragmentado em 3 blocos. O tamanho do arquivo configurado no simulador SimGrid [Legrand et al. 2016], varia entre 1, 10, 100 megabytes a 1 gigabyte. O roteamento dos pacotes de dados entre os participantes da rede foi feito pelo algoritmo Dijkstra [Dijkstra 1959]. Por fim, o critério escolhido para comparação entre os sistemas foi o tempo total de simulação.

5.1. Avaliação do Protocolo de Entrada

A etapa de análise do Protocolo de Entrada (PE) tem como objetivo exibir os resultados obtidos em relação ao método de propagação de dados utilizado pelo HyperDHT e VCubeDHT, destacando o resultado obtido durante o Protocolo de Entrada (PE). Vale ressaltar que em todos os testes, não foram geradas falhas durante a entrada dos participantes na rede. Além disso, o critério escolhido para comparação entre os sistemas foi o tempo de simulação total para que os participantes da rede repliquem seus dados. A Tabela 1 exibe os resultados dos testes de acordo com os parâmetros pré-estabelecidos. Nota-se que as colunas identificadas por (H) representam o HyperDHT, já aquelas em (V), o VCubeDHT. Em todos os testes o VCubeDHT obteve melhor tempo em relação ao HyperDHT, isso ocorreu pois o algoritmo proposto replica um conjunto menor de dados.

No cenário onde a rede é composta por 8 *peers* e cada um possui um arquivo de 1 MB (megabyte), no HyperDHT, cada *peer* transfere 3 MB, totalizando 24 MB propagados em 11,67 segundos. Já no sistema VCubeDHT, cada *peer* transfere dois blocos de aprox. 350 KB, totalizando cerca de 5 MB transferidos em 9,39 segundos.

No cenário descrito, houve um número menor de *bytes* a ser transferido pelo VCubeDHT, sendo aprox. 20 MB. Nos cenários com 8 e 512 *peers*, nota-se um resultado próximo para ambos sistemas. Isso ocorre pois a latência impactada na transferência do arquivo completo pelo HyperDHT é bem próxima a latência inferida no envio dos blocos feitos pelo VCubeDHT. Em razão disso, o ganho de desempenho do VCubeDHT não é muito significativo, mesmo com um número menor de *bytes* enviados. Além disso, é possível observar que o mesmo ocorre para os diferentes cenários. As Figuras 4 a 7 apresentam quatro gráficos de comparação entre os sistemas VCubeDHT e HyperDHT, de acordo com os resultados obtidos na Tabela 1.

5.2. Avaliação da Reorganização do Sistema

Após a detecção de um participante falho na rede, tanto o HyperDHT como o VCubeDHT tem como objetivo inicial identificar o novo *peer* responsável pelo vértice onde estava o

Tabela 1. Tempos de execução do protocolo de entrada do HyperDHT (H) e do VCubeDHT (V).

Peers	Tempo (segundos)							
	H	V	H	V	H	V	H	V
8	11,67	9,39	61,41	39,15	581,18	366,83	5917,56	3731,54
16	20,29	17,63	108,22	81,76	1047,35	769,15	10656,76	7815,15
32	28,64	25,47	179,39	140,84	1783,46	1406,43	18233,48	14374,41
64	47,96	40,53	345,92	288,01	3303,90	2727,09	34475,60	28479,64
128	47,35	42,42	341,96	289,03	3211,84	2823,17	33355,79	28379,51
256	83,93	71,56	743,39	653,22	7475,21	6465,41	75886,69	66317,94
512	108,67	101,72	1055,13	935,04	10468,69	9339,63	107317,68	95121,49
	1 MB		**10 MB**		**100 MB**		**1 GB**	
	Tamanho do Arquivo							

Figura 4. PE - Arquivo de 1 MB

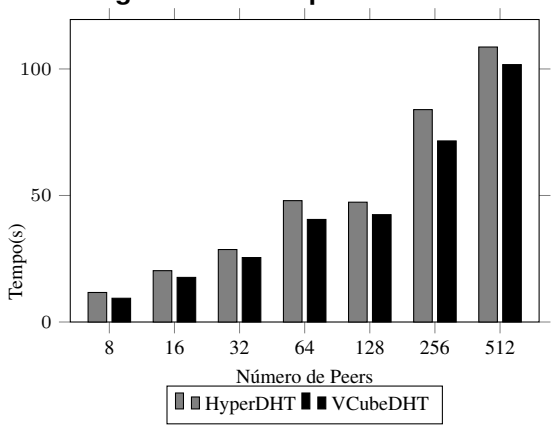
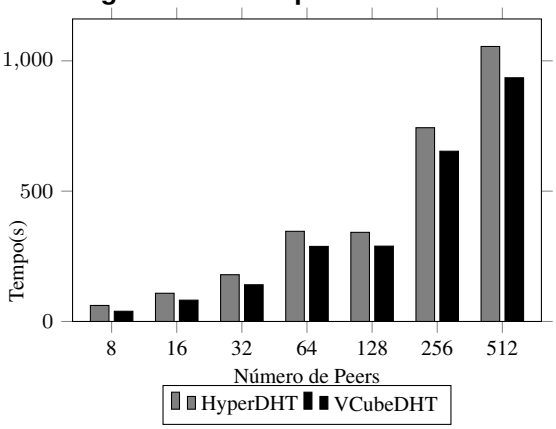


Figura 5. PE - Arquivo de 10 MB



participante falho, de modo a responder pelos dados armazenados por esse. Portanto, após essa etapa, inicia-se o processo de Reorganização do Sistema (RS), que verifica o comportamento do VCubeDHT após o diagnóstico de *peers* falhos na rede.

Nessa etapa foi considerado que nenhum novo *peer* solicita a entrada na rede. A escolha inicial do *peer* falho ocorreu de forma aleatória, porém, posteriormente foi pré-configurada a falha desse para coleta dos resultados em um sistema com maior número de participantes e com arquivos de tamanhos maiores. A Tabela 2 exibe os resultados dos testes realizados de acordo com os parâmetros pré-estabelecidos. O tempo de simulação foi novamente escolhido como critério de desempenho.

Baseado no cenário composto por 8 *peers*, onde o *peer* 7 encontra-se falho e o arquivo a ser replicado possui 1 *megabyte*, no sistema HyperDHT os *peers* 4, 5 e 6 (pertencentes ao grupo de replicação do *peer* 7), enviam uma réplica de seu arquivo aos *peers* 0, 1 e 2 nesta ordem, transferindo um total 3 MB. No VCubeDHT os *peers* 4, 5 e 6 enviam parte de seu arquivo, em dois blocos de 350 KB, aos *peers* 0, 1 e 2, nesta ordem. Em seguida, é identificado pelo *peer* 5, o único bloco de dado necessário para que o *peer* 6 possua o arquivo completo referente ao *peer* falho. Na sequência o *peer* 5 transfere

Figura 6. PE - Arquivo de 100 MB

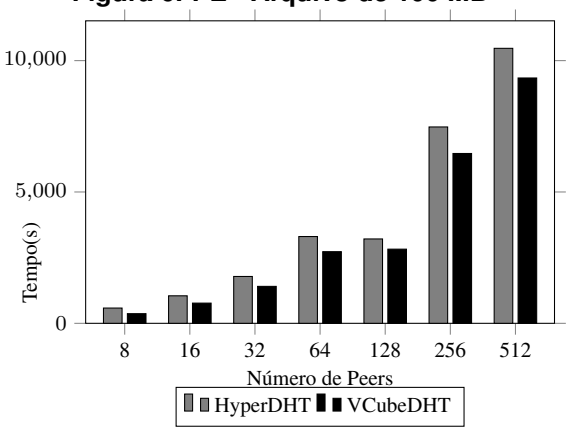
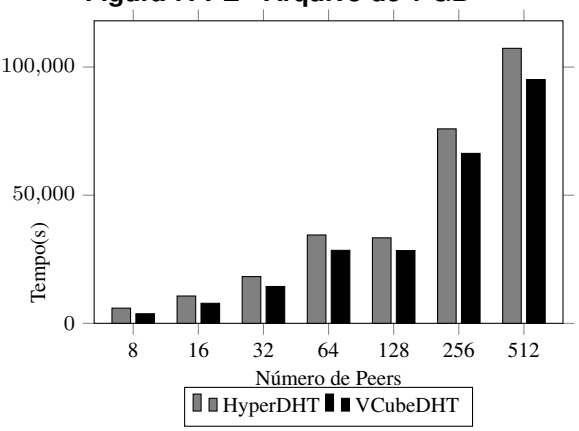


Figura 7. PE - Arquivo de 1 GB



apenas um bloco de dado cujo tamanho é 350 KB ao *peer* 6. Portanto, o número de *bytes* transferidos pelo VCubeDHT foi de aprox. 2,5 MB contra aprox. 3 MB no HyperDHT, ou seja, cerca de 700 MB a menos. Nota-se, através da Tabela 2, que o VCubeDHT obteve pior desempenho, transferindo as réplicas em 2,12 segundos, sendo que o HyperDHT as transferiu em 1,50 segundos. As Figuras 8 e 9 exibem os arquivos de *log* da execução dos sistemas de acordo com o cenário descrito.

Tabela 2. Tempo para a reorganização do HyperDHT (H) e do VCubeDHT (V).

Peers	Tempo (segundos)							
	H	V	H	V	H	V	H	V
8	1,50	2,12	8,25	7,73	76,91	66,48	781,81	669,59
16	1,89	2,56	6,70	7,10	54,77	52,44	548,32	517,94
32	3,12	2,99	9,26	9,05	77,36	70,34	776,56	699,62
64	3,83	4,81	9,67	10,19	72,14	67,35	717,55	658,98
128	1,64	3,79	6,67	6,48	58,27	50,08	588,07	504,19
256	2,88	3,79	9,69	10,13	77,79	73,50	776,99	704,14
512	2,94	3,64	10,72	11,02	88,55	86,45	887,63	861,12
Tamanho do Arquivo	**1 MB**		**10 MB**		**100 MB**		**1 GB**	

É possível observar que para cada dado enviado, o VCubeDHT obteve um pequeno ganho, porém, o tempo para transferência de cada dado em ambos sistemas é próximo, mesmo que o número de *bytes* enviado pelo HyperDHT seja superior ao VCubeDHT. Além disso, é possível notar que o *peer* 5, após enviar os blocos de dados 1 e 2 ao *peer* 1, aguarda até o mesmo recebe-lo (devido ao envio bloqueante) e, posteriormente realiza o envio do bloco de dado 1 ao *peer* 6. Nota-se que até o momento o VCubeDHT possui uma pequena vantagem, porém, ao emitir o último bloco, o tempo total se torna maior em relação ao obtido pelo HyperDHT.

Mesmo com grande parte dos resultados destacando um melhor desempenho do sistema HyperDHT, onde o tamanho do arquivo armazenado pelos *peers* é de 1 *megabyte*, é possível notar que o VCubeDHT obteve um melhor desempenho em uma rede composta

```
[peer-5: 160000.004] send: peer-4 | rec: peer-0 | size block: 699050.666 | data: 730750_R_1_2
[peer-6: 160000.005] send: peer-5 | rec: peer-1 | size block: 699050.666 | data: 913438_R_1_2
[peer-7: 160000.006] send: peer-6 | rec: peer-2 | size block: 699050.666 | data: 109612_R_1_2
[peer-1: 160001.043]    peer-0 receive piece of peer-4  | data: 730750_R_1_2
[peer-1: 160001.043]    size block: 699050.66 | time to transfer: 1.039

[peer-3: 160001.130]    peer-2 receive piece of peer-6 | data: 109612_R_1_2
[peer-3: 160001.130]    size block: 699050.66 | time to transfer: 1.124

[peer-2: 160001.397]    peer-1 receive piece of peer-5 | replicas: 3 | data: 913438_R_1_2
[peer-2: 160001.397]    size block: 699050.66 | time to transfer: 1.392

[peer-6: 160001.397] send: peer-5 | rec: peer-6 | size block: 349525.333 | data: 127881_R_1

[peer-7: 160002.127]    peer-6 receive piece of peer-5 | data: 127881_R_1
[peer-7: 160002.127]    size block: 349525.33 | time to transfer: 0.729

[peer-7: 165002.127] time send: 160000.004 | time receive: 160002.127 | time total = 2,123
```

Figura 8. Arquivo do *log* de execução do VCubeDHT composto por 8 *peers*.

```
[peer-5: 160000.004] send: peer-4 | rec: peer-0 | size block: 1048576.0 | data: 730750
[peer-6: 160000.005] send: peer-5 | rec: peer-1 | size block: 1048576.0 | data: 913438
[peer-7: 160000.006] send: peer-6 | rec: peer-2 | size block: 1048576.0 | data: 109612

[peer-1: 160001.281] peer-0 receive piece of peer-4 | replicas: 4 | data: 730750
[peer-1: 160001.281] size file: 1048576.0 | time to transfer: 1.277

[peer-3: 160001.385] peer-2 receive piece of peer-6 | replicas: 4 | data: 109612
[peer-3: 160001.385] size file: 1048576.0 | time to transfer: 1.379

[peer-2: 160001.502] peer-1 receive piece of peer-5 | replicas: 4 | data: 913438
[peer-2: 160001.502] size file: 1048576.0 | time to transfer: 1.497

[peer-2: 165001.502] time send: 160000.004 | time receive: 160001.502 | time total = 1,498
```

Figura 9. Arquivo do *log* de execução do HyperDHT composto por 8 *peers*.

de 32 *peers*. Tal desempenho se deve ao fato de que a rota percorrida pelo *peer* responsável por replicar o último bloco de dado possui menor latência, logo, o VCubeDHT manterá a pequena vantagem que possuía no decorrer do envio dos blocos de dados.

Nota-se que se a rota percorrida pelo *peer* responsável por replicar o último bloco de dado referente ao *peer* falho possuir consideravelmente uma menor latência em relação as demais, o resultado na replicação de um arquivo pequeno pelo VCubeDHT se mostrará mais eficiente em relação ao HyperDHT. Caso contrário, possuirá um pior desempenho.

A partir dos dados da Tabela 2, nota-se que o tempo de transferência de ambos sistemas diminui constantemente a medida com que o tamanho do arquivo é maximizado. No momento em que os sistemas trabalham com arquivos de 100 *megabytes* de tamanho, o VCubeDHT começa a obter um melhor desempenho em relação ao HyperDHT.

Através do método utilizado pelo simulador para efetuar o cálculo do tempo de comunicação entre os participantes do sistema, notou-se que a latência impacta fortemente no envio de dados com tamanhos menores, como na replicação de arquivos com 1 *megabyte*, onde o VCubeDHT obteve pior desempenho. A medida com que o tamanho do arquivo aumenta, torna-se visível a eficiência do VCubeDHT sobre o HyperDHT. As Figuras 10 a 13 apresentam quatro gráficos de comparação entre os sistemas VCubeDHT e HyperDHT, de acordo com os resultados obtidos na Tabela 2.

Vale ressaltar que não se torna possível, de acordo com o cenário estipulado, comparar as diferentes redes, isto é, a rede composta por 128 *peers* com a de 64 *peers*. Isso ocorre pois mesmo que o número de réplicas a serem enviadas no primeiro seja maior que segundo cenário, as características da rede, serão diferentes para ambos e impactarão de

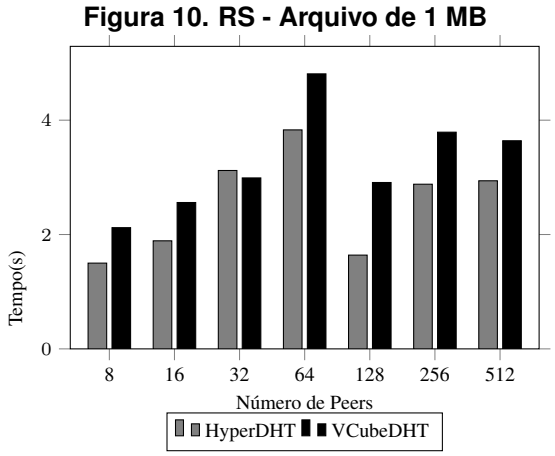


Figura 10. RS - Arquivo de 1 MB

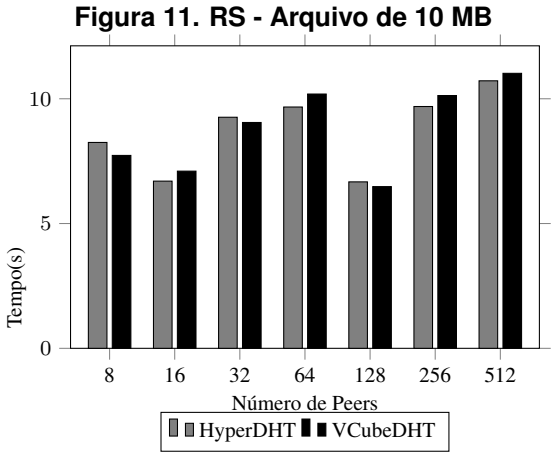


Figura 11. RS - Arquivo de 10 MB

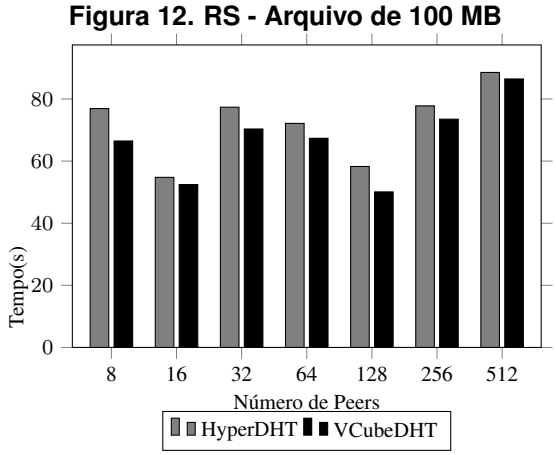


Figura 12. RS - Arquivo de 100 MB

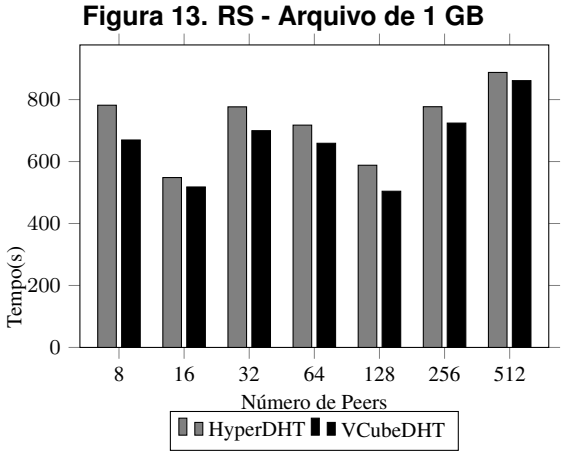


Figura 13. RS - Arquivo de 1 GB

diferentes formas. Além disso, de acordo com o algoritmo de Reorganização do Sistema, os novos vizinhos a serem identificados serão diferentes para ambas as redes.

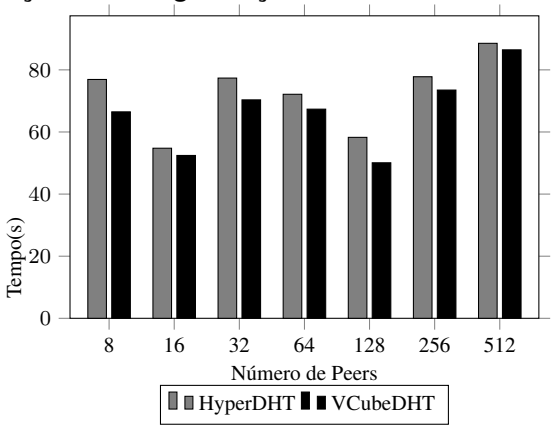
5.3. Disponibilidade dos Dados

Além de analisar o desempenho do algoritmo proposto na Reorganização do Sistema, há a necessidade de investigar a disponibilidade dos dados garantida por ele. Sendo assim, a partir de outro modo de visualização dos resultados da ultima análise, é possível definir cenários que destacam a disponibilidade dos sistemas HyperDHT e VCubeDHT.

A Figura 14 exibe os resultados obtidos na Reorganização do Sistema com arquivos de 100 *megabytes* de tamanho. Analisando a primeira coluna, que representa a rede composta por 8 *peers*, nota-se que, se por algum motivo, após 66,48 segundos do início do processo de reorganização do sistema, os participantes que estavam replicando seus dados se tornem falhos, o sistema HyperDHT não garantiria a disponibilidade total dos dados armazenados pelos *peers*. Já no VCubeDHT, observa-se que após esse instante, os *peers* já haveriam terminado a reorganização, garantindo uma maior disponibilidade. Porém, nos cenários onde o VCubeDHT replica arquivos de 10 *megabytes* de tamanho, nota-se que este não garante uma melhor disponibilidade sobre o HyperDHT, visto que o tempo para reorganizar o sistema, no primeiro, na maior parte dos casos, se torna superior.

No HyperDHT não há a redistribuição dos dados referentes ao *peer* falho. Isso

Figura 14. Replicação na Reorganização do Sistema com Arquivo de 100 MB



ocorre pois o novo responsável possuirá a réplica dos arquivos em questão. Porém, em um cenário onde os *peers* não falhem instantaneamente, após a falha de todos os *peers* pertencentes a um único grupo de replicação, nota-se que as réplicas do primeiro *peer* a se tornar falho não existirão mais no sistema. Já no VCubeDHT, como há essa etapa de redistribuição dos blocos, todas as réplicas estarão disponíveis no sistema. Sendo assim, o VCubeDHT garante, novamente, uma maior disponibilidade dos arquivos.

6. Considerações Finais

A principal contribuição deste trabalho é a investigação da replicação de dados a partir da fragmentação de arquivos em uma DHT de um salto sobre a topologia virtual VCube. O VCube tem várias propriedades logarítmicas, que garantem a escalabilidade da solução proposta. Na abordagem empregada pelo VCubeDHT, os *peers* do sistema replicam seus dados de forma parcial, buscando obter melhorias em relação ao tempo de transmissão e disponibilidade, garantindo a tolerância a falhas de participantes da rede responsáveis por determinados segmentos de dados. O sistema foi implementado através de simulação e foram feitos testes comparativos com outro sistema DHT, utilizado como base para o desenvolvimento deste trabalho.

Com base nos resultados obtidos, pode-se concluir que a alternativa de replicação baseada em fragmentação, implementada pelo VCubeDHT, se mostra mais eficiente que a alternativa de replicação total do HyperDHT, onde os arquivos a serem replicados possuem tamanhos iguais ou superiores a 100 *megabytes*. Na emissão de arquivos cujos tamanhos são inferiores a 100 *megabytes*, o algoritmo proposto obtém melhores resultados se a latência impactada na emissão do último bloco de dado referente ao *peer* falho for significantemente menor que as demais. Além disso, é importante ressaltar que a alternativa abordada pelo VCubeDHT garante a disponibilidade total dos dados caso os *peers* do sistema não se tornem falhos instantaneamente. Porém, no pior caso, o sistema mantém a disponibilidade dos arquivos, tolerando $frag - 1$ falhas, onde $frag$ representa o fator de fragmentação configurado no sistema.

Neste trabalho, as informações armazenadas pelos participantes do sistema não sofrem alterações. Dessa forma, propõe-se como trabalho futuro, a exploração do VCubeDHT para que este suporte trabalhar com arquivos modificados, garantindo a consistência dos dados. Além disso, propõe-se a implementação e execução do VCubeDHT em

um ambiente real, ou mesmo em um ambiente de testes como o PlanetLab.

Agradecimentos

O trabalho foi desenvolvido no âmbito do grupo PET-Comp, financiado pelo Ministério da Educação (MEC), com apoio da Fundação Araucária/SETI, proj. 144/2015-45112 e CNPq, proj. 309143/2012-8.

Referências

Androutsellis-Theotokis, S. e Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36:335–371.

Bona, L. C. E., Duarte Jr., E. P., Mello, S. L. V. e Fonseca, K. (2006). Hyperbone: Uma rede overlay baseada em hipercubo virtual sobre a internet. In: *Anais do XXIV Simpósio Brasileiro de Redes e Sistemas distribuídos*, SBRC'2006.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *NUMERISCHE MATHEMATIK*, 1(1):269–271.

Duarte, Jr., E. P., Bona, L. C. E. e Ruoso, V. K. (2014). VCube: A provably scalable distributed diagnosis algorithm. In: *5th Work. on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ScalA'14, pp. 17–22, Piscataway, USA. IEEE Press.

Ghodsi, A. (2006). *Distributed k-ary system: Algorithms for distributed hash tables*. Tese de Doutorado, Department of Electronic, Computer, and Software Systems, KTH-Royal Institute of Technology.

Koppe, J. P., de Bona, L. C. E. e Jr., E. P. D. (2014). Hyperdht - dht de um salto baseada em hipercubo virtual distribuído. In: *Anais do IX Workshop de Redes P2P, Dinâmicas, Sociais e Orientadas a Conteúdo*, SBRC-Wp2p+.

Legrand, A., Marchal, L. e Casanova, H. (2016). Scheduling distributed applications: the simgrid simulation framework.

Ratnasamy, S., Francis, P., Handley, M., Karp, R. e Shenker, S. (2001). A scalable content-addressable network. In: *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pp. 161–172, New York, NY, USA. ACM.

Raynal, M. (2013). *Distributed Algorithms for Message-Passing Systems*. Springer Publishing Company, Incorporated.

Ruoso, V. K. (2013). *Uma estratégia de testes logarítmica para o algoritmo Hi-ADSD*. Tese de mestrado, Universidade Federal do Paraná, Curitiba, PR.

SEU Mitigation for SRAM FPGAs: A comparison via Probabilistic Model Checking

Viny Cesar Pereira[1], **Valdivino Alexandre de Santiago Júnior**[1], **Silvio Manea**[1]

[1]Instituto Nacional de Pesquisas Espaciais (INPE)
Av. dos Astronautas, 1758, São José dos Campos, São Paulo – SP – Brazil

{viny.pereira,valdivino.santiago,silvio.manea}@inpe.br

Abstract. *Although there are several Single-Event Upset (SEU) mitigation techniques for SRAM-based Field Programmable Gate Arrays (FPGAs), comparisons are still necessary regarding dependability analyzes of these techniques. Most of these assessments analyze the techniques after design and implementation in FPGA which may be too costly. Stochastic/Probabilistic analysis allow to obtain results in the early stages of design. In this paper, we compare three of these strategies, Scrubbing, Triple Modular Redundancy (TMR), and Hamming code, via Probabilistic Model Checking. Results show that TMR allows upsets to accumulate and must be combined with Error Correction Codes (ECCs), such as Hamming, and that the Scrubbing interval directly affects reliability while safety is more related to the coverage rate.*

1. Introduction

Ensuring safety, reliability and availability in complex systems has been a major challenge for designers, especially in applications where hardware and software failures can cause catastrophic financial and scientific harm. For aerospace systems development, SRAM-based Field Programmable Gate Arrays (FPGAs) have some advantages over the Anti-fuse technology such as decreased cost and reconfiguration flexibility [Berg et al. 2008]. However, the harsh environment of space may trigger the occurrence of Single-Event Effects (SEEs) such as Single-Event Upset (SEUs) in SRAM-based FPGAs. Thus, it is necessary to develop techniques to mitigate the consequences of SEUs in FPGA's configuration and funcional logic paths.

Several techniques of detection, mitigation and correction of SEU have appeared in the past as a way to avoid failures in space systems based on FPGAs [Heiner et al. 2009, Hentschke et al. 2002, Liu et al. 2012, Morgan et al. 2007]. In recent years, researchers have proposed improvements to adapt these techniques to advances in the manufacturing process of integrated circuits. The most common approaches involve hardware redundancy, such as the Triple Modular Redundancy (TMR) [Rollins et al. 2003] which consists of tripling circuit modules that demonstrate greater sensitivity to upsets and, via a voting mechanism, to decide the correct output based on the majority. Other forms of redundancy exist, such as temporal redundancy which masks the occurrence of a fault because it obtains the signal at different times, saves the values obtained and, as in TMR, uses a voter to decide which output is correct.

Another category of technique is known as Error Correction Code (ECC) based on the addition of information (parity bits) next to the input data to detect and mitigate possible errors [Dutta and Touba 2007]. There are several ECCs used in the context of SEUs

such as Hamming code [Kumar and Umashankar 2007], Reed-Solomon code, and Bose-Chaudhuri-Hocquenghem (BCH) codes. By checking the parity bits these techniques are able to detect and correct single-bit errors and, in some more complex implementation, also correct double-bit errors [Kastensmidt 2007].

Comparative analyzes of SEU mitigation techniques have been recently published [Shuler et al. 2009, Morgan et al. 2007, Liu et al. 2012, Hentschke et al. 2002]. Many factors must be taken into account before deciding on one particular technique or even the decision to combine the techniques. Some important aspects such as additional area required, performance impact, ability to correct faults or just mitigate them, robustness against multiple upsets may interfere in deciding which technique is most interesting to use. Studies such as [Ejlali et al. 2006, Sari et al. 2013] suggest combining techniques for best results, however, ensuring performance, reliability and safety can become a challenge given the complexity of the problem. To validate these techniques, most of these works use simulations and tests after design and implementation in FPGAs, and fault-injection tools to simulate the upsets and evaluate the behavior of the techniques. These approaches are not able to evaluate, earlier within the development process, all possible situations and may miss faults that will be costly to detect and correct later.

Formal Verification methods [Baier et al. 2008] have been used in several different application domains. Particularly, Probabilistic Model Checking [Kwiatkowska et al. 2009] has shown to be an efficient and robust technique for modeling a wide variety of problems in diverse fields such as biology, security, network and communication protocols, performance and reliability just to name a few. Probabilistic models such as Discrete-Time Markov Chain (DTMC), Continuous-Time Markov Chain (CTMC) and Markov Decision Process (MDP) have been employed in this context.

In [Hoque et al. 2014, Hoque 2016] the authors analyzed the dependability of SEU mitigation methods on SRAM-based FPGAs via Probabilistic Model Checking. However, the authors' approach deals with a combination of two techniques (TMR and Scrubbing), which when compared, are placed in only two situations: only Scrubbing and TMR with Scrubbing. In addition to not being considered independently, the TMR was analyzed in a simplified way, only as spares components, ignoring important details such as the voter and the input and output data. Hence, in this paper we present a comparison between three popular SEU mitigation techniques for SRAM FPGAs, Scrubbing [Berg et al. 2008], TMR [Kastensmidt et al. 2005], and Hamming Code [Kumar and Umashankar 2007], via Probabilistic Model Checking. Our dependability analysis relied on CTMC and Continuous Stochastic Logic (CSL) extended with Rewards. Results show that TMR allows upsets to accumulate and must be combined with ECCs, such as Hamming, and that the Scrubbing interval directly affects reliability while safety is more related to the coverage rate.

This paper is structured as follows. Section 2 presents some background to better understand the three mitigation techniques and an overview of Probabilistic Model Checking. Section 3 shows a general explanation of how the CTMC models were created to support our analysis, and what types of CSL properties we considered. Dependability results and discussion are in Section 4. Section 5 presents related work. In Section 6, we point out the conclusions and future directions of this research.

2. Background

This section presents details of the SEU mitigation techniques addressed in the paper and the concepts related to Probabilistic Model Checking.

TMR is one of the most commonly used technique to prevent failures caused by SEUs, especially single bit upsets (SBUs) [Kastensmidt et al. 2005]. In general, the TMR strategy is to use three replicates of the components that are most sensitive to the effects of radiation, because in case of upset the voting mechanism will be able to choose through the majority what will be the correct output [Rollins et al. 2003].

The masking of the component reached by radiation is possible because TMR removes individual points of failure through redundancy however, as in any redundant system, there is a cost. To ensure that a failure does not reach a common point between all replicas and propagates the fault to the output, each redundant component must have individual input and output ports, causing an approximately six-fold increase in the required memory area [Morgan et al. 2007]. Thus, deciding which logical level the TMR will be applied is a complicated task, and the relation between the additional area and the effective gain must be considered.

In addition to area overload, another important feature about TMR is that due to masking of the fault rather than the correction, the hit component will continue to deteriorate, causing an upset buildup [Kastensmidt 2007]. TMR should be considered in applications with a low incidence of MBUs, since it is not capable of handling more than one simultaneous failure in the replicated modules.

Scrubbing is the mechanism that reconfigures the memory using the bitstream with the original configuration, usually stored in a memory that is immune to radiation and SEEs [Sari et al. 2013]. One of the greatest advantages of Scrubbing over full reconfiguration is that Scrubbing does not completely interrupt the operation mode of the system, only part of the configuration is updated with each Scrubbing interval [Berg et al. 2008].

The most common form of Scrubbing is known as Blind Scrubbing because it does not include a fault detection mechanism, only a copy of the original data is used to rewrite the current configuration. One of the decisions in blind Scrubbing is the time interval, i.e. the frequency at which a cycle should occur. The Scrubbing interval depends on the level of reliability desired by the system and how often the upsets occur. We also consider the coverage rate which is responsible for the conditional probability of detecting the fault given that the fault exists. In this way, it is possible to change the coverage rate of the model and observe how it behaves. Other more sophisticated forms of Scrubbing, such as Readback Scrubbing, use detection techniques and are able to restore settings only when a failure is reported [Heiner et al. 2009].

Hamming code is one of the techniques of ECC used in several applications for the detection of single or double bit errors and correction of single bit errors in binary codes [Dutta and Touba 2007]. As in TMR, Hamming code is a technique indicated in applications with low incidence of MBUs.

The operation of the Hamming code is structured in two modules, an encoder and a decoder. The encoder is responsible for calculating the parity bits based on the original data and, as the name says, encode it to the bitstream [Kastensmidt et al. 2005]. The

actual work is done by the decoder which must check the parity bits again to see if there is a error. In case of error, the decoder is also responsible for finding which bit has failed and repairing it [Liu et al. 2012]. There are several Hamming code variations designed to serve applications with different goals. In this research, we used Hamming code (7,4) which encodes three parity bits for each four bits of data.

2.1. Probabilistic Model Checking

Critical complex systems used in aerospace, medical, communication and other applications are very sensitive to software and/or hardware failures. To date, classical systems validation methods include simulation which is still utilized at a modeling level and later, with the model already constructed, the test cycle runs. Although efficient for most problems, when it comes to critical systems, only simulation may not be sufficient to address all possibilities and ensure the desired level of safety.

To increase accuracy, Formal Methods have shown great potential to reduce the time spent and increase the effectiveness of the modeling process by applying mathematical rigor to ensure system correctness [Baier et al. 2008]. Model Checking is a very popular Formal Verification method that exhaustively explores the state space in an automated way and is able to find failures in the solution (Transition System - TS) like unreachable states or deadlocks. Typically, a property is formalized via a variety of temporal logics such as Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) and the approach systematically verifies if the Transition System (model) satisfies this property ($TS \models \Phi$).

Within Model Checking, a category that deals with systems that present stochastic behavior is Probabilistic Model Checking [Baier et al. 2008, Kwiatkowska et al. 2009, Hoque et al. 2014] where, instead of the absolute guarantee of correctness hardly achieved by Model Checking, is able to guarantee, within a specified probability and in a less demanding manner, the correctness of the system. Markov chains (DTMCs, CTMCs) are typically used as the probabilistic model of the system. Variations of CTL, such as Probabilistic Computation Tree Logic (PCTL) for DTMC and CSL for CTMC, are used in this approach.

3. CTMC Models and CSL Properties

For our dependability analysis we used PRISM [Kwiatkowska et al. 2009], a tool that allows to create stochastic models and check their properties. We selected CTMC to model the three mitigation techniques and properties were formalized in CSL. The CSL operators P, S and R indicate the probability of an event occurring, the long-run probability of a condition being satisfied and the values returned by the costs or rewards of the models, respectively.

Our work is based on [Hoque 2016, Hoque et al. 2014] where the authors investigated Scrubbing and TMR combined. The CTMC models represent SEU mitigation techniques in applications that use adders and multipliers because these components are implemented in the memory of SRAM-based FPGAs, therefore, sensitive to upsets. Failure rates (λ) of the components were estimated in [Hoque 2016], using per bit upset rate for Xilinx Virtex-5 in Highly Elliptical Orbit (HEO), which is $7.31\text{x}10^{-12}$ SEUs/bit/sec.

This rate is multiplied by the number of critical bits of each component. For the components used, the Mean Time Between Failures (MTBF) was 11.85 days for the Wallace Tree Multiplier and 38.15 days for the Kogge-Stone Adder. Hence, $\lambda = 1/MTBF$.

The mitigation techniques were designed in PRISM by describing the behavior of each technique and the characteristics desired for the system, such as quantity of components, size of input and output data, failure rates and coverage rates. With this information, each model was implemented in modules, where each module has sets of commands that will make the necessary transitions to represent the states of the models.

For Scrubbing, the model was provided in [Hoque 2016] but rather than analyzing the results with adders and multipliers limited to three components each, our model handled 10 adders and 10 multipliers to evaluate the availability, safety and reliability of Scrubbing into a more complex application that requires more components in the configuration. In summary, our aim here is to confirm whether the conclusions presented in [Hoque 2016] are valid for more complex applications. In Scrubbing there are only two modules, one for adders and another for multipliers and the function of these modules is to control the amount of operational and degraded components.

As the model addresses Blind Scrubbing, there is no fault detection technique and the repair is performed regardless of the number of operating components. The Scrubbing interval stipulated in [Hoque 2016] is 1, 4 and 9 days and occurs synchronously between adders and multipliers.

The CTMC model in [Hoque 2016] does not actually address a true TMR. In his analysis, the author used a model with Scrubbing and to represent the TMR, added spare components, but ignored the structure and functioning of the TMR, since it did not include the majority voting mechanism nor the input and output data of each component. In addition to a superficial modeling of the technique, the TMR was not analyzed individually, only combined with Scrubbing. Thus, we decided to create our own TMR model.

Our PRISM code has eight modules: six represent the triple redundancies of the adders and multipliers, and there is one voter for the adders and another for the multipliers. Figure 1 presents PRISM's code of the first adder, the same logic was applied to multipliers. Note that `a1` and `a2` represent the 4-bit inputs, `outA1` represents the output of the module, `Na` is the number of operational adders and `lambda_A` is the failure rate stipulated for the adders. PRISM's language allows synchronization between modules and we have used this feature in our CTMC model. Unlike the model in [Hoque 2016], our solution is more refined in the sense that we explicitly represent inputs (4-bit for adders and multipliers) and outputs (5-bit for adders and 8-bit for multipliers), as well as the modules that actually represent the functioning of a voter. The calculation of the event rates in synchronized commands is the product of the rates stipulated for each command. We chose the passive/active approach: rate 1 in one module and the true λ in the other module.

Initially, all components are operational and receive two 4-bit input values. Based on the stipulated failure rate, each module can function properly and assign the correct value to the output or have an upset and return an incorrect value. The failure rate is multiplied by the number of operating components, so the upset modules will not be taken into account in the next cycle. With the output values returned by the modules,

```
module adder1
  a1 : [0..15] init 0;
  a2 : [0..15] init 0;
  outA1 : [0..31] init 31;
  [input] (a1=0 & a2=0) -> 1: (a1'=5) & (a2'=7);
  [adder] ((a1 > 0 | a2 > 0) & (Na > 0)) -> Na*(1-lambda_A) :
  (outA1'= a1+a2) + Na*lambda_A : (outA1'= a1+a2-1);
endmodule
```

Figure 1. Adder module for our TMR model in PRISM.

the voters take action and decide what the final output will be based on the majority, so TMR is able to ignore the occurrence of an upset because the other two modules have the correct output value. In the case of two simultaneous upsets, the voter will consider the wrong output as a majority, and consequently the result will be incorrect.

To classify the states of CTMC models, PRISM offers the concept of formulas. In our TMR we separated them into three types: operational, where all components are functional; degraded, where at most one component of adders and multipliers suffered an upset, but the output is still correct; and failed, where more than one component has suffered upset and the output generated by the voter is incorrect. A representation of the *failure* formula is shown below:

```
formula fail = ((outMf!=m1*m2) | (outAf!=a1+a2));
```

We considered a third SEU mitigation approach in our research. The model developed for the Hamming code is not directly related to adders and multipliers because it can be implemented to detect errors in any memory component of SRAM-based FPGAs. To represent the operation, two modules, an encoder and a decoder, were created, where the encoder receives a 4-bit input and based on these values, calculates the 3 additional parity bits. The detection and correction of a possible error is performed in the decoder which can receive the unchanged bits from the encoder or insert a error in any of the bits based on a failure rate.

As the model is generic, the failure rate of the Hamming code is dependent on the sensitivity of the FPGA model and the architecture of the protected component. For a fair comparison between the techniques and considering that Hamming code is dealing with upsets in the same components, the failure rate chosen follows the previous techniques with a 10-day MTBF to ensure that the results represent a scenario where the incidence of upsets is greater than stipulated for adders and multipliers in TMR and Scrubbing. When the decoder checks the parity and input bits and does not detect any changes, the value is passed to the output normally, but if the values do not match, the decoder identifies the position of the upset and restores the correct bit value. As well as TMR, we used formulas to classify the states: operational, indicating that the output bits of the decoder coincide with the input bits of the encoder; degraded, which represents the states that detected a fault and are in the process of correction; and failed for the interval between the occurrence of a SEU and the detection by the decoder.

Each property we used is in Table 1 where we show the formula in CSL extended with Rewards (R), related dependability attribute, and its informal meaning. Note that

since only in Scrubbing exists the possibility of a fault occurring and not being detected, only this technique had its safety analyzed. We considered that in both TMR and Hamming code, the occurrence of a fault will always be detected by the voter and the decoder, respectively.

Table 1. Properties formalized in CSL extended with Rewards

Attribute	Formula	Meaning
Availability	$R\{$"*timeOperational*"$\}$=? [$C <= t$]	Accumulated time in the operational mode within the time interval $[0, t]$.
Availability	$R\{$"*timeDegraded*"$\}$=? [$C <= t$]	Accumulated time in the degraded mode within the time interval $[0, t]$.
Availability	$R\{$"*timeFailure*"$\}$=? [$C <= t$]	Accumulated time in the failure mode within the time interval $[0, t]$.
Availability	S=? [*fail*]	The long-run probability of the system fails.
Reliability	P=? [$G[0, t]$ *oper*|*degrade*]	The probability of the system being operational or degraded in the first t days.
Safety	P=? [$G[0, t]$ *oper*|*degrade*|$fail_{safe}$]	The probability of the system being operational, degraded or failed safe, in the first t days.

4. Results and Discussion

This section presents the results obtained with CTMC models for SEU mitigation techniques discussed in this paper and a comparative analysis.

4.1. Availability Analysis

For this analysis, we considered a 90-day mission time to analyze how long each SEU mitigation technique would be in an operational, degraded, or failure state. Figure 2 shows the rewards results obtained with TMR where approximately half of the mission time in failed state demonstrates that, a mitigation technique implemented individually without a correction technique capable of restoring the functioning of the degraded components, can cause problems in a short period of time.

By analyzing the same properties for the Hamming code model, Figure 2 also illustrates a better result even if we have a higher failure rate than the one used in TMR. With more than 80% of the total mission time in operational state, Hamming code becomes an interesting correction technique for systems with low incidence of MBUs. One of the disadvantages of Hamming code is that if the number of protected bits is too large and several memory components implement the encoder and decoder blocks, the impact on FPGA's performance can be very large. An option to work around this problem is to implement TMR and Hamming code together, so the correction will only be performed in the case of upsets accumulations.

Rewards results obtained with Scrubbing using 10 adders and 10 multipliers were compared directly with the results presented in [Hoque 2016] using 2 adders and 2 multipliers. Figure 3 shows that for a 1 day interval, the results presented by the model with less components can keep the system in an operational state for a longer time, while the configuration with 10 adders and 10 multipliers spends more time degraded than operational but remains functional almost 100% of the time. The coverage rate used in both models was 99%.

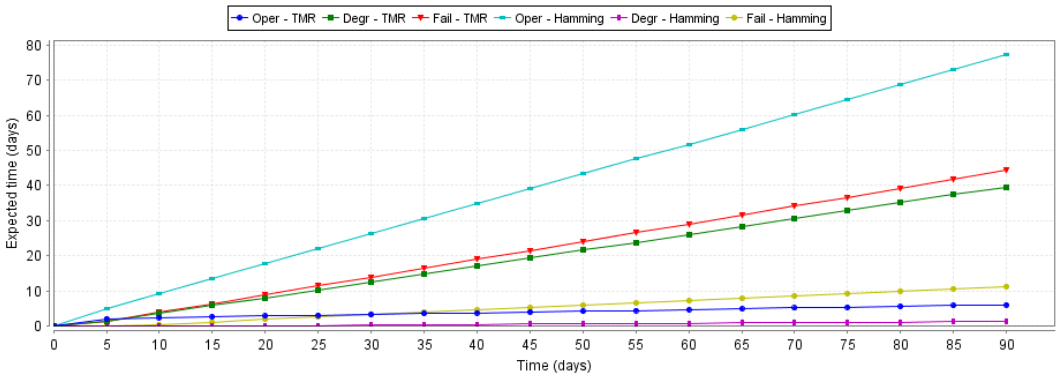


Figure 2. TMR and Hamming code's rewards analysis: 90-day mission.

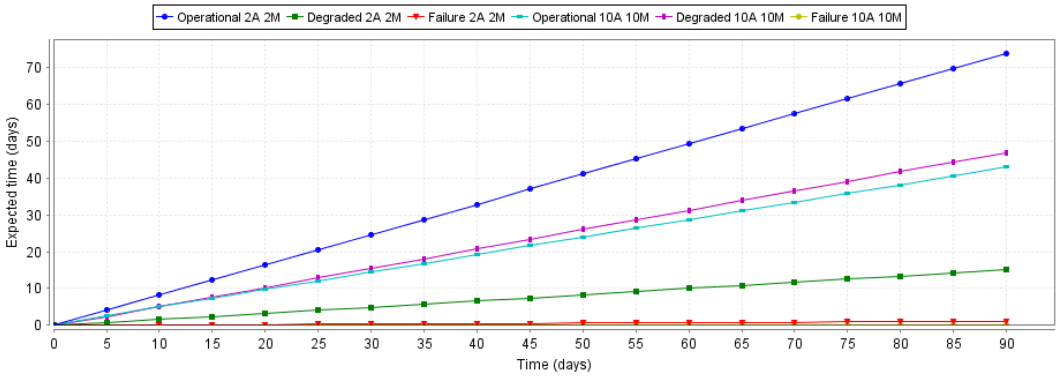


Figure 3. Rewards analysis, 1-day Scrubbing interval: 2A 2M = presented in [Hoque 2016]; 10A 10M = our contribution.

The situation is reversed with a Scrubbing interval of 9 days, even degraded, the configuration of 10 adders and 10 multipliers can keep the system running for much of the mission time, while the time in failure state presented by the other configuration goes up and may become unacceptable for systems that require high reliability, as shown in Figure 4.

With these results, it is possible to infer that in models with a smaller area, i.e. with less components susceptible to upsets, a very large Scrubbing interval can become a problem because the upsets will accumulate and it is likely that no operational components will left over. With a larger number of components, the larger Scrubbing interval becomes acceptable, since even if the upsets accumulates, the operating components will be overloaded but will keep the system functional. The accumulation of upsets occurs when more than one component is reached within the same Scrubbing interval, and since there is no detection technique (Blind Scrubbing), the corrections will only occur at the end of the stipulated interval.

Regarding the other perspective of availability, the probability of long-run failure of TMR and Hamming code techniques, Table 2 shows the values obtained in the long run. Again, due to the lack of a technique that prevents the accumulation of upsets, TMR presents a high probability of failure.

For Scrubbing, the failure probability in the long-run directly depends on the cho-

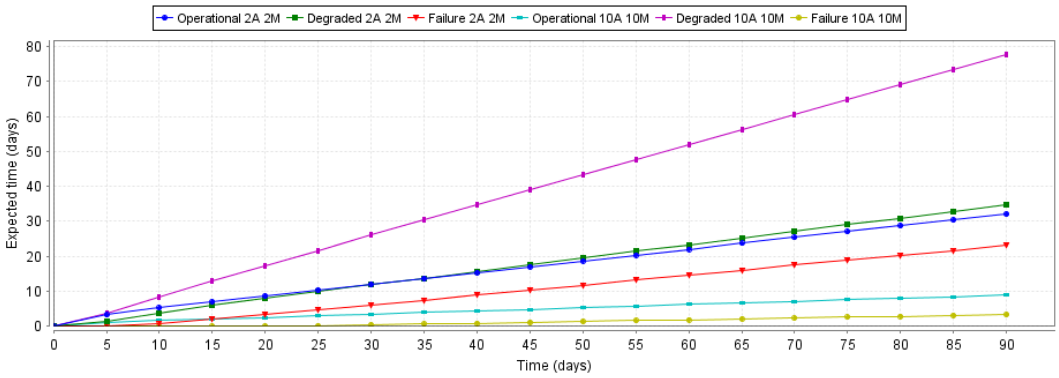


Figure 4. Rewards analysis, 9-day Scrubbing interval: 2A 2M = presented in [Hoque 2016]; 10A 10M = our contribution.

Table 2. TMR and Hamming code failure probability in the long-run.

SEU mitigation technique	Failure probability
TMR	50.5%
Hamming Code	13.5%

sen interval, as shown in Figure 5. For a small interval, the results are very similar between the configurations, but as the interval grows, the amount of components becomes inversely proportional to the probability of failures. In other words, the more components in the system, the less chance that all will suffer an upset within an interval of Scrubbing.

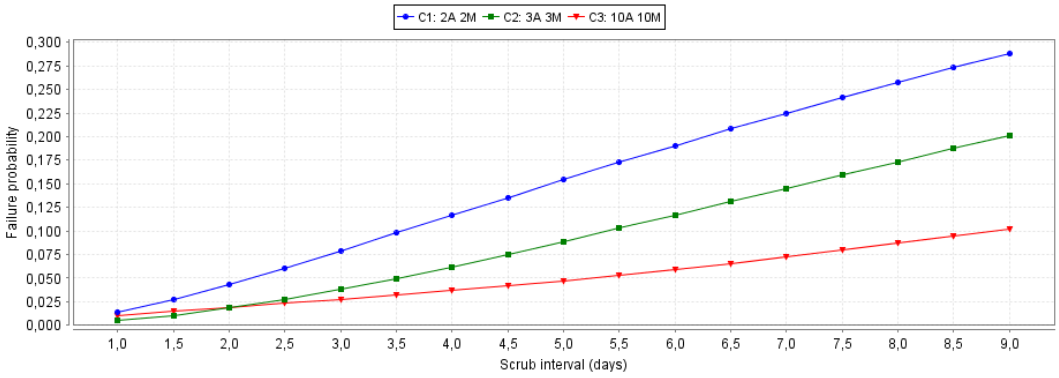


Figure 5. Scrubbing failure probability in the long-run: 2A 2M and 3A 3M = presented in [Hoque 2016]; 10A 10M = our contribution.

4.2. Reliability and Safety Analysis

In the reliability analysis, which is the probability of the system being operational or degraded in the first 90 days, the Hamming code presented a superior result compared to TMR: in half the mission time, TMR was already with zero chances of remaining without failures. Figure 6 shows the results obtained in the analysis and despite the superiority of the Hamming code, both results were lower than expected.

As before, Scrubbing reliability analysis was split into two intervals: 1 day and

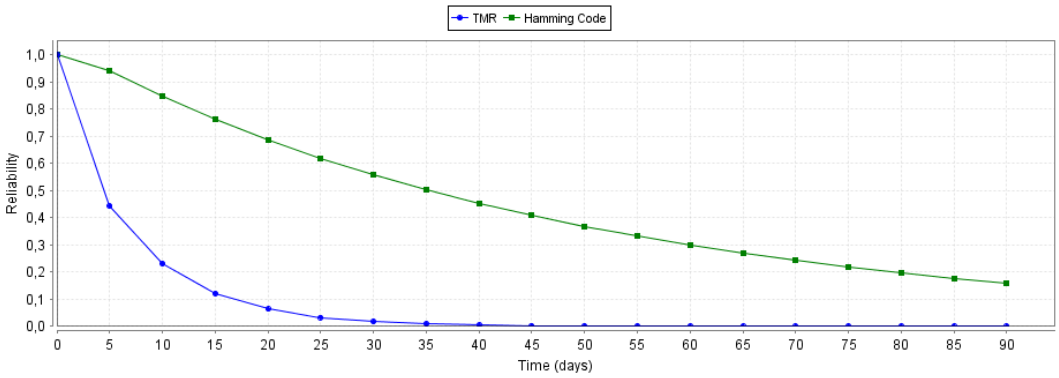


Figure 6. Reliability of TMR and Hamming code: 90-day mission.

9 days. From now on, the configuration presented in [Hoque 2016] with 2 adders and 2 multipliers will be called C1 while our configuration of 10 adders and 10 multipliers will be C2. Figure 7 represents the reliability of Scrubbing for C1 and C2 by varying the coverage percentage between 100% and 90% on a 90-day mission with a 1-day Scrubbing interval. The maximum reliability was obtained in C2 with 100% coverage, but it is possible to observe that the reduction of the coverage rate affected more drastically the reliability of C2 in both 95% and 90%. This shows that the more components in the system, more important is to ensure an efficient fault detection mechanism to keep the coverage always close to 100%.

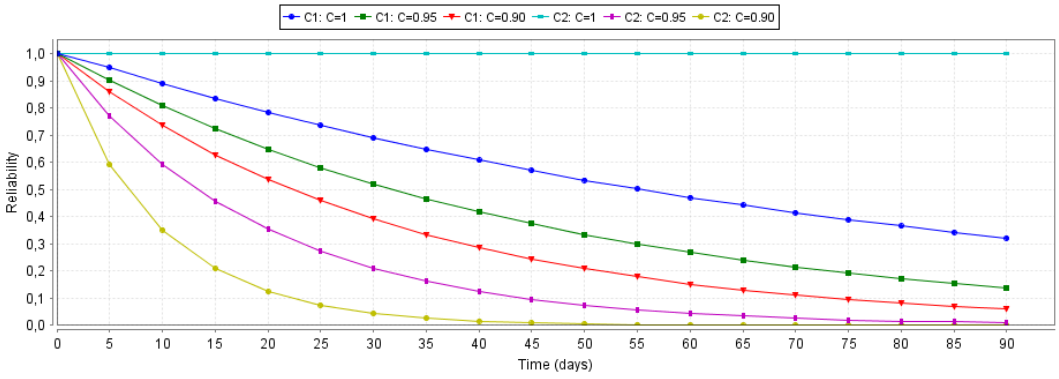


Figure 7. Scrubbing reliability, 1-day interval: 90-day mission.

Keeping the same properties but increasing the Scrubbing interval to 9 days, with the exception of C2 with 100% coverage, all other configurations lost their reliability until the end of the 90 days, as shown in Figure 8. This indicates how much the Scrubbing interval can influence the correct functioning of the system, especially in systems with fewer components.

The safety of a system that has a mitigation method such as Blind Scrubbing is related to the probability of detecting a fault since the fault occurred, i.e. with the coverage rate. This is because there is no fault detection technique in the Blind Scrubbing, so it runs at every preset time interval, even if no fault has occurred. Figures 9 and 10 show how decreasing coverage from 99% to 95% drastically decreases safety in both C1 and C2. This result indicates that for systems that value for safety, ensuring a high coverage rate

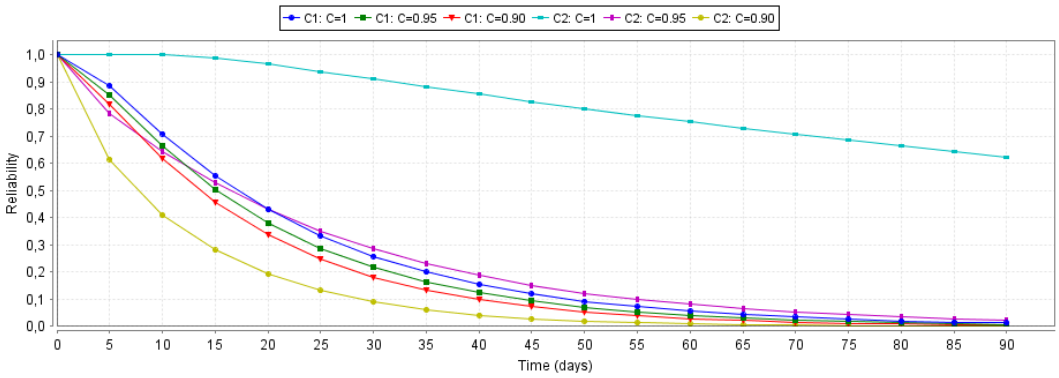


Figure 8. Scrubbing reliability, 9-day interval: 90-day mission.

will yield better results than changes in the Scrubbing interval. In addition, C1 was able to maintain safety greater than C2 at both coverage rates, showing that systems with fewer components decrease the chances of failures to go unnoticed and make the system unsafe.

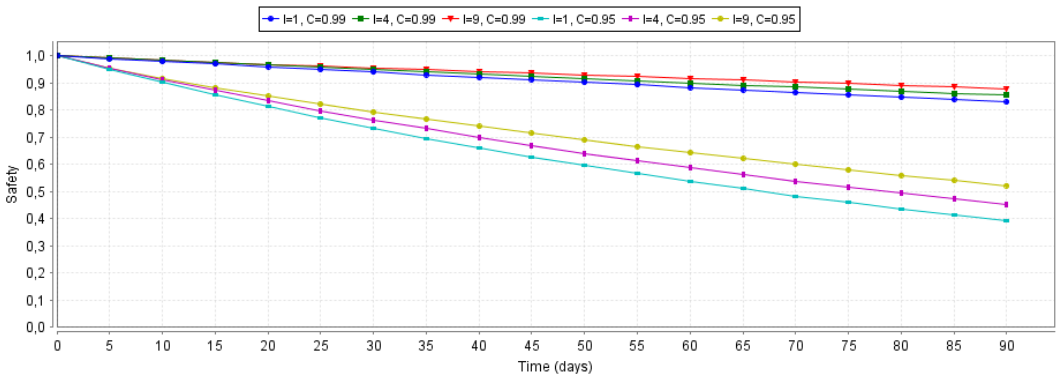


Figure 9. Scrubbing safety, 2 adders and 2 multipliers (C1): 90-day mission.

5. Related Work

SEU mitigation techniques in SRAM-based FPGAs used today have also undergone evolutions, and several works over the past few years have brought comparisons, combinations, and variations of these techniques to fit their needs. In [Morgan et al. 2007], the authors compare TMR with three additional techniques of SEU mitigation for FPGAs: quadded logic, state machine encoding and temporal redundancy. To simulate the harsh environment of space, they used a Xilinx Virtex fault-injection tool and analyzed the results based on reliability, area overload and reduced sensitivity of FPGAs. The conclusion was that these techniques presented a greater area overload and a smaller sensitivity reduction when compared with TMR, guaranteeing TMR a higher reliability. In [Shuler et al. 2009], TMR is also compared with other mitigation methods such as radiation-hardened flip flops and dual-rail logic. Considering area efficiency, TMR presented the best result with 95% efficiency.

In [Berg et al. 2008], the authors compare the effectiveness of two different forms of Scrubbing. The first form is internal, implemented by Xilinx, and the second one is external, but without a frame by frame readback and no method of detection of upsets.

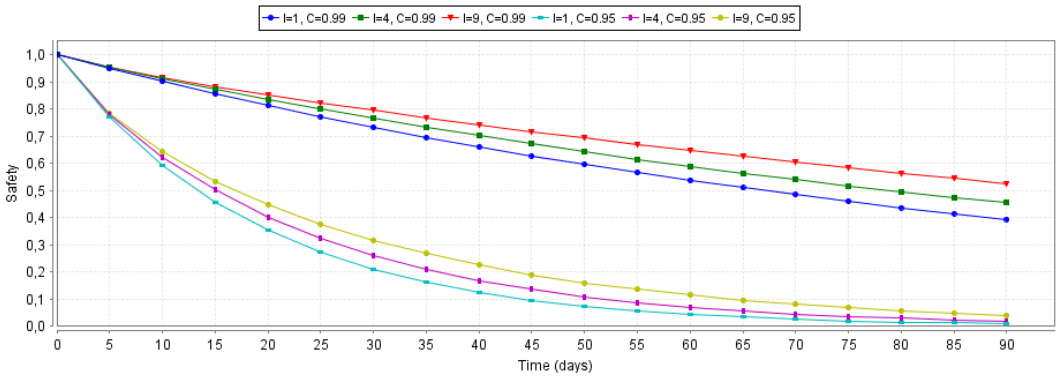


Figure 10. Scrubbing safety, 10 adders and 10 multipliers (C2): 90-day mission.

Results showed that even the internal scrubber being more complete, its efficiency was not superior to the external scrubber, being little ahead of an FPGA with no Scrubbing technique.

In [Liu et al. 2012], the authors compare Hamming code with another ECC, the Difference-Set Cyclic Codes (DSCC). The objective was to analyze performance, cost design and failure rate. The results showed that although the DSCC is more suitable for the Xilinx FPGA architecture, in practice Hamming code showed a lower failure rate and a better energy consumption, resulting in a more feasible cost design. A direct comparison between the TMR and the Hamming code was presented in [Hentschke et al. 2002] analyzing the impact on performance and area. As expected, the results showed advantages and disadvantages in each of the techniques. The TMR significantly increases the area of memory cells but performs better because it has a lower delay while Hamming code has only a small increase in memory cells, but depending on the amount of bits it must protect, it can add a significant delay. In our analysis, TMR had a lower performance.

All previous comparative analyzes have been conducted after implementing the techniques in FPGAs and simulate upsets with fault-injection tools, but this type of approach can be costly, since it presents the results only at the end of the simulations. On the other hand, stochastic models can be used in initial stages of the project and possibly help traditional approaches achieve better results in a complementary manner. Probabilistic analyzes via Formal Verification methods are still scarcely found in the literature, but in [Hoque et al. 2014] and [Hoque 2016] the authors have presented case studies using Probabilistic Model Checking for dependability analysis of SEU techniques. Our work is different from theirs in several ways, starting with the individually analyzed techniques rather than a single model combining two techniques. Moreover, our TMR not only deals with spare components but also includes the input and output bits and the voter. In Scrubbing, a different scenario was addressed, where 10 adders and 10 multipliers showed significant results. In addition, Hamming code was another technique analyzed and showed promising results.

6. Conclusions

In this work, we presented a comparison between three well-known techniques of mitigation of SEU in SRAM-based FPGAs. By using Probabilistic Model Checking, we claim that the benefits are high because we are able to get earlier results of these tech-

niques without the cost and effort spent designing and implementing in FPGA and using fault-injection tools. Eventually, our analysis may not replace such implementation-based approaches, but our results may complement such strategies with an additional probabilistic perspective. We relied on CTMC models to verify possible failures and to analyze the availability, reliability and safety of the techniques within a radiation environment with the characteristics described in this work. The properties formalized in CSL were automatically verified by PRISM [Kwiatkowska et al. 2009].

Results obtained in the availability analysis showed that Hamming code was the technique that stayed the longest in operational state even if we considered the worst-case failure rate. Scrubbing comes in the sequence with interval of 1 day and with interval of 9 days. Comparing the results of the two Scrubbing configurations, smaller systems (less amount of components) can spend less time in failure if the Scrubbing interval is as small as possible. For larger systems, like our configuration with 10 adders and 10 multipliers, larger intervals make the system degraded, but still functional. The worst result was TMR which spent half the total mission time failing because it could not fix the upsets, generating a buildup that makes the system inoperable. One of the alternatives is to combine TMR with some correction technique such as Scrubbing or Hamming code.

In the reliability analysis, TMR also presented the worst result but Hamming code was worse than presented in our availability analysis. For Scrubbing, the greatest influence on reliability is the length of the interval between corrections, especially on smaller systems. Thus, if the application focus is to ensure reliability, the ability to detect faults (coverage rate) can be relaxed, but the Scrubbing interval should be as small as possible. The safety analysis presented the inverse result, since the coverage rate was more relevant in the results than the variation in the Scrubbing interval in both configurations.

We need to realize about the effectiveness of this Formal Verification and probabilistic analysis. Therefore, our next efforts will focus on implementing these SEU mitigation techniques in FPGA and compare whether the results of our analysis meet the practical results and, if the conclusion is positive, in the future one may rely on such probability comparison rather than really implementing the techniques in FPGAs.

References

Baier, C., Katoen, J.-P., and Larsen, K. G. (2008). *Principles of model checking*. MIT press.

Berg, M., Poivey, C., Petrick, D., Espinosa, D., Lesea, A., LaBel, K. A., Friendlich, M., Kim, H., and Phan, A. (2008). Effectiveness of internal versus external seu scrubbing mitigation strategies in a xilinx fpga: Design, test, and analysis. *IEEE Transactions on Nuclear Science*, 55(4):2259–2266.

Dutta, A. and Touba, N. A. (2007). Multiple bit upset tolerant memory using a selective cycle avoidance based sec-ded-daec code. In *VLSI Test Symposium, 2007. 25th IEEE*, pages 349–354. IEEE.

Ejlali, A., Al-Hashimi, B. M., Schmitz, M. T., Rosinger, P., and Miremadi, S. G. (2006). Combined time and information redundancy for seu-tolerance in energy-efficient real-time systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(4):323–335.

Heiner, J., Sellers, B., Wirthlin, M., and Kalb, J. (2009). Fpga partial reconfiguration via configuration scrubbing. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 99–104. IEEE.

Hentschke, R., Marques, F., Lima, F., Carro, L., Susin, A., and Reis, R. (2002). Analyzing area and performance penalty of protecting different digital modules with hamming code and triple modular redundancy. In *Proceedings. 15th Symposium on Integrated Circuits and Systems Design*, pages 95–100. IEEE.

Hoque, K. A. (2016). *Early Dependability Analysis of FPGA-Based Space Applications Using Formal Verification*. PhD thesis, Concordia University Montréal, Québec, Canada.

Hoque, K. A., Mohamed, O. A., Savaria, Y., and Thibeault, C. (2014). Probabilistic model checking based dal analysis to optimize a combined tmr-blind-scrubbing mitigation technique for fpga-based aerospace applications. In *2014 Twelfth ACM/IEEE Conference on Formal Methods and Models for Codesign*, pages 175–184. IEEE.

Kastensmidt, F. L. (2007). See mitigation strategies for digital circuit design applicable to asic and fpgas. In *IEEE NSREC Short Course*.

Kastensmidt, F. L., Sterpone, L., Carro, L., and Reorda, M. S. (2005). On the optimal design of triple modular redundancy logic for sram-based fpgas. In *Proceedings of the conference on Design, Automation and Test in Europe-Volume 2*, pages 1290–1295. IEEE Computer Society.

Kumar, U. and Umashankar, B. (2007). Improved hamming code for error detection and correction. In *Wireless Pervasive Computing, 2007. ISWPC'07. 2nd International Symposium on*. IEEE.

Kwiatkowska, M., Norman, G., and Parker, D. (2009). Prism: probabilistic model checking for performance and reliability analysis. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):40–45.

Liu, S., Sorrenti, G., Reviriego, P., Casini, F., Maestro, J., Alderighi, M., and Mecha, H. (2012). Comparison of the susceptibility to soft errors of sram-based fpga error correction codes implementations. *IEEE Transactions on Nuclear Science*, 59(3):619–624.

Morgan, K. S., McMurtrey, D. L., Pratt, B. H., and Wirthlin, M. J. (2007). A comparison of tmr with alternative fault-tolerant design techniques for fpgas. *IEEE transactions on nuclear science*, 54(6):2065–2072.

Rollins, N., Wirthlin, M., Caffrey, M., and Graham, P. (2003). Evaluating tmr techniques in the presence of single event upsets. In *Proceedings of the 6th Annual International Conference on Military and Aerospace Programmable Logic Devices*, page P63.

Sari, A., Psarakis, M., and Gizopoulos, D. (2013). Combining checkpointing and scrubbing in fpga-based real-time systems. In *VLSI Test Symposium (VTS), 2013 IEEE 31st*, pages 1–6. IEEE.

Shuler, R. L., Bhuva, B. L., O'Neill, P. M., Gambles, J. W., and Rezgui, S. (2009). Comparison of dual-rail and tmr logic cost effectiveness and suitability for fpgas with reconfigurable seu tolerance. *IEEE Transactions on Nuclear Science*, 56(1):214–219.

Arquitetura de hardware de uma estação remota de entrada analógica em conformidade com a IEC 61508

Taisy S. Weber, Sérgio Cechin, João de Moraes, João C. Netto

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{taisy,cechin,joao.moraes,netto}@inf.ufrgs.br

Abstract*. The paper introduces the initial step of designing an analog input module for a remote input and output station used in safety related systems. The module was developed according to the IEC 61508 standard aiming at a high level of safety integrity (SIL 3). To achieve certification of compliance with the standard, the initial step is the development of the module architecture and its safety specification. Only after the approval of this step by the certification agency, the implementation of hardware and software can be initiated. The article presents the fault-tolerant hardware architecture of the input module.*

Resumo*. O artigo apresenta o resultado da etapa inicial do projeto de um módulo de entrada analógica para uma estação remota de entrada e saída usada em sistemas de segurança. O projeto foi desenvolvido de acordo com a norma IEC 61508 visando alto nível de integridade (SIL 3). Para almejar a certificação de conformidade com a norma, a etapa inicial consiste no desenvolvimento da arquitetura do módulo e a sua especificação de segurança. Apenas após a aprovação dessa etapa pela agência certificadora, a prototipação do hardware e software do módulo pode ser iniciada. O foco do artigo é a arquitetura de hardware tolerante a falhas do módulo de entrada.*

1 Introdução

Sistemas de segurança podem ser usados para interromper o fluxo de combustíveis e isolar equipamentos elétricos (através de atuadores) após detectar (através de sensores) alta pressão, alta temperatura, fogo ou vazamento de gás. Um exemplo são os sistemas de proteção contra alta pressão usados para prevenir acidentes em dutos de óleo ou gás (Lundteigen, Rausand, and Utne 2009).

O projeto de sistemas de segurança deve obedecer à rígida regulamentação. Em vários setores industriais, como extração e refino de petróleo, produção e distribuição de energia, e controle de máquinas e processos, regulamentações são baseadas na IEC 61508 (Gall and Wen 2010), que a partir do ano 2000, passou a permitir o uso de componentes programáveis no projeto de tais sistemas (Bell 2011).

Componentes programáveis estão sujeitos a falhas de hardware, interferências externas e erros de programação. Para garantir a integridade da função de segurança, o projeto do sistema deve ser conduzido obedecendo a rigorosos requisitos técnicos que impõem a escolha de mecanismos e estratégias adequados para manter a taxa de defeitos dentro dos limites adequados. A rigidez das normas de segurança limita as

opções de projeto, mas permite alcançar a integridade de segurança necessária, além de possibilitar certificação de integridade de segurança.

Um dos equipamentos de um sistema de segurança distribuído é a estação remota, formada por módulos de entrada e saída. Um módulo de entrada é responsável pela aquisição das informações dos sensores e o envio destas para um controlador programável (CP). Um módulo de saída recebe informações do CP e faz com que sejam aplicadas aos atuadores. A comunicação das estações remotas com o CP em um sistema de segurança é realizada através de um protocolo de comunicação seguro (Neumann 2007), sobre um *black channel*. A pilha do protocolo de comunicação deve ser implementada tanto na estação remota quanto no controlador programável, devendo também ser certificada quanto à integridade de segurança por órgão certificador competente.

Os módulos de entrada nas estações remotas podem ser digitais ou analógicos dependendo do tipo de sinal fornecido pelos sensores. Neste artigo, introduzimos o projeto do módulo de entrada SAIM (*Safety Analog Input Module*), que recebe oito sinais analógicos e os processa digitalmente antes de enviar ao controlador programável. SAIM está sendo projetado para ser usado com CPs seguros da Serie Nexto Safety, da empresa Altus SA, parceira no projeto. Depois de certificado, o produto resultante será aplicado para executar funções de segurança na indústria de máquinas e processos.

O projeto de um módulo de entrada analógica parece não representar grandes desafios. Mas o módulo não se restringe apenas a converter sinal analógico para digital. Ele deve executar os protocolos necessários para comunicação remota com o CP, detectar e corrigir erros, reportar os erros detectados e manter as máquinas ou processos sendo controlados em um estado seguro, caso os erros não possam ser corrigidos. Para atingir o nível de integridade de segurança almejado, o módulo deve ser projetado aplicando técnicas de tolerância a falhas e obedecendo todos os requisitos de segurança estabelecidos pela IEC 61508.

Para executar suas várias funções, SAIM emprega microcontroladores (MCU). Várias arquiteturas tolerantes a falhas podem ser implementadas com esses componentes, com diferentes níveis de redundância. Para SAIM foi escolhida uma arquitetura de 2 canais com diagnóstico cruzado de erros. Cada canal tem sua MCU. Uma terceira MCU executa funções não relacionadas à segurança, ou seja, funções que não precisam ser certificadas.

Esse artigo apresenta a arquitetura de hardware do módulo remoto de entrada analógica, SAIM, construído em conformidade a norma IEC 61508 como parte do projeto de transferência tecnológica SDCD financiado pelo BNDES, cuja empresa parceira, Altus SA, é fabricante de Controladores Programáveis. O objetivo do artigo é apresentar a experiência do grupo de pesquisadores do laboratório LAIS, da UFRGS, no desenvolvimento de sistemas de segurança visando certificação.

Existem vários fabricantes internacionais com equipamentos semelhantes certificados. Não é do nosso conhecimento a existência de produtos nacionais neste nicho de mercado. A empresa Altus é pioneira nacional no projeto de sistemas de segurança e, através de cooperação com universidades, desenvolve tecnologia para atuar como fornecedora de equipamentos certificados, não apenas localmente, mas como competidora internacional.

SAIM teve seu projeto de arquitetura concluído e se encontra na fase de aprovação da especificação dos requisitos de segurança por uma agência certificadora. O resultado da primeira fase de projeto é o diagrama de blocos, com a descrição de todos os componentes, e o documento que contém a especificação dos requisitos de segurança.

O artigo está organizado como segue: inicialmente são apresentados os fundamentos da norma IEC 61508. Em seguida são apresentados os conceitos relacionados a uma estação remota de E/S e os componentes básicos de SAIM. Depois é mostrado como são tratados os sinais analógicos de entrada e finalmente o artigo é concluído.

2 Norma para segurança funcional IEC 61508

A IEC 61508 (Bell 2006), foi a primeira norma a permitir o emprego de componentes programáveis para a construção de sistemas de segurança. Ela estabelece técnicas e métodos para evitar e controlar falhas aleatórias de hardware e erros de projeto e de software.

Um sistema de segurança desenvolvido criteriosamente segundo a norma pode almejar certificação para um determinado nível de integridade de segurança, SIL. Os níveis variam de 1 a 4 (Joannou and Wassyng 2014). Sistemas com menor probabilidade de apresentar defeitos são classificados em SIL 4. SAIM almeja alcançar certificação para SIL 3, nível de integridade de segurança usual na área de controle de máquinas e processos.

A norma visa aumentar a garantia de que a função de segurança esteja livre de falhas que possam conduzir a defeitos perigosos (Jin, Lundteigen, and Rausand 2011). Dessa forma, é natural que obrigue o emprego tolerância a falhas (Lundteigen and Rausand 2009). Adicionalmente, caso o sistema não possa ser mantido operacional diante da ocorrência de falhas, suas saídas devem ser comutadas para um estado seguro, geralmente parando a máquina ou processo que está sendo controlado.

A IEC 61508 foi publicada originalmente em meados do ano 2000. Em 2010 a norma foi revisada consolidando 10 anos de experiência na sua aplicação (Bell 2011). No meio industrial brasileiro, é crescente o interesse pelo desenvolvimento de produtos em conformidade com essa norma devido à regulamentação de setores como energia, petróleo e transporte público.

2.1 Certificação de segurança

O objetivo de uma certificação de segurança é prover a garantia, reconhecida pelo órgão regulador, que o sistema foi considerado seguro pelo órgão certificador (Panesar-Walawege et al. 2010).

A certificação de conformidade com a IEC 61508 é dada a um produto específico, e envolve análise do processo de desenvolvimento e produção para aquele produto e também da competência da equipe de desenvolvimento.

O conjunto de técnicas e medidas adotadas em todas as fases do ciclo de vida deve convencer o órgão certificador que o projeto apresenta o nível de integridade de segurança almejado.

O produto é certificado, principalmente, através da documentação gerada, mas a norma não estabelece o formato dos documentos. Todas as fases do ciclo de vida do produto devem ser auditadas. Os certificadores devem ter acesso a toda documentação e pessoas envolvidas no desenvolvimento. As ferramentas também devem ser avaliadas, assim como o todo o material usado durante o desenvolvimento está sujeito à avaliação.

A IEC 61508 é uma norma extensa, com centenas de páginas e centenas de medidas prescritivas baseadas em uma vasta experiência prática, pesquisas e discussões. Como consequência, desenvolver um projeto em conformidade com a norma e obter certificação é uma tarefa árdua (Bilich and Hu 2009).

2.2 Esquema básico de um sistema de segurança

Em sistemas de segurança construídos seguindo a IEC 61508, o hardware e o software, assim como a aplicação que implementa a função de segurança, devem ser construídos e validados usando estratégias e técnicas da área de tolerância a falhas e engenharia de software embarcado.

A Figura 1 mostra um sistema formado por estações remotas de entrada e saída, um componente eletrônico programável central (no caso um CP), e a comunicação entre eles. As estações remotas podem conter vários módulos de E/S. Cada módulo tem sua lógica de leitura e escrita de dados, comunicação e detecção de erros implementada por seus próprios microcontroladores (MCU). Um sistema de segurança pode apresentar alguns módulos E/S seguros (em amarelo na Figura 1) e outros módulos de E/S convencionais, mas o CP deve ser seguro.

Na área de segurança, é necessária uma alta probabilidade de uma resposta segura do sistema a defeitos tanto do processo ou equipamento sendo controlado como do próprio sistema de segurança. Entretanto, estes sistemas podem introduzir novos defeitos. Se a taxa de falsos alarmes ou ativações indevidas for muito alta (Lundteigen, Rausand, and Utne 2009), os operadores podem perder a confiança no sistema.

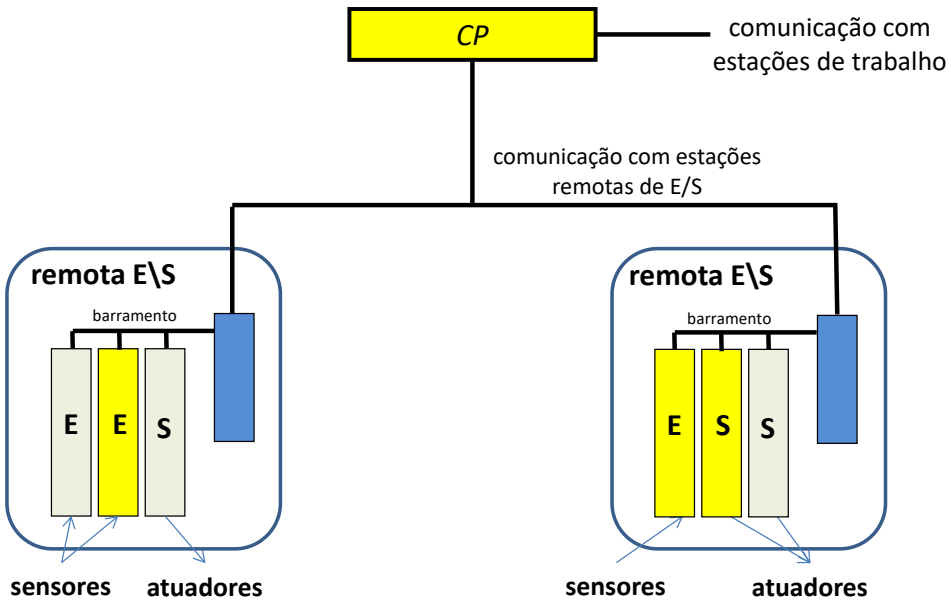


Figura 1 – Sistema distribuído com módulos de segurança

2.3 Comunicação segura e *black channel*

A IEC 61508 define que os sistemas de comunicação não podem ser responsáveis por mais do que 1% dos defeitos provocados por falhas não cobertas pelos mecanismos de detecção e diagnóstico.

A primeira versão da IEC 61508 estabelecia que todos os subsistemas de comunicação deveriam ser certificados. Essa exigência impunha sérias limitações ao desenvolvimento de estações remotas, pois sistemas de segurança distribuídos dependem de comunicação. Era impossível usar protocolos já consolidados na área de controle e automação, pois tais protocolos não são certificados pela IEC 61508. Uma possibilidade seria desenvolver e certificar tanto a pilha dos protocolos já em uso como os dispositivos de rede. Mas essa possibilidade tem um custo proibitivo e poucos fornecedores interessados.

A melhor solução veio com o conceito de *black channel*: toda a pilha de protocolos e o meio físico que transporta o sinal elétrico, magnético ou ótico são considerados um canal opaco. Sobre essa camada de comunicação insegura, cujos detalhes de implementação não precisam ser conhecidos, é implementado um protocolo seguro. O protocolo seguro é uma camada extra que reduz a probabilidade de erro na comunicação através de diversos mecanismos para detecção e correção de erros. Basta que a implementação dessa nova camada seja certificada como segura e toda a comunicação será considerada segura.

O conceito de *black channel* é extremamente útil, pois torna possível o uso de uma pilha convencional de protocolos de comunicação sobre a qual basta adicionar um protocolo seguro implementado em software. Esse conceito foi responsável pelo aparecimento de vários protocolos seguros (Neumann, 2007).

Perfis de protocolos seguros foram normatizados pela IEC 61784-3 (Bell, 2011). Uma dada implementação em software destes protocolos é passível de ser certificada até o nível SIL3. No projeto, foi escolhido o PROFIsafe, um dos protocolos padronizados pela IEC 61784-3, principalmente devido a sua popularidade, qualidade da documentação e a pré-aprovação para SIL 3. A pré-aprovação da especificação para SIL 3 não implica em garantias que a implementação em software para um dado hardware será certificada, mas facilita o processo de certificação do sistema de segurança distribuído que emprega o protocolo.

Um protocolo seguro forma uma camada acima de outro protocolo de mais baixo nível que reside no *black channel*. O PROFIsafe (Malik and Mühlfeld 2003), que é um protocolo seguro, opera sobre o protocolo PROFIBUS, um protocolo de comunicação popular no setor industrial, não certificado quanto a segurança.

3 O módulo de entrada analógica SAIM

SAIM é um módulo de entrada analógica que faz parte de uma estação remota. A função executada por SAIM é a de receber e executar os comandos enviados pelo CP, obter os dados dos sensores, processá-los e enviá-los para o CP. Para ser usado em sistemas de segurança, todo o projeto do módulo deve passar por um processo de certificação de conformidade com a IEC 61508. Esse processo inicia junto com a avaliação da especificação de segurança do módulo e sua arquitetura de hardware por uma agência certificadora.

3.1 Módulo de entrada SAIM em um sistema de segurança

A estação remota, mostrada na metade direita da Figura 2, é formada por uma cabeça PROFIBUS, uma interface para comunicação interna entre os módulos de E\S de uma mesma estação e um ou mais módulos de entrada e saída.

A comunicação segura da estação remota com o CP ocorre através de um canal seguro sobre PROFIBUS, que transporta os quadros PROFIsafe. O *black channel* engloba as cabeças PROFIBUS, que se encontram nos dois lados da comunicação, e a comunicação pelo barramento EtherCat entre os diversos módulos de E/S da estação remota.

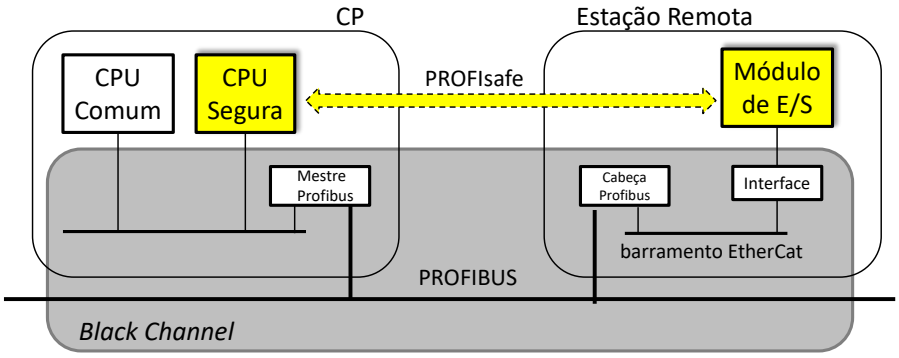


Figura 2 – *Black channel* entre estação remota e o controlador

EtherCat (Rostan, Stubbs, and Dzilno 2010) é um barramento muito usado em ambiente industrial, e é construído sobre Ethernet (Lin and Pearson 2013).

O CP, como pode ser observado na Figura 2, opera com duas unidades de processamento: um processador convencional, a CPU Comum, não relacionada à segurança; e uma CPU Segura, que processa o protocolo de comunicação segura e executa a função de segurança.

A estação remota deve contribuir com apenas uma baixa percentagem de erros residuais. No máximo 5% dos defeitos de todo o sistema de segurança podem ter sido gerados por erros na estação remota. Para isso, os módulos que executam o protocolo seguro devem ter incorporados mecanismos para detecção de erros e diagnósticos com alta cobertura de falhas. Assim, além de executar o protocolo de comunicação segura, o módulo deve conduzir todos os testes necessários para alcançar a cobertura de diagnóstico e de detecção de erros especificada para o nível de SIL desejado.

O hardware da estação remota é separado em dois grupos de circuitos: um dentro e outro fora do *black channel*. O primeiro grupo pode ser implementado usando hardware convencional. Assim, pode-se usar, sem certificação, componentes comuns que implementam os barramentos e seus protocolos correspondentes, PROFIBUS e EtherCat.

O grupo de circuitos fora do *black channel*, que na estação remota corresponde aos módulos seguros de E/S, deve ser construído seguindo as restrições da IEC 61508. Para isso, deve-se escolher uma arquitetura tolerante a falhas com grau de redundância adequado e implementar a cobertura de falhas exigida para alcançar o SIL desejado.

3.2 Comunicação via PROFIsafe

SAIM recebe comandos do CP remotamente via comunicação segura PROFIsafe. Planejamos usar um código fonte certificado de uma pilha PROFIsafe, comercializado pela Siemens, denominado "*PROFIsafe driver for F-Slaves*". O código fonte escolhido está em conformidade com a IEC 61784-3-3. A implementação já certificada do protocolo vai reduzir a sobrecarga de desenvolvimento e de certificação do módulo de entrada.

SAIM opera executando comandos do CP recebidos em um quadro PROFIsafe. Esse quadro passa pelo PROFIBUS e chega a SAIM via a cabeça PROFIBUS e o barramento interno da estação, que opera com EtherCat. Os dados analógicos obtidos dos sensores são convertidos para digital, processados, e montados em um quadro PROFIsafe. Este quadro, através de uma conexão EtherCat com o barramento, é passado para a cabeça PROFIBUS. A cabeça PROFIBUS é o dispositivo que permite comunicação remota com um mestre PROFIBUS residente no CP.

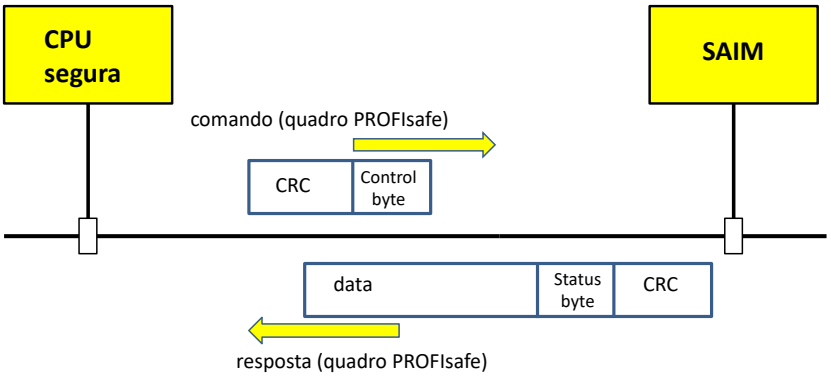


Figura 3 – Comunicação segura entre SAIM e CP

Os seguintes quadros PROFIsafe são manipulados em SAIM (Figura 3):

- Comando: consiste em 4 bytes enviados pela CPU Segura para SAIM, formado por 1 byte de controle e 3 bytes CRC.
- Resposta: consiste de 20 bytes enviados do SAIM e lidos pela CPU Segura. 16 bytes de dados são gerados no SAIM a partir do valor processado das 8 entradas analógicas; adicionalmente são computados um byte de status e 3 bytes CRC.

4 Diagrama de blocos da arquitetura de hardware

A Figura 4 mostra o diagrama de blocos da arquitetura da estação remota. A cabeça PROFIBUS, o barramento e o processamento da interface EtherCat pertencem ao *black channel*. Os dois processadores redundantes, SMCU1 e SMCU2, que processam os comandos vindos do CP e os sinais recebidos dos terminais de entrada de dados, devem garantir o nível de integridade de segurança necessária.

Várias opções de microcontroladores especialmente desenvolvidos para aplicações de segurança estão disponíveis no mercado. Nessa classe podem ser encontradas as famílias Hercules, da Texas, e a PXS da Freescale.

Os processadores da família Hercules, do tipo ARM com dois núcleos, foram escolhidos para SAIM, porque a empresa parceira Altus, já possui experiência na sua

aplicação em produtos industriais. Além de apresentarem alta cobertura de falhas, oferecem toda a documentação necessária para possibilitar um projeto de hardware seguro. O fabricante fornece informações que vão deste as taxas de defeito do componente até os métodos usados para reduzir as falhas sistemáticas de projeto e para detectar falhas aleatórias.

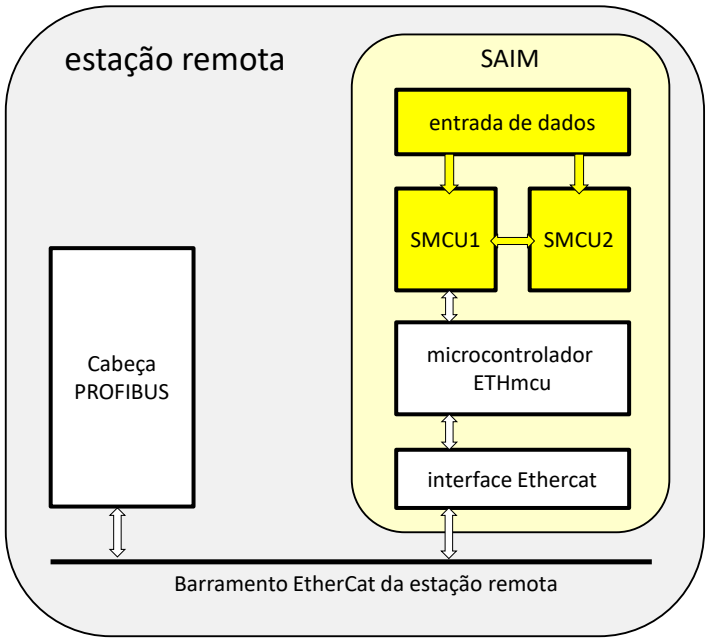


Figura 4 – Diagrama geral da estação remota com módulo SAIM

O processador ETHmcu é um microcontrolador ARM convencional.

4.1 Processamento da pilha de comunicação EtherCat: ETHmcu

A maioria das tarefas de software não relacionadas à segurança são executadas pelo ETHmcu, incluindo algumas tarefas complexas como a execução da pilha de comunicação EtherCAT. O ETHmcu não executa nenhuma tarefa relacionada à segurança, fazendo parte do *black channel.*

As tarefas básicas executadas pelo ETHmcu são:

- Processamento da pilha EtherCAT para comunicação com a cabeça PROFIBUS através do barramento.
- Verificação de endereço de barramento para comunicação EtherCAT com a cabeça PROFIBUS.
- Intermediação da comunicação entre o microcontrolador seguro SMCU1 e a cabeça PROFIBUS.
- Gestão da interface com o usuário.

O protocolo EtherCat, usado no barramento, pode ser trocado por outro protocolo de comunicação baseado em Ethernet industrial, apenas alterando o software do microcontrolador ETHmcu e desenvolvendo um hardware para interface com o novo barramento. Esta alteração pode ser realizada sem necessidade de reprojetar e certificar a parte segura do módulo SAIM.

O ETHmcu intermedia a comunicação PROFIsafe entre a CPU Segura no CP e os processadores SMCU1 e SMCU2. A troca das informações seguras é garantida pelo mestre PROFIsafe, executado na CPU Segura do CP, e o escravo PROFIsafe executado no módulo de entrada pelos processadores SMCU1 e SMCU2. O ETHmcu também intermedia a parametrização enviada da CPU Comum do CP para o par SMCU1/SMCU2, usando a parametrização padrão PROFIBUS. A verificação CRC é executada no par SMCU1/SMCU2 após a recepção dos parâmetros.

Todas as tarefas relacionadas com a segurança são executadas pelo par redundante SMCU1/SMCU2.

4.2 Processamento relacionado à segurança: SMCU1 e SMCU2

Os microcontroladores SMCU1 e SMCU2 (Figura 5) estão configurados de forma redundante para obter tolerância a falhas de hardware. Eles são interligados através de um canal serial, para que possam trocar dados e realizar detecção de erros cruzada. Estão configurados como um par rodando em *lock step* (ou seja, com sincronização de relógios).

SMCU1 e SMCU2 são micros Hercules, da Texas. Cada Hercules tem dois núcleos. São usados dois microcontroladores, totalizando assim quatro núcleos para processamento seguro. Como os núcleos dentro de um MCU operam de forma redundante, é possível diagnosticar erros por comparação dos resultados. Adicionalmente, realizando verificação cruzada entre os dois MCUs, é possível alcançar o grau de tolerância a falhas e de cobertura de falhas exigido pela norma para SIL 3.

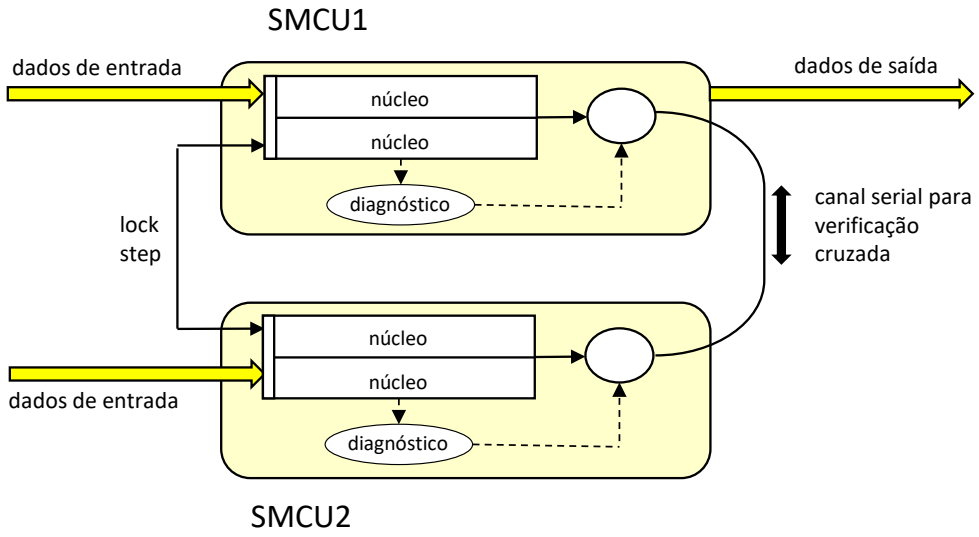


Figura 5 – Microcontroladores seguros em arquitetura redundante

O fato de termos quatro núcleos executando a mesma tarefa de forma redundante pode parecer um desperdício de capacidade de processamento. Entretanto para o nível de integridade de segurança almejada, SIL 3, tal redundância de hardware é plenamente justificada. Esta arquitetura tolerante a falhas simplifica as tarefas de detecção de erros e aumenta a cobertura de falhas sem a necessidade de exaustivos testes de diagnóstico por software sobre todos os componentes de hardware internos aos microcontroladores.

O software nestes microcontroladores redundantes é dividido em duas fases: inicialização e operacional. Durante a fase de inicialização, o software executa auto-teste em SMCU1 e SMCU2 e configura todos os registradores internos para executar as funcionalidades necessárias. A fase operacional é cíclica, com período de ciclo compatível com os tempos de resposta esperados na área de aplicação (1ms), nos quais são executadas as seguintes tarefas principais:

- Interface com o barramento através do ETHmcu (somente no SMCU1, que está conectado ao ETHmcu).
- Gestão da comunicação PROFIsafe com a CPU Segura no CP.
- Gestão da comunicação com a CPU Comum do CP (para diagnóstico e parametrização).
- Leitura e processamento de todas as 8 entradas analógicas do SAIM.
- Detecção, recuperação e sinalização de falhas.
- Diagnóstico.
- Sincronização e verificação cruzada entre os microcontroladores redundantes SMCU1 e SMCU2.

Decidimos que para SMCU1 e SMCU2 não seria usado qualquer tipo de sistema operacional de tempo real, tanto para reduzir a complexidade do software embarcado como para reduzir os custos de produção. Isto significa que todo o software de baixo nível necessário será desenvolvido no escopo do projeto e submetido à certificação.

4.3 Separação entre hardware de segurança e não relacionado à segurança

O hardware não relacionado à segurança é separado do hardware relacionada à segurança por circuitos para proteção contra sobretensão e isolamento na comunicação entre ETHmcu e SMCU1.

As tarefas não relacionadas à segurança executadas pelo ETHmcu poderiam ser executadas pelo SMCU1, mas a nossa escolha no projeto foi alocar um microcontrolador adicional (o ETHmcu) para essas tarefas. Esta escolha apresenta algumas vantagens importantes: diminui a complexidade do software no SMCU1 e, portanto, simplifica o cumprimento dos requisitos de segurança do software; e reduz o número de sinais de interface entre o SMCU1 e o barramento, reduzindo assim a complexidade da proteção contra sobretensão.

5 Tratamento dos sinais de entrada analógicos

Para SAIM poder ser considerado seguro, a entrada dos sinais analógicos também deve ser segura. Assim, cada uma das 8 entradas é conectada a um circuito de condicionamento de sinal diferente e a um conversor digital para analógico (ADC) diferente, formando uma arquitetura de canal duplo (Figura 6).

Cada circuito de condicionamento de sinal tem uma relação de tensão diferente. Além disso, cada conversor (ADC1 e ADC2) tem uma tensão de referência diferente, com a mesma relação de tensão. Isso resultará em ambos os ADCs retornando o mesmo valor binário. Podem ocorrer discrepâncias devido a ruído ou diferenças nos componentes. Portanto, uma pequena diferença pode ser tolerada. Se a diferença for maior, um erro deve ser sinalizado e o módulo deve ir para o estado seguro.

O valor de saída é sempre o valor médio entre as leituras dos dois ADCs. SMCU1 só pode ler ADC1, e SMCU2 só pode ler ADC2, portanto, os valores devem ser trocados e o valor médio é calculado por ambos os microcontroladores. Como cada microcontrolador cria todo o pacote PROFIsafe e o CRC é calculado em cada microcontrolador, qualquer erro durante o cálculo do valor médio é detectado na verificação do CRC do pacote.

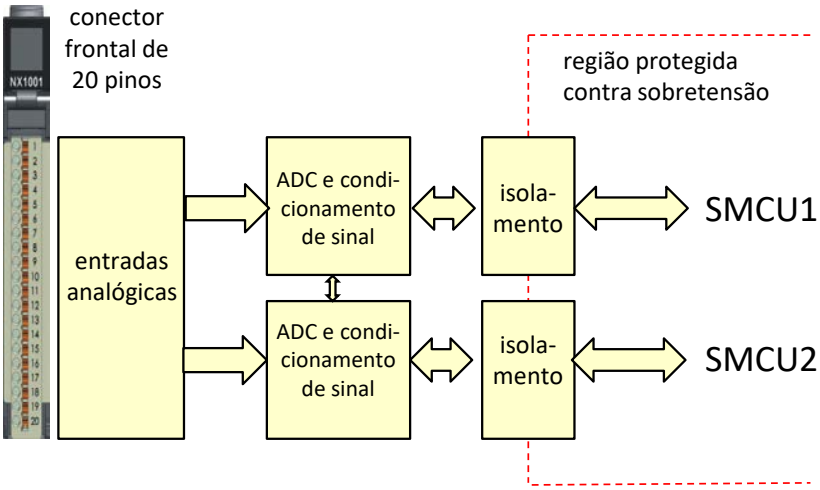


Figura 6 – Entrada de sinais analógicos

O ADC tem resolução de 24 bits. Para reduzir o ruído lido pelo ADC, apenas os 16 bits mais significativos são utilizados na leitura. Além disso, cada referência pode ser lida pelo outro ADC. Cada ciclo de leitura executa 9 leituras, isto é, todas as 8 entradas analógicas, e um teste de referência numa entrada alternada. Assim, todas as entradas são testadas após 16 ciclos de leitura. Depois disso, um interruptor analógico alterna entre o valor de referência e 0 Vdc, para o teste de valor seguro. Após mais 16 ciclos de leitura, o teste está completo.

6 Conclusões

Lloyd e Reeve (Lloyd and Reeve 2009) listam várias dificuldades que as empresas enfrentam para se adaptar a norma IEC 61508 e obter a certificação do primeiro produto. As dificuldades passam pela falta de experiência com segurança funcional da equipe de desenvolvimento e validação, pouca familiaridade com ferramentas e técnicas de tolerância a falhas e engenharia de software embarcado, documentação incompleta ou inconsistente, práticas usuais de desenvolvimento muito diferentes das impostas ou sugeridas pela norma, impossibilidade de rastrear os requisitos de segurança até a implementação, uso de código legado, adoção de ciclo de vida inadequado. Uma vez, entretanto, que consigam a primeira certificação, os demais produtos são mais facilmente certificados devido à mudança na cultura técnica introduzida na empresa.

O mercado para equipamentos certificados com SIL é excelente e talvez seja essa a melhor motivação para que tolerância a falhas e engenharia de software sejam bem recebidas no ambiente industrial de desenvolvimento e produção de equipamentos para sistemas relacionados à segurança. Da perspectiva do fabricante, a certificação de um equipamento aumenta enormemente a credibilidade do seu produto (Bard and Petprayoon 2013).

O projeto de SAIM é um movimento no sentido de dominar a tecnologia de desenvolvimento de equipamentos para sistemas de segurança. A função do nosso grupo é estudar os problemas relacionados e ajudar no processo de projeto e certificação (Vidal et al. 2014) (Bandeira et al. 2013). A empresa parceira, Altus SA, já recebeu e aprovou o resultado da primeira etapa do projeto de SAIM que foi realizada no nosso grupo, nas dependências da Universidade. O resultado desta etapa consiste na arquitetura de hardware do sistema, relatada aqui, e nas especificações de segurança correspondentes a esta arquitetura.

O documento que contém as especificações de segurança detalha não apenas a estrutura de blocos dos componentes, mas também a especificação elétrica e mecânica do módulo de entrada analógica. Todos os requisitos de projeto de hardware, tanto os relacionados à segurança como os requisitos não relacionados à segurança, fazem parte deste documento, que na versão atual tem a dimensão de 80 páginas. O documento precisa agora ser aprovado pela agência certificadora para a prototipação em hardware e o desenvolvimento do software possam ser iniciados. O encaminhamento deve ser pela empresa interessada na produção, que será também avaliada para fins de certificação de produto.

Como o processo de certificação envolve a empresa e também o seu processo de produção do produto, o desenvolvimento precisa ser conduzido com forte cooperação entre a empresa e o grupo de pesquisa. O aprendizado para ambas as partes tem sido o melhor resultado do projeto até o momento.

Agradecimentos

O trabalho foi financiado com recursos BNDES no âmbito do projeto SDCD.

Referências

Bandeira, Diego, Taisy S. Weber, Sergio L. Cechin, Rodrigo Dobler, and João C. Netto. 2013. “Assistente Para Desenvolvimento de Software Crítico Segundo a IEC 61508.” In *Workshop de Tolerância a Falhas*, 17–30. Brasília, DF: SBC.

Bard, Irv, and Patchanee Petprayoon. 2013. “Successful Compliance with IEC 61508 Safety Standards.” CT316. *IBM developerWorks*. May 21. http://www.ibm.com/developerworks/rational/library/compliance-IEC-61508-safety-standards/.

Bell, Ron. 2006. “Introduction to IEC 61508.” In *Proceedings of the 10th Australian Workshop on Safety Critical Systems and Software-Volume 55*, 3–12. Australian Computer Society, Inc.

———. 2011. “Introduction and Revision of IEC 61508.” In *Advances in Systems Safety*, edited by Chris Dale and Tom Anderson, 273–91. Springer London.

Bilich, Carlos, and Zaijun Hu. 2009. “Experiences with the Certification of a Generic Functional Safety Management Structure According to IEC 61508.” In *Computer Safety, Reliability, and Security*, edited by Bettina Buth, Gerd Rabe, and Till Seyfarth, 5775:103–17. Lecture Notes in Computer Science. Springer Berlin / Heidelberg.

Gall, Heinz, and Joachim Wen. 2010. “Functional Safety IEC 61508 and Sector Standards for Machinery and Process Industry the Impact to Certification and Users Including IEC 61508 2nd Edition.” In *23rd International Congress on Condition Monitoring and Diagnostic Engineering Management, COMADEM 2010, June 28, 2010 - July 2, 2010*, 73–81. Nara, Japan: Sunrise Publishing Limited.

Jin, H., M. A. Lundteigen, and M. Rausand. 2011. “Reliability Performance of Safety Instrumented Systems: A Common Approach for Both Low-and High-Demand Mode of Operation.” *Reliability Engineering & System Safety* 96 (3): 365–373.

Joannou, Paul, and Alan Wassyng. 2014. “Understanding Integrity Level Concepts.” *Computer*, no. 11: 99–101.

Lin, Zhihong, and Stephanie Pearson. 2013. “An inside Look at Industrial Ethernet Communication Protocols.” *White Paper Texas Instruments*..

Lloyd, M. H., and P. J. Reeve. 2009. “IEC 61508 and IEC 61511 Assessments-Some Lessons Learned.” *4th IET International Conference on Systems Safety*, 2A1.

Lundteigen, Mary Ann, and Marvin Rausand. 2009. “Architectural Constraints in IEC 61508: Do They Have the Intended Effect?” *Reliability Engineering & System Safety* 94 (2): 520–25.

Lundteigen, Mary Ann, Marvin Rausand, and Ingrid Bouwer Utne. 2009. “Integrating RAMS Engineering and Management with the Safety Life Cycle of IEC 61508.” *Reliability Engineering & System Safety* 94 (12): 1894–1903.

Malik, Robi, and Reinhard Mühlfeld. 2003. “A Case Study in Verification of UML Statecharts: The PROFIsafe Protocol.” *J. UCS* 9 (2): 138–51.

Neumann, Peter. 2007. “Communication in Industrial automation—What Is Going On?” *Control Engineering Practice* 15 (11): 1332–47.

Panesar-Walawege, Rajwinder Kaur, Mehrdad Sabetzadeh, Lionel Briand, and Thierry Coq. 2010. “Characterizing the Chain of Evidence for Software Safety Cases: A Conceptual Model Based on the IEC 61508 Standard.” In *3rd International Conference on Software Testing, Verification and Validation, ICST 2010, April 7, 2010 - April 9, 2010*, 335–44. ICST 2010 - 3rd International Conference on Software Testing, Verification and Validation. Paris, France: IEEE Computer Society.

Rostan, Martin, Joseph E. Stubbs, and Dmitry Dzilno. 2010. “EtherCAT Enabled Advanced Control Architecture.” In *2010 IEEE/SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, 39–44. IEEE.

Vidal, William, Rodrigo Dobler, Sérgio Cechin, Taisy Weber, and João Netto. 2014. “Aplicação Da IEC 61508 Na Prototipação de Protocolos Seguros de Comunicação.” In *Workshop de Testes E Tolerância a Falhas*, 147–59. Florianópolis: Sociedade Brasileira de Computação.

XVIII Workshop de Testes e Tolerância a Falhas

SBRC 2017
Sessão Técnica 3
Estratégias de Teste

Um injetor de falhas de comunicação para implementações do protocolo EtherCAT

Luiz Gustavo A. Gomes[1], João de Moraes[1], Taisy S. Weber[1], Sérgio L. Cechin[1], João Netto[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

{lgagomes, joao.moraes, taisy, cechin, netto }@inf.ufrgs.br

Abstract. *Computational systems are frequently used in many industrial areas by communicating geographically distant devices. Ethernet-based protocols, like EtherCAT, are commonly adopted for this communication to take place. By having a specification that does not dictate how a new EtherCAT device must be implemented, for a new product, a new coding must be made and validated. This work introduces a communication fault injector for the EtherCAT protocol, aiming the process of validation of new implementations against the occurrence of failures. In this article, we present a functional injector capable of injecting faults into commercial products that implement EtherCAT.*

Resumo. *Sistemas computacionais são frequentemente usados em diferentes áreas da indústria, comunicando dispositivos geograficamente distantes. Protocolos baseados em Ethernet, como o EtherCAT, são comumente adotados para que essa comunicação seja realizada. Por ter uma especificação que não dita como um novo dispositivo EtherCAT deve ser implementado para um novo produto, uma nova codificação deve ser feita e validada. Este trabalho introduz um injetor de falhas de comunicação próprio para o protocolo EtherCAT, visando o processo de validação de novas implementações frente à ocorrência de falhas. Neste artigo nós apresentamos um injetor funcional capaz de injetar falhas em produtos comerciais que implementam EtherCAT.*

1. Introdução

Com sistemas de controle digital se comportando como uma grande rede interconectada e o custo reduzido de equipamentos com interface Ethernet, surgiu um interesse econômico em introduzir sistemas de comunicação baseadas em Ethernet no domínio industrial [Neumann 2007]. Portanto, soluções em comunicação baseadas em Ethernet foram desenvolvidas especialmente para atender requisitos industriais como tempo e sincronismo.

Uma destas soluções é o protocolo EtherCAT [“EtherCAT Technology Group | HOME” 2016], planejado para dar suporte ao processamento de informações, monitoramento e controle de sistemas de controle para diversos setores industriais de maneira simples. Como o padrão EtherCAT não define uma implementação padrão pré-certificada [Zhou e Hu 2011] para qualquer novo dispositivo, o protocolo deve ser implementado pelos desenvolvedores e sua validação feita pelo órgão certificador responsável (*EtherCAT Technology Group*). Ferramentas que possam auxiliar desenvolvedores a testarem seus códigos diante da ocorrência de falhas existem, porém, são ferramentas comerciais ou são ferramentas que não cobrem o modelo de falhas definido pela especificação. Com o auxílio de uma ferramenta de injeção de falhas, o

processo de validação torna-se mais rápido, visto que a implementação já terá sido testada na ocorrência de falhas antes de ser submetida ao órgão certificador.

Com base neste cenário, o objetivo deste trabalho é apresentar um injetor de falhas próprio para EtherCAT, para validar os mecanismos de tolerância a falhas de implementações deste protocolo. Tal injetor faz parte de uma arquitetura de validação que futuramente será utilizada pelo grupo de pesquisa no desenvolvimento de uma nova linha de produtos. A partir dos resultados obtidos e apresentados aqui, é possível afirmar que foi desenvolvida uma ferramenta inovadora, especialmente feita para o processo de validação de implementações de EtherCAT, customizável e que pode ser estendida a outros protocolos.

Injeção de falhas é um dos pilares deste trabalho devido à sua importância no processo de validação de sistemas, onde mecanismos de tolerância a falhas são aprimorados de modo a garantir a dependabilidade deste sistema em sua fase operacional [Arlat et al. 1990].

O resultado deste artigo é uma continuação do trabalho anterior do grupo de pesquisa [Gomes et al. 2016], onde detalhes iniciais do projeto são apresentados, bem como o injetor resultante que serviu como prova de conceito e pavimentou o caminho para este trabalho alcançar o estágio atual.

O artigo apresenta um breve resumo do protocolo EtherCAT, trabalhos relacionados, experimentos realizados, decisões de projeto e resultados alcançados.

2. Protocolo EtherCAT

EtherCAT é um protocolo industrial baseado em Ethernet desenvolvido para oferecer alta performance em tempo real, baixo *jitter* e uma alta utilização da banda disponível. É largamente utilizado no controle de servo motores, coleta de dados de forma sincronizada e na área de automação em geral [Zhou e Hu 2011].

Uma rede EtherCAT é composta de um mestre EtherCAT e até 65535 escravos que podem estar conectados sem qualquer restrição de topologia (linha, árvore, anel, estrela ou qualquer combinação destas), permitindo que quaisquer dois dispositivos estejam distantes em até 100 metros entre si.

Para permitir a comunicação entre o mestre e os escravos, EtherCAT encapsula seus dados em telegramas EtherCAT, que são transportados dentro de um *frame* Ethernet padrão, como mostra a Figura 1. Através do *EtherType* (0x88A4) [EtherCAT Technology Group 2015] é possível saber quando dados EtherCAT estão contidos dentro do *frame*. Cada telegrama EtherCAT é subdividido em um cabeçalho EtherCAT e um ou mais datagramas EtherCAT [EtherCAT Technology Group 2014], que indicam qual tipo de acesso o mestre deseja executar (leitura / escrita) e quais escravos devem realizar uma determinada tarefa [EtherCAT Technology Group 2015].

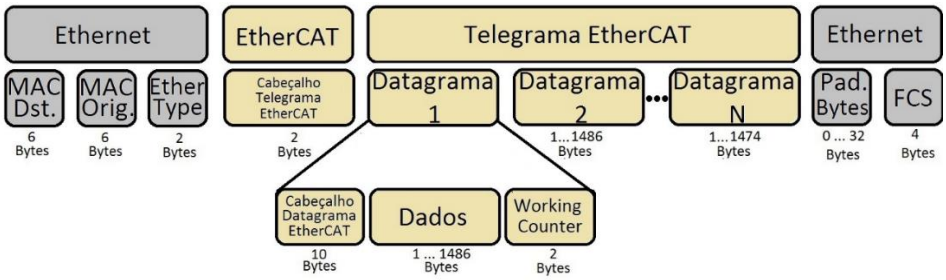


Figura 1. Estrutura de um *frame* Ethernet com dados EtherCAT

O início do ciclo de comunicação EtherCAT se dá com o mestre enviando um telegrama que passa por todos os nós escravos da rede. Cada escravo lê e insere seus respectivos dados no seu datagrama designado enquanto este está em trânsito. Como os *frames* não são colocados em um *buffer* para leitura ou escrita de dados, este é atrasado apenas pelos tempos de propagação do *hardware*. O último escravo a receber o *frame* detecta que não possui escravos subsequentes a quem possa repassá-lo e envia este *frame* no sentido contrário, em direção ao mestre, utilizando a característica *full duplex* da tecnologia Ethernet [EtherCAT Technology Group 2015]. Do ponto de vista de criação de *frames,* apenas o mestre pode criar e enviar *frames,* enquanto os escravos apenas podem realizar leituras e escritas nestes *frames* [EtherCAT Technology Group 2015].

2.1. Modelo de falhas de comunicação

Erros de comunicação podem ser causados pelos mais variados motivos, desde falhas de hardware até interferências do ambiente externo. As falhas consideradas na especificação do EtherCAT serão apresentadas a seguir.

Falhas de perda de pacotes ocorrem quando um ou mais pacotes não conseguem alcançar seu destino. Podem ser causadas, por exemplo, por congestionamento do enlace, fazendo com que o protocolo descarte pacotes. Entre outras causas, pode-se citar baixo desempenho de equipamentos físicos (roteadores, *switches*, etc.), mau funcionamento do *software* utilizado e rompimento do meio físico de transmissão.

Falhas por corrupção de dados ocorrem quando a mensagem que sai do dispositivo de origem é diferente da mensagem que chega ao dispositivo de destino. Pode ser causada por falhas em elementos físicos do sistema ou por erros de *software*.

Atraso de mensagens é observado quando restrições temporais não são respeitadas, isto é, o tempo esperado para um pacote ir de sua origem até seu destino é maior ou menor do que o esperado. No caso de EtherCAT apenas pacotes atrasados são considerados pela especificação, pois pacotes que chegam antes do tempo previsto não representam qualquer ameaça ao funcionamento normal.

Circulação de *frame*(s) indefinidamente pelo sistema acontece no caso de ruptura do enlace ou mau funcionamento de alguma porta do dispositivo escravo, fazendo com que uma topologia em linha seja dividida em duas. Um *frame* viajando através dos dispositivos pode começar a circular, ou seja, ser transportado apenas entre os nós de um lado da ruptura sem nunca alcançar o último escravo ou o mestre. Tal situação pode ser observada a seguir. Na Figura 2 (a) é exibido o caminho normal que um pacote percorre. Ao ocorrer uma falha de enlace entre os escravos 1 e 2 (Figura 2 (b)), estes fecham suas portas 1 e 0 respectivamente e *frames* à direita do escravo 2 seguirão circulando indefinidamente, pois estão impossibilitados de alcançar o nó mestre. Como não existe

um contador de *hops* em um telegrama ou datagrama EtherCAT, esta situação representa um risco real ao funcionamento da rede.

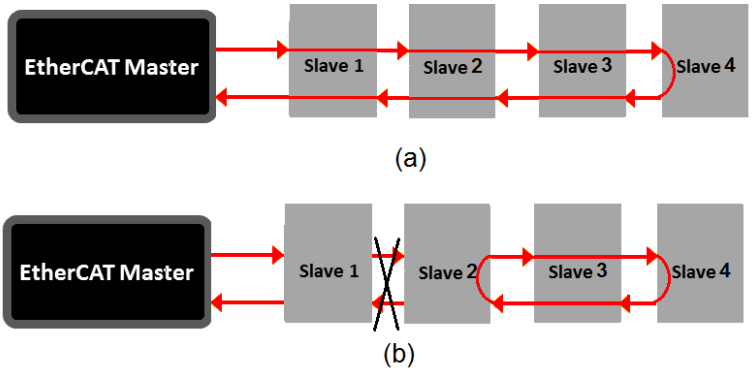


Figura 2. (a) Caminho normal percorrido pelos *frames*. (b) Caminho percorrido pelos *frames* após uma falha de enlace

2.2. Mecanismos de detecção e correção de falhas

A seguir são descritos os métodos do protocolo para lidar com as falhas apresentadas anteriormente.

A primeira medida é o *Frame Check Sequence* (FCS) presente no *frame* Ethernet. Este é verificado tanto pelo mestre quanto pelos escravos para determinar se um *frame* foi recebido corretamente. Como vários escravos podem alterar o conteúdo do *frame* durante o percurso, seu FCS é calculado por cada nó escravo tanto na sua recepção quanto na sua retransmissão. Se um erro de *checksum* é encontrado, o escravo não repara o FCS, mas sinaliza o mestre incrementando um contador de erros interno, de modo que o ponto da falha seja precisamente localizado na topologia.

O *Working Counter* é o último campo de um datagrama EtherCAT. Possui um valor esperado calculado pelo mestre e é incrementado pelos escravos a cada leitura ou escrita bem-sucedida realizada no datagrama, ou seja, se ao menos um *byte* ou um *bit* em todo o datagrama foi lido ou escrito com sucesso. Ao comparar o *Working Counter* com o número esperado de escravos que deveriam acessar dados, o mestre pode saber quantos escravos processaram seus dados correspondentes. Se o *Working Counter* presente no datagrama vindo dos escravos possui um valor diferente do esperado, o mestre não envia os dados deste datagrama para a aplicação [EtherCAT Technology Group 2015].

Escravos contam com dois *watchdogs* para monitorar comunicações malsucedidas: enquanto um monitora acessos de escrita feitos *no* escravo, outro monitora leituras e escritas bem-sucedidas feitas *pelo* escravo. Caso não ocorra nenhuma comunicação dentro do tempo previsto, o respectivo *watchdog* tem seu contador incrementado e seus sinais de saída não são entregues [EtherCAT Technology Group 2014].

Para lidar com *frames* que podem circular indefinidamente pelo sistema, existe o *Circulating Bit,* que é um *bit* para informar aos dispositivos da topologia quando um *frame* está circulando pela topologia na ocorrência de uma falha de enlace que cause uma situação semelhante à da Figura 2 (b). Para prevenir isto, um escravo sem um enlace em sua porta 0 fará o seguinte: se o *Circulating Bit* de um datagrama EtherCAT é 0, muda

seu valor para 1 e o envia adiante. Se o *Circulating Bit* é 1, o *frame* não é processado e sim descartado [EtherCAT Technology Group 2014].

3. Trabalhos relacionados

A seguir é apresentado um conjunto, não exaustivo, de ferramentas voltadas à verificação e teste em geral de implementações do protocolo EtherCAT. Por ser um protocolo aberto e sem uma implementação padrão para todos os dispositivos, ferramentas deste tipo são desejáveis para se avaliar a conformidade de uma implementação frente à especificação. As ferramentas descritas a seguir foram escolhidas por servirem, mesmo que parcialmente, a este propósito.

beStorm [“beSTORM® Software Security Testing Tool” 2016] é uma ferramenta voltada para avaliação de segurança que realiza uma análise exaustiva em busca de novas vulnerabilidades em aplicações que utilizam algum protocolo de rede. Ao gerar, automaticamente, ataques, *beSTORM* testa as aplicações quanto a sua segurança (*security*) antes que estes atinjam o mercado.

EC-STA [“EC-STA [EtherCAT Slave Test Application]” [S.d.]] é um *framework* desenvolvido para realização de testes para verificação do funcionamento durante e após o desenvolvimento de escravos EtherCAT, como alteração de estado e operações de escrita e leitura. Possui como componente principal uma aplicação baseada em .NET e escrita em linguagem C#. Também é fornecido o código fonte completo, permitindo ao usuário adicionar suas próprias funções. Seu funcionamento se dá configurando um computador com um mestre EtherCAT (fornecido separadamente) e conectando este ao escravo que será avaliado.

EtherCAT Conformance Test Tool[“EtherCAT Technology Group | EtherCAT Conformance Test Tool (CTT)” 2016] é uma ferramenta desenvolvida pelo próprio grupo responsável por manter e gerenciar o protocolo, (*EtherCAT Technology Group* – ETG) para realizar testes em dispositivos escravos EtherCAT a fim de verificar sua conformidade. Nesta ferramenta, *frames* EtheCAT são enviados ao dispositivo sob teste através de uma porta Ethernet padrão de modo a estimulá-lo. Por ter suporte e ser licenciada pelo grupo responsável pelo protocolo EtherCAT, é utilizada pelo órgão verificador como última etapa dos processos de validação e certificação e é tida como mandatória para que produtos que implementam EtherCAT possam ser certificados e assim liberados para atingir o mercado.

O uso da ferramenta *beStorm* para a verificação do protocolo EtherCAT é válido, mas por ser focada apenas em encontrar falhas de segurança, pode não ser mandatória quando se procura testar o correto funcionamento de uma dada implementação. Além disso, por ser uma ferramenta do tipo “caixa-preta”, não é sabido quais destes testes de segurança podem ser utilizados para identificar erros na implementação.

As ferramentas *EC-STA* e *EtherCAT Conformance Test Tool* são capazes de testar completamente uma implementação do protocolo EtherCAT de acordo com a especificação. Porém além de serem ferramentas pagas, são voltadas para o produto em seu estágio final de desenvolvimento, quando se está prestes a ser lançado no mercado, podendo não se adequar a testes realizados durante o processo de desenvolvimento. Além disso, os testes são realizados seguindo uma abordagem simples, através de estímulos nos dispositivos e comparação das respostas com resultados esperados, não atuando nas mensagens de comunicação.

4. O injetor de falhas TANATTOS

4.1. Visão geral

A empresa parceira no projeto pretende desenvolver uma nova linha de produtos, que implementam o protocolo EtherCAT. Para tal, é necessário que este novo equipamento passe por uma bateria de testes de campo em condições normais de operação e em condições de falhas antes de ser enviado para o processo de validação e certificação do equipamento pelo órgão responsável.

O uso de um injetor de falhas que ponha o produto à prova irá auxiliar a equipe na identificação de eventuais falhas de implementação. Com isso as chances de que novas implementações do protocolo EtherCAT sejam certificadas aumentará, pois torna possível identificar combinações de eventos que possam levar a comportamentos não desejados [Yu et al. 2005].

O injetor de falhas TANATTOS (*Tool for fAult iNjection on ethercAT implementaTiOnS*), foi desenvolvido para que implementações do protocolo EtherCAT possam ser testadas à medida que vão sendo construídas. Além disto, tal ferramenta foi planejada para contar com uma baixa intrusividade, ou seja, mínima interferência causada pelo ato de injetar as falhas. A Figura 3 exibe a janela principal do injetor, que conta com botões relativos às funções de injeção de falhas, abrir os arquivos de *log* pós-injeção das falhas e arquivo de ajuda, bem como informações referentes ao desenvolvedor.

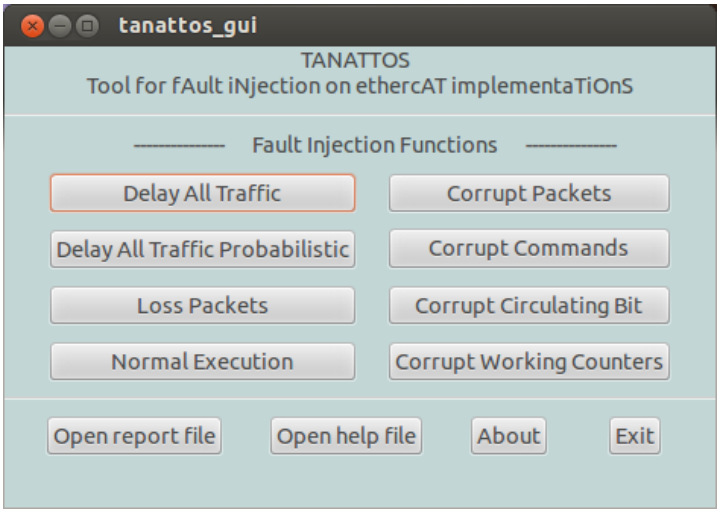


Figura 3. Janela principal do injetor TANATTOS

TANATTOS age simulando um ataque do tipo *man-in-the-middle*, ou seja, interceptando pacotes trocados entre dois dispositivos (Figura 4 (b)), sendo assim um componente independente do sistema. Do ponto de vista dos dispositivos mestre e escravo a comunicação flui diretamente sem interferências (Figura 4 (a)), quando na verdade esta comunicação passa através do injetor. Para isto, o injetor é executado em um computador exclusivo que conta com duas interfaces de rede.

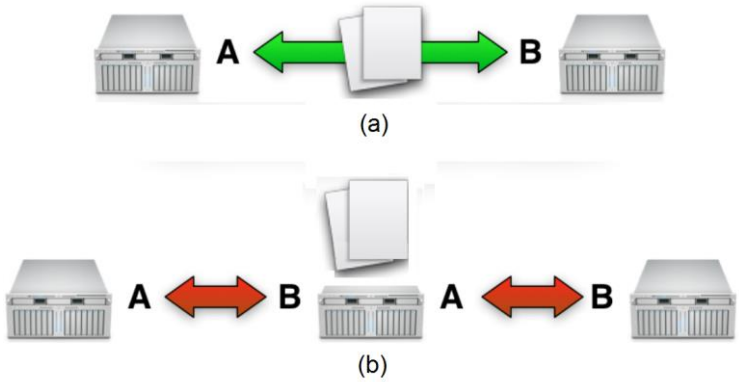


Figura 4. (a) Como os dispositivos enxergam a comunicação. (b) Como a comunicação ocorre de fato

Com relação à estrutura do injetor TANATTOS, este é dividido em três componentes principais: uma biblioteca para captura de pacotes em alta velocidade, chamada PF_RING [“PF_RING” 2011], uma aplicação que implementa as funções de injeção de falhas e gera um arquivo de *log* com dados referentes ao teste aplicado e uma interface gráfica que envia parâmetros para esta aplicação, com base no tipo de falha a ser injetada, como ilustrado pela Figura 5.

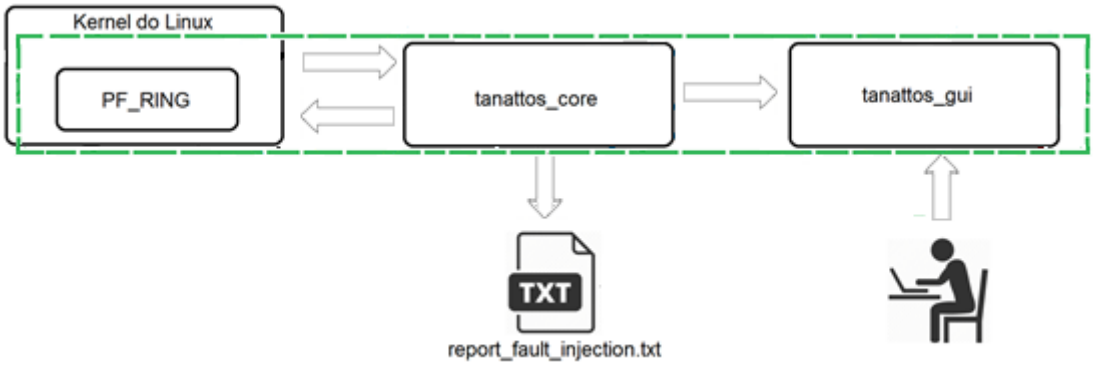


Figura 5. Organização interna do injetor TANATTOS

PF_RING é uma biblioteca para captura e envio de pacotes em alta velocidade, a qual torna possível a análise e manipulação de pacotes e tráfego ativo de rede. Essa biblioteca possui um módulo que deve ser carregado no *Kernel* do *Linux*, permitindo que os pacotes sejam trocados diretamente entre a interface de rede e aplicações do usuário sem utilizar mecanismos do *Kernel* do *Linux*. Neste injetor utiliza-se o modo DNA (*Direct NIC Access*) do PF_RING, que mapeia a memória e registradores das interfaces de rede para o espaço do usuário sem utilizar a *Linux Network API* (*Linux NAPI*), evitando perdas de performance significativas.

4.2. Funções de injeção de falhas

Com base no modelo de falhas levantado a partir da especificação do protocolo, foram criadas funções de injeção de falhas que exercitam os mecanismos de detecção e correção de falhas já descritos. Assim, o injetor TANATTOS conta com as funções de atraso de pacotes, atraso de pacotes intermitente, perda de pacotes, corrupção de pacotes, corrupção

de comandos de escrita e leitura, corrupção do *Circulating Bit*, corrupção do *Working Counter* e uma função de execução normal.

As funções de atraso de pacotes mantêm os pacotes recebidos por uma das interfaces de rede em um *buffer* (próprio do PF_RING) por um tempo especificado pelo usuário. A diferença entre as funções de atraso de pacotes e atraso de pacotes intermitente é que a função de atraso comum atrasa todo o fluxo de dados, enquanto a função de atraso intermitente simula uma situação onde a comunicação entre dois dispositivos alterna entre períodos de funcionamento normal e períodos onde o fluxo de pacotes inteiro é atrasado. Estes testes possuem dois objetivos principais: o primeiro é avaliar o correto funcionamento dos *watchdogs* do escravo ao reproduzir uma situação de "silêncio" na comunicação por um período de tempo maior que o contador interno dos *watchdogs*. O segundo objetivo é obter uma métrica da margem de operacionalidade do dispositivo mestre através da seguinte questão: "*se é sabido que um dispositivo escravo introduz um atraso de X milissegundos na comunicação do sistema como um todo, qual o maior atraso que um mestre pode suportar antes de entrar em colapso?* ".

Na função de perda de pacotes, o injetor não reenvia os pacotes que estiverem sendo recebidos em uma das interfaces de rede. Este cenário simula um barramento ou dispositivo com problemas de conexão, onde pacotes são perdidos durante a comunicação e tem como finalidade avaliar a robustez do dispositivo.

As funções de corrupção de pacotes e comandos de leitura / escrita permitem respectivamente alterar o conteúdo de um *byte* ou *bit* específico de um pacote ou os dados oriundos de uma leitura ou escrita. Estas têm por objetivo forçar erros de *checksum* e verificar o correto funcionamento das funções de cálculo de CRC da camada de Enlace do escravo e do contador interno responsável por armazenar o número de erros deste tipo.

Já a função de corrupção do *Working Counter* é a única que não tem como alvo testar a funcionalidade de um dispositivo escravo e sim observar como o mestre se comporta ao receber um valor inesperado de *Working Counter.*

A função de corrupção do *Circulating Bit* busca alterar o valor do *Circulating Bit* dos datagramas EtherCAT contidos dentro de cada pacote. Com isso, alterar seu valor de 0 (padrão) para 1 fará com que o pacote não seja processado e sim descartado ao chegar no dispositivo escravo.

Finalmente, a função de execução normal oferece a opção de uma comunicação sem falhas. Esta função tem o objetivo de verificar se o ambiente de testes montado está configurado de maneira correta, ou seja, se a comunicação entre dois dispositivos está funcionando adequadamente antes que os testes de injeção de falhas possam ser iniciados.

4.3. Metodologia de injeção de falhas

O método com que as falhas são injetadas busca chegar o mais próximo de um cenário real, onde falhas podem acontecer a qualquer momento. Deste modo, dificilmente um teste retornará os mesmos resultados caso seja executado várias vezes com os mesmos parâmetros de entrada. Para isto, foi adotado um método onde os pacotes que irão sofrer a injeção de falhas são escolhidos aleatoriamente seguindo uma distribuição uniforme.

Para se injetar uma falha, primeiro seleciona-se o tipo de falha e em seguida devem ser escolhidos alguns parâmetros. Dentre estes parâmetros três são comuns a todas as funções: as duas interfaces de rede correspondentes aos dispositivos mestre e escravo

(interfaces DNA) e uma direção para injetar a falha, seja no sentido mestre para escravo ou escravo para mestre. Dependendo do tipo de falha a ser injetada, outros parâmetros específicos devem ser informados, como chance de cada pacote sofrer a falha, um *bit* ou *byte* a ser corrompido ou tempo de atraso.

Após a definição da função de falhas escolhida e do preenchimento dos seus respectivos parâmetros, o experimento está pronto para ser iniciado. Com exceção do teste de atraso de pacotes, o usuário informa um valor entre 0,0 e 1,0, que corresponde à chance percentual de cada pacote sofrer a falha (0,0 sendo equivalente a 0% e 1,0 a 100%). Assim que um pacote chega à interface definida, o injetor calcula um valor entre 0,0 e 1,0 de maneira aleatória, seguindo uma distribuição uniforme. Caso este valor seja menor que o valor informado pelo usuário, o pacote sofrerá a injeção de falhas, caso contrário, o pacote será encaminhado normalmente.

Uma característica existente no modo de operação de TANATTOS é que apenas um tipo de falha pode ser injetada por vez, não sendo permitido que diferentes testes sejam executados sequencialmente de forma automática. Para tal é necessário que após a execução de um teste outro seja iniciado de forma manual.

4.4. Geração de *log* de injeção de falhas

Após o encerramento da seção de injeção de falhas, informações pertinentes à função de injeção de falhas escolhida serão salvas em um arquivo de texto, podendo ser acessado pelo botão correspondente na janela principal do injetor, sendo exibido através do editor de texto padrão do sistema (*gedit* por exemplo).

Neste relatório são exibidos a data e hora de início do teste, a quantidade de pacotes recebidos e enviados em cada interface e direção bem como uma indicação de qual direção foi escolhida para a injeção de falhas e a quantidade de falhas que foram injetadas. Nesse caso, para os testes de atraso de pacotes é informado quantos pacotes foram atrasados e o tempo de atraso. Para o teste de corrupção de pacotes também é exibida a quantidade (absoluta e percentual) de pacotes que foram corrompidos. Para os testes de corrupção de dados de comandos, corrupção de *Circulating Bit* e corrupção de *Working Counter* é exibida a quantidade de datagramas EtherCAT afetados e a média de datagramas por pacote. Para o teste de execução normal é exibida apenas a quantidade de pacotes transmitidos entre as interfaces.

Nos testes que envolvem corrupção dos pacotes (corrupção, *Circulating Bit, Working Counter* e dados de comandos), quando um pacote é alterado pela respectiva função, é exibida em uma tela de terminal o conteúdo do pacote antes e depois da falha

5. Ambiente experimental e testes realizados

O ambiente experimental apresentado na seção 4.1, conta com dispositivos que executam o protocolo EtherCAT cedidos pela empresa parceira no projeto: um mestre EtherCAT que é monitorado em tempo real por um computador separado, um escravo EtherCAT que recebe comandos vindos do mestre e um segundo computador que executa o injetor, sendo colocado entre o mestre e o escravo, interceptando os pacotes trocados. O fato dos dispositivos cedidos já serem previamente certificados serviu como meio de validar o injetor, pois estes dispositivos já foram certificados.

O computador onde o injetor foi executado conta com processador *Intel Core* i7-3770 3.4 GHz com 16 GB de memória RAM, com sistema operacional *Ubuntu* 12.04

LTS 64 *bits* e duas placas de rede *Intel Gigabit* modelo e1000e 82574L que são responsáveis por interligar os dispositivos mestre e escravo ao injetor de falhas. Já o computador que monitorava o mestre EtherCAT conta com um processador *Intel Core* 2 *Duo* E4000 2.4 GHz, 4 GB de memória RAM e sistema operacional *Windows*® 7 32 bits.

Antes que o sistema fosse fisicamente montado, utilizou-se uma ferramenta de configuração do *hardware* a ser utilizado que define: uma aplicação que é a carga de trabalho e quais dispositivos seriam usados, como seriam conectados e qual seria sua disposição física nos barramentos, de modo que fosse possível o monitoramento via *software*. A Figura 6 exibe uma captura de tela que mostra a modelagem final do sistema, com os dois barramentos (barramento do mestre à esquerda e do escravo à direita) conectados e a disposição física dos dispositivos a serem usados.

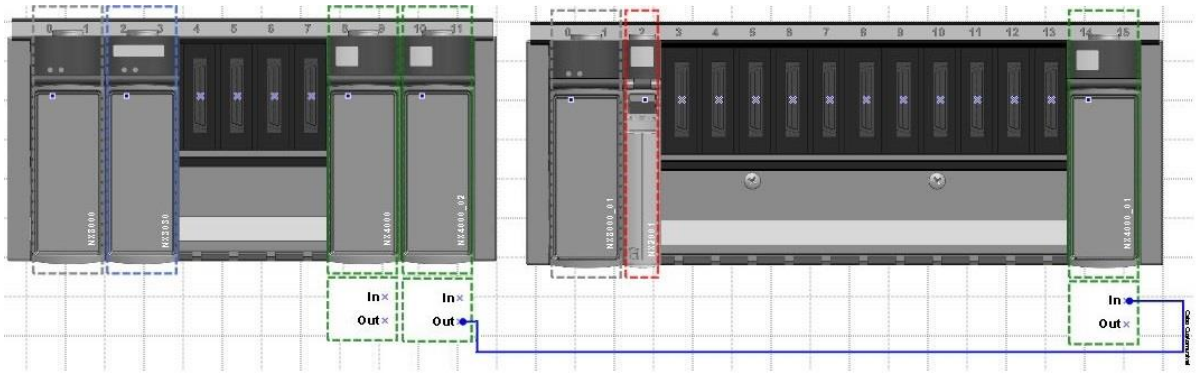


Figura 6. Arquitetura do sistema modelada via *software*

Em seguida, o ambiente de testes final foi montado seguindo o modelo desenvolvido em software. A Figura 7 mostra um esquemático do sistema, exemplificando as ligações entre todos os equipamentos. Tais equipamentos são: dispositivos “A” são fontes de alimentação 30 W e 24 Vdc, uma para cada barramento. O dispositivo “B” é o módulo escravo EtherCAT. Dispositivos “C” são responsáveis por transportar os pacotes de e para o mestre e o escravo, por isto devem ser dois. Estes dispositivos “C” serão conectados cada um a uma das placas de rede do computador executando o injetor de falhas. O dispositivo “E” é um controlador programável que age como mestre. Dispositivo “D” serve para monitorar o fluxo de pacotes que transita pelo mestre e está conectado ao computador monitor para fins de depuração, onde o *sniffer Wireshark* [“Wireshark · Go Deep.” 2017] é executado. Com o auxílio do *Wireshark* é possível verificar se os testes que envolvem corrupção de dados estão surtindo efeito. Isto se faz comparando o conteúdo do pacote que sai do mestre com o conteúdo do pacote corrompido que sai do injetor em direção ao escravo, por exemplo.

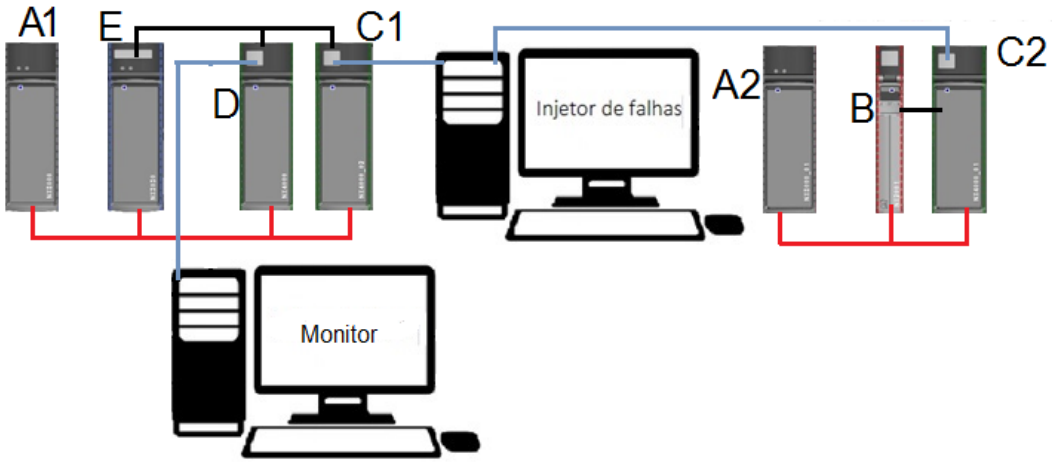


Figura 7. Arquitetura do ambiente utilizado durante os experimentos

O *software* de monitoramento conta com variáveis de diagnóstico referentes ao comportamento do mestre: *byWHSBBusErrors* que é um contador de falhas no barramento variando de 0 a 255; *adwModulePresenceStatus*, que é um *array* de DWORDS representando o *status* de presença dos dispositivos de entrada e saída declarados nos barramentos individualmente; *adwRackIOErrorStatus,* que representa erros nos dispositivos dos bastidores e *byHotSwapAndStartupStatus,* que informa através de códigos de erro, situações anormais no barramento responsáveis pela parada da aplicação.

Inicialmente foi aplicada a função de execução normal para saber o estado normal das variáveis apresentadas anteriormente, ou seja, qual valor é exibido por elas nas condições normais de funcionamento. Assim, são obtidos os valores correspondentes a uma execução sem falhas, que serão usados como referência para testes futuros. A Figura 8 mostra tais valores, em especial NORMAL_OPERATING_STATE para *byHotSwapAndStartupStatus*, que indica operação normal do sistema, os valores 1024 e 16388 para *adwModulePresenceStatus*, que indicam que os dispositivos funcionam corretamente e os valores 0 para *adwRackIOErrorStatus* indicando uma operação sem erros. Neste trabalho, por uma questão de espaço, serão apresentados e discutidos apenas os testes de perda e corrupção de pacotes.

WHSB	T_DIAG_WHSB	
byHotSwapAndStartupStatus	EN_HOT_SWAP	NORMAL_OPERATING_STATE
adwRackIOErrorStatus	ARRAY [0..31] OF DWORD	
adwModulePresenceStatus	ARRAY [0..31] OF DWORD	
byWHSBBusErrors	BYTE	0

adwModulePresenceStatus	ARRAY [0..31] OF DWORD	
adwModulePresenceStatus[0]	DWORD	1024
adwModulePresenceStatus[1]	DWORD	16388
adwRackIOErrorStatus	ARRAY [0..31] OF DWORD	
adwRackIOErrorStatus[0]	DWORD	0
adwRackIOErrorStatus[1]	DWORD	0

Figura 8. Variáveis utilizadas para monitoramemento e seus valores padrão

Para o teste de perda de pacotes, o injetor foi configurado a descartar 25% dos pacotes na direção do mestre para o escravo. Este valor foi escolhido porque testes preliminares mostraram que probabilidades menores do que 25% não foram capazes de gerar erros na operação normal dos dispositivos.

Após o descarte de pacotes ocorrido, a variável *byHotSwapAndStartupStatus* apresentou o *status* APPL_STOP_MODULES_NOT_READY, que, segundo o manual dos dispositivos, indica que a aplicação se encontra em modo *Stop* e todas as consistências foram realizadas com sucesso, mas os dispositivos não estão aptos para a partida do sistema. Isto é um indicativo que a perda de pacotes foi significativa a ponto de os dispositivos mestre e escravo não conseguirem executar os procedimentos necessários para o início do funcionamento. A variável *adwRackIOErrorStatus* por sua vez, apresentava os valores 1024 e 16388, diferentes dos valores 0 obtidos pela execução normal, indicando problemas devido à perda de pacotes. Os dados coletados podem ser vistos na Figura 9.

WHSB	T_DIAG_WHSB	
byHotSwapAndStartupStatus	EN_HOT_SWAP	APPL_STOP_MODULES_NOT_READY
adwRackIOErrorStatus	ARRAY [0..31] OF DWORD	
adwRackIOErrorStatus[0]	DWORD	1024
adwRackIOErrorStatus[1]	DWORD	16388

Figura 9. Variáveis *byHotSwapAndStartupStatus* e *adwRackIOErrorStatus* após a ocorrência de perda de pacotes

No teste de corrupção do *Working Counter*, cada datagrama EtherCAT tinha 40% de chance de sofrer uma alteração em seu respectivo *Working Counter* na direção mestre para escravo. A Figura 10 exibe a tela de configuração deste teste.

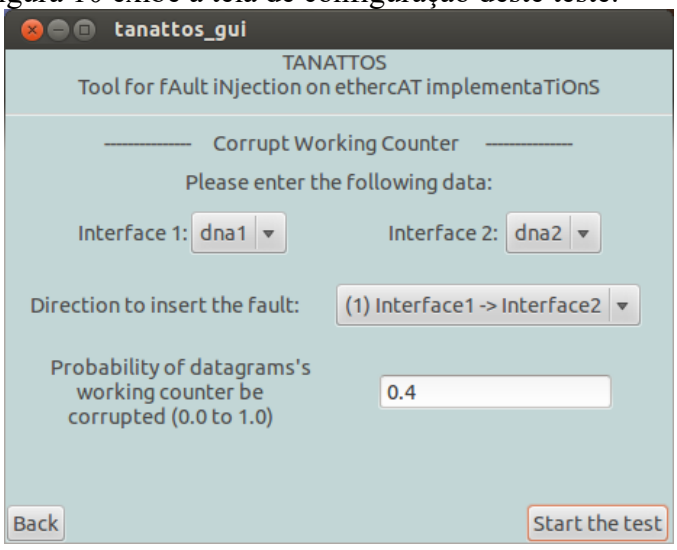


Figura 10. Configuração do teste de corrupção de pacotes

As variáveis *byHotSwapAndStartupStatus* e *byWHSBBusErrors* apresentam valores diferentes do encontrado pelo teste de execução normal, indicando que problemas devido à alteração de dados dos pacotes sem alterar o campo FCS foram detectados. Como mostrado pela Figura 11, *byHotSwapAndStartupStatus* apresenta o *status* APPL_STOP_MODULES_NOT_READY e *byWHSBBusErrors* agora contabiliza 215 erros.

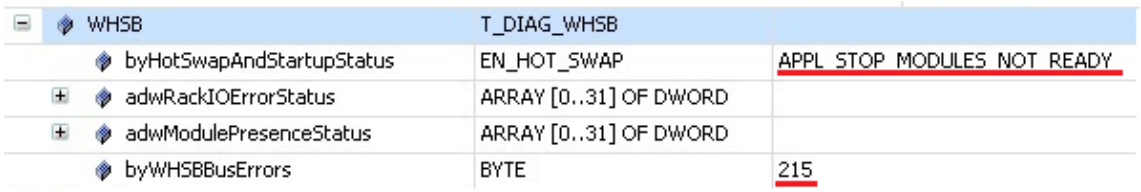


Figura 11. Variáveis *byHotSwapAndStartupStatus* e *adwRackIOErrorStatus* após corrupção de pacotes

O comportamento apresentado pelos dispositivos e refletido através das variáveis de diagnóstico, mostra que a comunicação entre eles foi de fato perturbada pelas falhas injetadas. Dessa maneira, os testes aplicados bem como os resultados obtidos serviram o propósito de validar a ferramenta desenvolvida.

6. Conclusão

Este trabalho descreveu o projeto e implementação do injetor de falhas TANATTOS, sendo próprio para o protocolo EtherCAT. Este injetor permite que implementações do protocolo sejam testadas mediante falhas durante seu processo de desenvolvimento, antes do processo de validação final realizado pelo órgão responsável. Para tanto, não é exigida nenhuma modificação no código-alvo nem uma carga de trabalho específica.

Por ser localizado fora dos dispositivos mestre e escravo, TANATTOS apresenta uma natureza minimamente intrusiva, agindo diretamente nos pacotes trocados entre estes. Além disto, a rápida operação do injetor sobre os pacotes é alcançada através do uso da biblioteca PF_RING em seu modo DNA.

TANATTOS pode ser utilizado para medir a cobertura de falhas de uma dada estratégia de tolerância a falhas implementada no sistema-alvo, porém seu objetivo principal é ajudar desenvolvedores a avaliar se as implementações dos métodos de tolerância a falhas do protocolo EtherCAT atendem aos requisitos impostos pela especificação.

Por ser uma ferramenta em *software*, TANATTOS é flexível o bastante para ser adaptado para protocolos diferentes de EtherCAT. De fato, as funções de atraso e perda de pacotes podem ser reaproveitadas. Outros tipos de falhas, porém, podem precisar de mudanças para atender às características específicas destes protocolos.

Em seu estado atual TANATTOS possui todas as funções de injeção de falhas anteriormente descritas já implementadas, porém a função de corrupção de pacotes não pode ser testada de forma adequada. É necessário que um erro forçado de CRC seja detectado e para isto é preciso que: (1) as interfaces de rede transmitam o campo de FCS dos pacotes para as camadas superiores e (2) que pacotes com erro de CRC não sejam descartados. Assim, não é desejável que tais pacotes sejam descartados pelas interfaces ou que o CRC seja recalculado, mascarando assim a falha injetada. Uma possível solução usando o utilitário *ethtool* presente na distribuição *Ubuntu* do *Linux* permitia que as condições (1) e (2) fossem atendidas, mas os comandos utilizados no *ethtool* para isto acabaram causando conflitos com o PF_RING.

Agradecimentos

Trabalho desenvolvido no Projeto SDCD - FAURGS/UFRGS/BNDES – (13.2.0646.1 - projeto 8108).

7. Referências

Arlat, J. et al. (1990). “Fault injection for dependability validation: A methodology and some applications”, *IEEE Transactions on Software Engineering*, v. 16, n. 2, p. 166–182.

beSTORM® Software Security Testing Tool (2016). Disponível em: <http://www.beyondsecurity.com/bestorm.html>, Acesso em 26 Out. 2016.

EC-STA [EtherCAT Slave Test Application] ([S.d.]). Disponível em: <http://www.acontis.com/eng/products/ethercat/ec-sta/index.php>, Acesso em 26 Out. 2016.

EtherCAT Technology Group (2014). “EtherCAT Slave Controller – Hardware Data Sheet Section 1”.

EtherCAT Technology Group (2015). “EtherCAT – The Ethernet Fieldbus”, . EtherCAT Technology Group. Disponível em: <https://www.ethercat.org/en/downloads.html>.

EtherCAT Technology Group | EtherCAT Conformance Test Tool (CTT) (2016). Disponível em: <https://www.ethercat.org/en/ctt.htm>, Acesso em 27 Out. 2016.

EtherCAT Technology Group | HOME (2016). Disponível em: <https://www.ethercat.org/default.htm>, Acesso em 30 Mai. 2016.

Gomes, L. G. et al. (2016). “Injeção de falhas de comunicação sobre implementações do protocolo EtherCAT”, *14ª Escola Regional de Redes de Computadores*, p. 53–60.

Neumann, P. (2007). “Communication in industrial automation—What is going on?”, *Control Engineering Practice*, v. 15, n. 11, p. 1332–1347.

PF_RING (4 ago 2011). Disponível em: <http://www.ntop.org/products/packet-capture/pf_ring/>, Acesso em 18 Out. 2016.

Wireshark · Go Deep. (2017). Disponível em: <https://www.wireshark.org/>, Acesso em 27 Mar. 2017.

Yu, Y. et al. (2005). “A state of research review on fault injection techniques and a case study”, Em *Annual Reliability and Maintainability Symposium, 2005. Proceedings.* IEEE.

Zhou, T. e Hu, J. (2011). “Design and realization of EtherCAT master”, Em *Electronic and Mechanical Engineering and Information Technology*. IEEE.

Análise e avaliação de Teste de Intrusão para a estratégia de recomendações Tramonto

Daniel Dalalana Bertoglio[1], **Avelino Francisco Zorzo**[1]

[1]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brazil

daniel.bertoglio@acad.pucrs.br, avelino.zorzo@pucrs.br

Abstract. *Nowadays, research on Penetration Testing (Penetration Tests) has new directions in Information Security. Methodologies, techniques and tools have been developed to meet the main challenges of these tests: automation, effectiveness and flexibility. Tramonto is a recommendation strategy designed to assist the tester with these challenges, relating concepts and applications of widely-known methodologies for security testing. This paper discusses the relation of Penetration Tests with Tramonto, analyzing a real test case from the perspective of the proposed strategy.*

Resumo. *Atualmente, as pesquisas sobre Testes de Intrusão (Penetration Tests) apresentam novos rumos para a área de Segurança da Informação. Metodologias, técnicas e ferramentas têm sido desenvolvidas para atender os principais desafios destes testes: automatização, eficácia e flexibilidade. A Tramonto é uma estratégia de recomendações criada para auxiliar o tester com esses desafios, relacionando conceitos e aplicações de metodologias amplamente conhecidas para testes de segurança. Este trabalho discute a relação de Testes de Intrusão com a Tramonto, analisando um caso real de teste sob a ótica da estratégia proposta.*

1. Introdução

Atualmente, a informação é considerada o ativo mais valioso para a maioria das empresas. Isso é justificado pelo fato de que o mercado vem emergindo com empresas que possuem um modelo de negócio baseado na informação. Paralelamente, as pesquisas relacionadas com a Segurança da Informação têm procurado soluções para proteger e mitigar o alto número de incidentes de segurança no contexto empresarial. Essas soluções não buscam apenas avaliar a existência de fraquezas e vulnerabilidades nos cenários-alvo, mas sim tratar o impacto dessas brechas na organização, caso algum atacante explore-as e efetue seus ataques.

Tais soluções variam entre mecanismos de defesa, softwares de monitoramento de incidentes, políticas e controles de diretivas e também de procedimentos de avaliação e teste de segurança. Tratando-se de avaliação e teste de segurança, que procuram mensurar, identificar e analisar qual o estado de segurança de um processo, controle, ativo, sistema e até rede, uma das técnicas conhecidas é o Teste de Intrusão (*Pentest*). Segundo [Lam et al. 2004], testes de intrusão aproximam a realidade dos ataques através da simulação do comportamento de um atacante. Em termos gerais, um teste de intrusão

pode ser definido como a tentativa deliberada e controlada de invadir um sistema ou rede com o objetivo de avaliar o estado de segurança do alvo [Bishop 2007].

Testes de intrusão possuem características específicas que variam de acordo com o alvo. Podem ser considerados diferentes aspectos quanto as atividades executadas, divisão de fases, estratégia do teste e até mesmo nas ferramentas utilizadas. Assim, o teste de intrusão permite também que as avaliações possuam objetivos variados, como o aumento da segurança dos sistemas, a identificação de vulnerabilidades, o teste da equipe de segurança da empresa alvo e até mesmo o aumento da segurança organizacional e de pessoas [Henry 2012].

Tantas possibilidades ofertadas por esse tipo de teste de segurança tornam a padronização destes testes de intrusão uma tarefa complexa. Existem metodologias de teste de segurança, posteriormente citadas nesse trabalho, que objetivam tratar esse problema da padronização. Contudo, das metodologias que são mais disseminadas, apenas uma é direcionada para testes de intrusão. A partir disso, a criação de uma estratégia de recomendações de teste é uma possível solução para auxiliar as atividades no processo de um teste de intrusão. Dessa forma, criou-se a Tramonto [Dalalana Bertoglio and Zorzo 2016], uma estratégia de recomendações direcionada a testes de intrusão que objetiva padronizar os testes ao mesmo tempo que procura oferecer flexibilidade ao *tester* e maior eficácia a partir da normatização dos processos. Neste artigo discutimos a relação de um teste de intrusão real com a estratégia de recomendações Tramonto,

Este artigo está organizado da seguinte forma: a Seção 2 descreve os principais conceitos e características de Testes de Intrusão. Já a Seção 3 apresenta a conceituação e formalização das características da estratégia de recomendações Tramonto. A Seção 4 explica detalhadamente o caso real de um teste de intrusão realizado, enquanto a Seção 5 detém a discussão e análise a respeito das principais contribuições da Tramonto direcionadas ao caso apresentado. Por fim, a Seção 6 apresenta as considerações finais deste trabalho.

2. Testes de Intrusão

Hoje em dia, proteger sistemas e redes exige um conhecimento amplo das melhores táticas, ferramentas e motivações do atacante. Conhecer a natureza e as técnicas do atacante melhora a capacidade de evitar ataques bem sucedidos, já que auxilia a verificação de quais as vulnerabilidades existentes que o próprio hacker malicioso pode encontrar.

Testes de intrusão caracterizam a simulação de um ataque a um sistema, rede ou serviço, com o objetivo de comprovar a vulnerabilidade desse sistema e até mesmo o risco de que o alvo possa sofrer um verdadeiro ataque [Geer and Harthorne 2002]. Dessa forma, proteger as redes corporativas envolve mais do que simplesmente estratégias padrão (como gerenciamento de patches, firewalls, e conscientização de usuários), envolve uma frequente validação de como funciona o "mundo real" [Bishop 2007].

A estruturação de um teste de penetração passa por uma série de critérios que permitem que os testes se diferenciem uns dos outros. Essa diferenciação também implica nas variações dos objetivos do teste e, naturalmente, na eficácia da avaliação proposta. Assim, a aplicação do teste de penetração pode ser classificada em critérios como [Whitaker and Newman 2005]:

- Base de informações. Determina qual o nível de conhecimento que o *tester* possui sobre o alvo antes da execução do teste.
- Agressividade. Critério responsável por definir o quão agressivo age o profissional que está efetuando o teste de intrusão. Em resumo, mensura se o *tester* explora as vulnerabilidades por completo, parcialmente, ou não as explora.
- Escopo. Define o escopo do teste de intrusão, determinando quais os sistemas ou cenários serão testados. O escopo implica diretamente no tempo requerido para a execução do teste.
- Abordagem. Trata o método de execução do teste de intrusão quanto à geração de ruído no ambiente. Em geral, a abordagem pode ser dividida em *covert*, onde os procedimentos do teste não são diretamente identificáveis, e *overt*, onde o teste se assemelha às configurações de uma base de informações *white-box*.
- Técnica: Determina quais as técnicas e metodologias serão usadas no teste de intrusão. Em um teste de intrusão convencional, a técnica é baseada em rede, considerando a ideia de que todos os ataques realizados ao sistema são apenas via rede. Um teste de intrusão baseado em rede simula o típico ataque de um hacker malicioso que atua via protocolo TCP/IP [Fonseca et al. 2010][Jajodia et al. 2005].
- *Starting point*. O ponto onde o *tester* conecta seu equipamento para realizar os ataques do teste pode ser tanto de dentro ou de fora da rede ou ambiente físico do alvo.

2.1. Fases

Além dos critérios que diferenciam os testes, existem abordagens diferentes (que geralmente são relacionadas com a metodologia do teste utilizado) em relação às fases pelas quais o teste passa durante a sua aplicação. De maneira abrangente, pode-se delimitar três principais fases para um teste de intrusão: *Pre-Attack phase*, *Attack phase* e *Post-Attack phase*.

2.1.1. Fase 1: *Pre-Attack*

A fase *Pre-Attack* consiste em investigar ou explorar o alvo em potencial. Esta fase é basicamente responsável pela tarefa de obtenção de informações e pode ser dividida em reconhecimento passivo e reconhecimento ativo. Reconhecimento passivo envolve a coleta do maior número de informações sobre um alvo em potencial sem o conhecimento da organização, e sabe-se que muitos dados, por exemplo, sobre serviços que executam e até mesmo sobre a topologia da rede, vazam ou são facilmente conseguidos. O tester pode usar essas informações para traçar um mapa do seu alvo e utilizar tal recurso como apoio de decisões estratégicas em relação às possibilidades de ataque. Ainda nesse sentido, informações públicas que são disponíveis sobre o alvo também apoiam efetivamente o sucesso do teste a ser executado. Nesta fase é primordial manter o registro das atividades realizadas e principalmente dos resultados e informações obtidas. O tester precisa garantir e comprovar o seu trabalho através desses registros, além de comunicar os responsáveis da organização a respeito das suas ações (mediante tipo do teste de intrusão, conforme abordado anteriormente). Enquanto a estratégia de ataque é planejada, devem ser correlacionados as escolhas dos vetores de ataque em relação às entradas e saídas provenientes dessa fase. Como possíveis resultados desta fase, é possível obter informações a respeito

de itens como: localização física e lógica do alvo, conexões analógicas, informações pessoais, e informações sobre outras organizações conectadas com o alvo.

2.1.2. Fase 2: *Attack*

A fase *Attack* lida basicamente com o comprometimento do alvo, onde o *tester* pode explorar as vulnerabilidades descobertas na fase anterior, de forma a obter acesso ao sistema [Igure and Williams 2008][Sarraute et al. 2011]. As principais atividades nesta fase contemplam o núcleo do teste de intrusão e detém a maior parte do teste. Estas atividades podem ser listadas como: teste de perímetro, teste de firewall, teste de aplicações web, acesso ao alvo e escalada de privilégios. Além disso, há a atividade relacionada ao comprometimento efetivo do alvo alcançado através da execução de códigos arbitrários, cujo objetivo é explorar a extensão na qual a segurança falha.

2.1.3. Fase 3: *Post-Attack*

A fase *Post-Attack* contempla as revisões e atividades finais do teste de intrusão realizado, onde o *tester* atua na reconstrução do ambiente avaliado, bem como a restauração dos sistemas e segmentos que sofreram alterações mediante as explorações feitas. Analisando o fluxo do processo de teste, subentende-se que esta fase trata os devidos encaminhamentos dos resultados e de toda a avaliação de segurança efetuada no alvo, relacionando diretamente com o objetivo proposto inicialmente na concepção do teste. Assim, a apresentação dos resultados é a atividade final desta fase e, consequentemente, do teste de intrusão [Line et al. 2008][Bechtsoudis and Sklavos 2012].

Ao longo desta última fase o *tester* é responsável por remover todos os arquivos provenientes do teste, limpar todos os registros, recuperar todas as modificações realizadas em arquivos e alterar todas as configurações para seu estado original. Em contraponto, essa retomada do ambiente alvo para seu estado original passa também pelo processo de tratamento das vulnerabilidades encontradas, mesmo que essa tarefa só seja realmente efetivada após a documentação, apresentação e análise da aplicação do teste como um todo.

2.2. Metodologias

Atualmente, as principais metodologias de teste de segurança são: OSSTMM (*Open Source Security Testing Methodology Manual*), ISSAF (*Information Systems Security Assessment Framework*), PTES (*Penetration Testing Execution Standard*), NIST (*National Institute of Standards and Technology) Guidelines* e OWASP *Testing Guide* [Dalalana Bertoglio and Zorzo 2017].

- **OSSTMM**. OSSTMM (*Open Source Security Testing Methodology Manual*) [Hertzog 2010] é a metodologia que detém um padrão internacional para testes de segurança, mantida pela ISECOM (*Institute for Security and Open Methodologies*). Essa metodologia é basicamente dividida em três classes (COMSEC (*Communications Security Channel*), PHYSSEC (*Physical Security Channel*) e SPECSEC (*Spectrum Security Channel*), que por sua vez, são divididas em cinco canais (Humano, Físico, Sem Fio, Telecomunicações e Redes de Dados).

- **ISSAF**. ISSAF (*Information Systems Security Assessment Framework*) [ISSAF 2006] é um *framework* que gerencia requisitos e diretivas de controle internos para a segurança da informação. É essencialmente constituído de três áreas de execução: planejamento e preparação, avaliação e relatório, limpeza e destruição de artefatos.
- **PTES**. A metodologia PTES (*Penetration Testing Execution Standard*) [Nickerson et al. 2014] é a única direcionada para testes de intrusão e fornece todo o aparato necessário para um *tester*. A estrutura da metodologia é composta por sete fases: *Pre-engagement interactions*, *Intelligence gathering*, *Threat modeling*, *Vulnerability analysis*, *Exploitation*, *Post-exploitation* e *Reporting*.
- **NIST Guidelines**. A metodologia proposta pela NIST (*National Institute of Standards and Technology*) [Stouffer et al. 2008] é apresentada na publicação especial 800-15 como *Technical Guide to Information Security Testing and Assessment* e é construída a partir de quatro etapas principais: **planejamento**, onde o sistema é analisado para encontrar os alvos de teste mais interessantes; **descoberta**, onde o *tester* procura as vulnerabilidades no sistema; **ataque**, onde o *tester* verifica se as vulnerabilidades encontradas podem ser exploradas; e **relatório**, onde cada resultado proveniente das ações realizadas na etapa anterior é reportado.
- **OWASP Testing Guide**. A metodologia proposta no OWASP (*Open Web Application Security Project*) *Testing Guide* [Meucci et al. 2008] tem a sua proposta voltada para o contexto web, e objetiva testar a segurança em softwares e aplicações web. São propostas atividades detalhadas para se avaliar diferentes tipos de riscos e vulnerabilidade na web, e tudo com base nas pesquisas e contribuições gerenciadas pela comunidade juntamente com a OWASP.

3. Tramonto

A partir das metodologias de teste de segurança citadas na Seção 2.2, e dos desafios que estão contidos nos estudos relacionados aos Testes de Intrusão e suas características, foi criada uma estratégia de recomendações chamada Tramonto [Dalalana Bertoglio and Zorzo 2016]. A Tramonto tem como objetivo principal solucionar os problemas relacionados a falta de metodologias específicas para Testes de Intrusão, auxiliando o *tester* no processo do teste. Adicionalmente, a Tramonto também tem por objetivo adequar atividades, planos, processos e fases de forma a oferecer recursos para um teste padronizado, flexível e eficaz.

Inicialmente, para que fosse possível idealizar soluções e definições da estratégia de recomendações Tramonto, foi necessário estabelecer a estrutura básica que determina as etapas pelos quais as atividades do Teste de Intrusão estão contidas, conforme apresentado na Figura 1.

Assim, a estrutura da estratégia de recomendações Tramonto é dividida basicamente em cinco etapas que descrevem o seu fluxo:

- **Adequação**. A etapa inicial da Tramonto é responsável por gerenciar todas as escolhas iniciais do *tester* em respeito às informações de escopo, abordagem, dados do alvo e tipo do teste a ser realizado. Essas definições iniciais são determinantes para o andamento e execução do teste, e precisam da aprovação do administrador ou responsável pelo ambiente alvo. Assim, é alinhado o máximo de

Figura 1. Estrutura da Tramonto.

informações com o intuito de que o plano seja melhor traçado. A partir das escolhas e determinações efetuadas, a Tramonto conduzirá o *tester* para os fluxos de trabalho seguintes, fornecendo todas as escolhas possíveis que são categorizadas de acordo com o andamento do teste. Neste ponto, toda experiência e *know-how* do *tester* é levada em conta já que a estratégia irá recomendar as melhores alternativas, não engessando as escolhas caso o profissional venha a decidir por outro caminho de execução.

- **Verificação**. Mediante as alternativas delimitadas na etapa anterior, a etapa de verificação consiste em efetuar os *checklists* de necessidades, dados, atividades orientadas que foram ou não efetuadas e demais itens relevantes. Essa etapa assemelha-se ao processo de auditoria, considerado também um teste de avaliação de segurança. Nesse sentido, validar as entradas e necessidades do teste permite também que o *tester* avalie suas decisões tomadas na etapa anterior e verifique se as escolhas tomadas inicialmente, aliadas aos checklists gerados, fornecem a base essencial para avaliação do estado de segurança do alvo. Assim, o intuito dessa etapa é minimizar o número de falhas para a continuidade do teste, de forma com que a Tramonto contribua para que o teste seja o mais detalhista possível. Da mesma forma, a etapa de Verificação também objetiva que o teste seja feito sem gerar retrabalhos, o que não significa que o *tester* não possa retomar as suas decisões feitas na etapa de Adequação e modificá-las para um melhor andamento.
- **Preparação**. Envolve a escolha das estratégias de teste de intrusão a serem efetuadas, bem como a indicação de kits de ferramentas de acordo com os processos anteriores. Os kits de ferramentas são um conjunto de soluções e aplicações que são utilizadas nas diversas etapas do teste. Oferecer diferentes possibilidades neste ponto pode permitir ao *tester* experimentar outras ferramentas que não aquelas que o mesmo utiliza trivialmente. A indicação desses kits é baseada em sugestões e opiniões de profissionais de segurança e pesquisas relacionadas. O processo de preparação idealiza fornecer ao *tester* a análise e detalhamento do planejamento e forma de execução do teste. Nesta etapa a Tramonto permite que as soluções de aplicação do teste de intrusão estejam suficientemente elaboradas e com as devidas prescrições mediante os dados provenientes das etapas anteriores.
- **Execução**. Trata o núcleo principal de execução do teste de intrusão. Nesta etapa, a Tramonto fornece todo o aparato em relação aos vetores de ataque, que são os possíveis caminhos utilizados pelo *tester* para realizar as intrusões. Os vetores de ataque refletem o planejamento do tipo de ataque que é efetuado, podendo esse ser baseado em computadores (por meios tecnológicos) ou baseado em pessoas (caracterizados pelo contato direto). Além disso, são listados também os possíveis resultados a serem obtidos de acordo com as ações e demais informações relaci-

onadas. É essencial ressaltar que, nesta etapa, os fluxos oferecidos como alternativas de execução do teste devem estar filtrados de acordo com as informações e verificações das etapas anteriores. Essa funcionalidade permite que o *tester* otimize o tempo do teste mediante suas escolhas anteriores, além de induzir atividades mais precisas sobre o escopo delimitado.

- **Finalização**. A última etapa proposta na estrutura da Tramonto contempla as ações relacionadas com a elaboração dos relatórios a serem fornecidos ao cliente. Ao mesmo tempo, como característica adicional, a construção de um relatório destinado ao próprio *tester* é um dos itens que a estratégia de recomendações produz ao término do teste, permitindo a criação de um padrão que, posteriormente, pode vir a fornecer as possibilidades de comparação entre os resultados obtidos. Ainda nesse processo são tratadas as atividades de cobertura de rastros, limpeza de registros e controle de estado dos sistemas e aplicações alvo.

É possível ainda descrever um sexto processo que descreve uma avaliação, oferecendo alternativas baseadas no resultado obtido e no resultado desejado inicialmente. Nesse sentido, é estabelecida uma nova funcionalidade específica da Tramonto chamada de Plano Alternativo (PA), responsável por fornecer atividades, fluxos, ferramentas, estratégias e todo o tipo de características iminentes que, dependendo das escolhas e ações do *tester*, trazem novas possibilidades para o fluxo processual do teste de intrusão. Dessa forma, cada uma das etapas estabelecidas pode conter planos alternativos que são identificados pelo nome do processo, seguido de um identificador (por exemplo, *PAP1* refere-se a um plano alternativo na etapa de preparação e detém um *ID* 1). Assim, compreende-se que devido a vasta quantidade de informações diferentes contidas nas metodologias analisadas, podem ser criados diversos sub-planos de acordo com as informações iniciais do teste.

Para o processo do teste, de acordo com a estrutura da Tramonto, cada uma das etapas requer uma ou mais entradas e gera uma ou mais saídas. A partir disso, os planos alternativos podem ser melhor determinados, uma vez que a estratégia controla e lista as entradas e saídas. Na etapa de Adequação, os artefatos de saída são determinantes para a construção da etapa seguinte, de Verificação. São gerados os documentos comprobatórios do teste e o levantamento de requisitos estipulados no planejamento realizado na Adequação. Para a etapa de Verificação, diante dessa entrada, a saída é a enumeração e listagem de todos os itens que necessitam ser verificados para que o teste ocorra da forma prevista. Na etapa de Preparação os artefatos de saída são relacionados com aspectos mais técnicos, considerando que são esperadas as estratégias e técnicas cabíveis para a execucação do teste requerido. A etapa de Execução, por sua vez, retrata a maior quantidade de artefatos em sua saída por lidar com o núcleo do teste. Isso também é ressaltado pelo fato de que as vulnerabilidades encontradas podem gerar novos vetores de ataque, e isso mantém a ideia do fluxo contínuo do teste, com frequente reavaliação do que foi testado. Por fim, a etapa de Finalização, mediante entrada de todos os artefatos obtidos nas etapas anteriores, gera de saída os relatórios, tanto para o cliente como para o *tester*.

4. Caso de Teste de Intrusão

Para a análise de um teste de intrusão a partir da estratégia de recomendações Tramonto, esta seção apresenta um caso real de teste aplicado a um cenário organizacional. Informações sensíveis e identificáveis estão omitidas, conforme orientação do acordo de

confidencialidade (NDA - *Non-disclosure Agreement*) determinado juntamente ao *tester* responsável pela aplicação deste teste. A apresentação do teste segue nas próximas subseções com a divisão entre: sumário executivo, narrativa das ações de intrusão e análise/recomendações.

4.1. Sumário Executivo

A aplicação do teste de intrusão na empresa alvo foi determinada para avaliar sua exposição a um ataque direcionado, considerando a ação de um ataque externo e simulando o comportamento de um atacante. O teste busca identificar se o atacante, de maneira remota, consegue invadir e ultrapassar as defesas da empresa alvo, além de determinar o impacto de violações de segurança relacionadas a confidencialidade dos dados privados da empresa e também a infra-estrutura interna/disponibilidade dos sistemas de informação.

As atividades executadas ao longo do teste foram focadas principalmente na identificação e exploração das deficiências de segurança que poderiam permitir ao atacante remoto obter acesso não autorizado a dados organizacionais. Os ataques foram conduzidos com o nível de acesso que um usuário comum da Internet teria. O processo do teste de intrusão e da avaliação de segurança seguiu com metodologias NIST e OSSTMM, com todos os testes e ações sendo conduzidos em condições controladas.

4.2. Narrativa das Ações de Intrusão

Para o início do teste de intrusão, a empresa forneceu o mínimo de informações a respeito da mesma, apenas o nome do domínio. Para não atingir sistemas terceirizados ou que simplesmente não fazem parte da propriedade da empresa, todos os ativos identificados foram submetidos para verificação de propriedade antes que quaisquer ataques fossem realizados.

Como primeira ação no teste, foram identificados os *name servers* do domínio através do *nslookup*. A partir disso, foi realizada uma tentativa de transferência de zona, efetuada com sucesso no *name server* 2 que era vulnerável. Assim, foi possível obter uma lista de hosts e endereços IP associados.

Antes da avaliação dos hosts identificados, a lista desses hosts foi verificada juntamente a empresa alvo para permitir e incluir os IPs no escopo de avaliação. Foram realizadas varreduras nos hosts para enumerar os serviços e determinar a potencial exposição desses serviços a um ataque direcionado. Através de uma combinação de outras técnicas de enumeração de DNS e demais varreduras realizadas na rede, foi possível idealizar a topologia e a infra-estrutura da empresa alvo, conforme apresentado na Figura 2.

Um dos vetores de ataque determinado a partir das varreduras foi a tentativa de intrusão do servidor web Apache na porta 80. Acessando a URL do endereço do servidor, e através da enumeração do sistema procurando diretórios e arquivos comuns, foi identificado o endereço **/portaladmin** que exigia a autenticação. Assim, para passar a autenticação foi efetuado um ataque de força bruta que utilizou uma *wordlist* criada especificamente para o alvo, envolvendo a combinação de palavras e termos relacionados com a empresa (totalizando 14.214 palavras). A senha foi descoberta com sucesso e o acesso a parte administrativa do site permitiu também o acesso ao gerenciador do banco de dados. Após esta etapa do teste, a interface forneceu acesso direto aos dados e a capacidade de extrair uma lista de usuários no sistema com os valores de hash de senha associados.

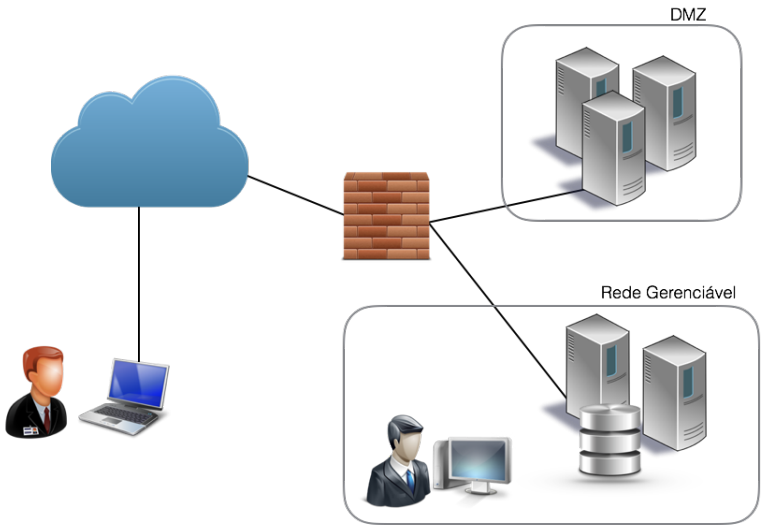


Figura 2. Topologia reconstituída a partir do teste.

Considerando o gerenciador de banco de dados vulnerável a injeção de código, a exploração bem-sucedida resultou em acesso de *shell* ao sistema alternativo relacionado ao usuário do servidor web. Através do uso de um exploit público, foi possível obter acesso remoto ao domínio **admin** do servidor web.

Com o acesso remoto obtido, o teste buscou a escalada de privilégios ao nível administrativo. Através de um novo exploit, o sistema mostrou-se vulnerável e foi possível obter as credenciais de administrador. Com a capacidade de obter acesso administrativo completo, uma parte mal-intencionada poderia utilizar este sistema vulnerável para ataques contra a própria empresa e até ataques contra os clientes relacionados.

De posse do acesso administrativo ao sistema, descobriu-se uma seção privada do site da empresa que fornece um serviço específico a um subconjunto de usuários internos. Dessa forma, foi possível estabelecer um caminho para chegar a esses usuários internos. Para tal, a aplicação de uma técnica de engenharia social, como *phishing*, foi a atividade determinada para simular esse serviço específico e manipular os usuários para que interagissem com o mesmo.

Como resultado dessa manipulação e do vetor de ataque, o acesso foi limitado ao nível de um usuário padrão. Para analisar realmente o impacto do ataque, foi necessário buscar o acesso ao nível de administrador de domínio. Em um primeiro momento foi necessário, através das diretivas de grupos no sistema operacional, conseguir acesso como administrador local.

Uma vez como administrador local, foram realizadas tentativas de conexão de sessões utilizando SSH, que foram efetuadas com sucesso. Neste sentido, o perímetro externo da rede da empresa estava totalmente comprometido. Por fim, o acesso como administrador de domínio foi alcançado via acesso ao servidor de posse das credenciais de administrador local.

4.3. Resumo dos Resultados

O reconhecimento inicial da rede da empresa resultou na descoberta de um servidor DNS mal configurado que permitia uma transferência de zona DNS. Os resultados listaram uma ordem de hosts específicos que foram alvos do teste. A partir de uma avaliação desses hosts foi possível identificar uma interface administrativa do webserver. Na tentativa de intrusão do webserver, foi criada uma *wordlist* com palavras e termos relacionados ao contexto da empresa alvo, para a realização de um ataque de quebra de senha por meio de força bruta.

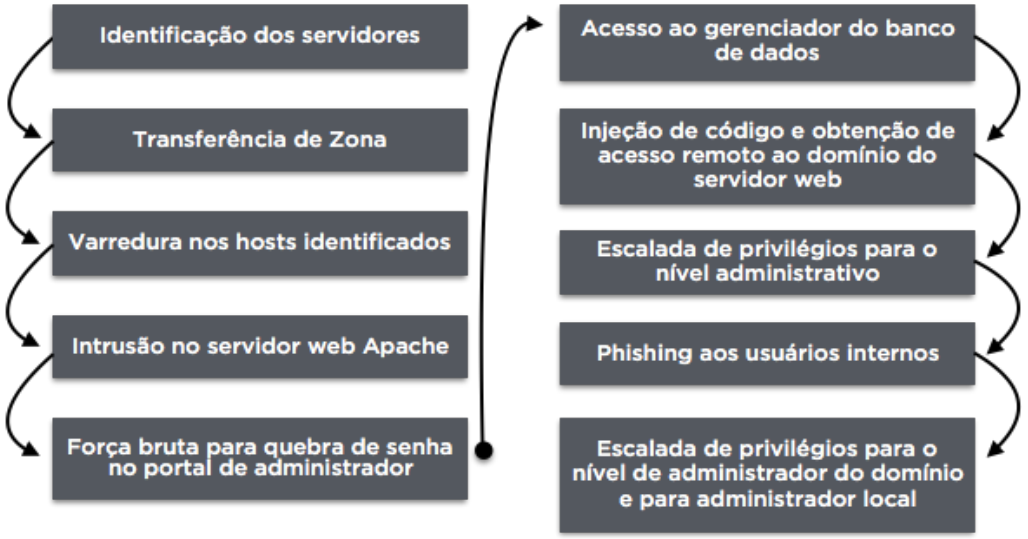


Figura 3. Resumo do teste de intrusão efetuado.

Após a obtenção da senha, uma análise da interface administrativa permitiu a identificação de vulnerabilidades de injeção de código remoto. Essa técnica de injeção foi utilizada para obter acesso a um sistema de produção conectado a esse webserver. Após isso, foi escalado o privilégio para um acesso como administrador em virtude da falta de atualizações desse sistema.

Uma vez com o acesso de administrador e dentro do servidor web, foi possível recuperar diversos usuários e senhas e alcançar recursos internos a partir desse acesso, que inicialmente era limitado. Com base nisso, o teste resultou no acesso como administrador local de hosts internos, ao completo comprometimento de um servidor de acesso remoto e também ao controle administrativo completo da infraestrutura do *Active Directory*.

4.4. Análises e Principais Recomendações

Baseado no teste de intrusão realizado, as principais análises e recomendações estão voltadas a dois grandes tópicos:

- Adoção de procedimentos padrão: é necessário, naturalmente, estar atento ao uso de credenciais fortes em todo o contexto organizacional. A intrusão realizada na empresa alvo apresentou relação diretada com o uso de senhas fracas e da falta de diferentes ações e níveis de segurança. Além disso, a empresa poderia, como parte do gerenciamento dos seus riscos, manter um controle sobre as avaliações e testes de segurança realizados com o intuito de padronizar e verificar constantemente o estado de segurança do seu ambiente.

- Implementação de recursos adicionais de segurança: estabelecer limites de confiança e acesso na rede interna, evitando efeito cascata quando do comprometimento da rede ou sistema por parte de um atacante. Da mesma forma, a implementação de controles e verificações de mudança nos principais sistemas, permitindo gerenciar erros de configuração e demais problemas direcionados a falta de tratamento correto. Por fim, implementar também uma organização para gerenciar atualizações e patches em geral, de acordo com alguma metodologia específica. Esses fatores são essenciais para atenuar e mitigar potenciais ataques como o que foi simulado pelo teste de intrusão realizado.

5. Tramonto x Caso de Teste de Intrusão

Após a compreensão da descrição geral da Tramonto e também do caso de Teste de Intrusão, esta seção confronta os itens vistos com o intuito de verificar a ligação entre ambos e analisar as ações realizadas no teste de acordo com o que a Tramonto propõe.

5.1. Organização

A estrutura do teste de intrusão apresentado na seção anterior segue, de forma sucinta, o processo padrão desse tipo de teste: coleta de informações, reconhecimento, exploração e pós-exploração. É importante ressaltar também que o caso cita que seu processo de teste foi baseado em duas metodologias, NIST e OSSTMM. Contudo, não há uma apresentação detalhada do que ocorre ao longo do teste em confronto com as definições fornecidas por essas metodologias.

Considerando a estrutura da Tramonto, as atividades descritas na Seção 4 podem ser divididas da seguinte forma:

- Adequação: Estabelecimento das informações iniciais (apenas o nome do domínio) e preocupação em não atingir sistemas terceirizados ou que simplesmente não fazem parte da propriedade da empresa.
- Verificação: Confirmação dos sistemas que não poderiam ser atingidos e da lista de hosts que foi obtida após o processo de enumeração.
- Preparação: Não foram listadas informações que se encaixam nessa etapa.
- Execução: Processo de enumeração dos *name servers* do domínio através do *nslookup*, transferência de zona e varreduras nos hosts para enumerar os serviços. Além disso, todos as intrusões, vetores de ataque e descobertas realizadas, contando também o ataque de força bruta para descobrir as senhas.
- Finalização: As recomendações e análises feitas para mitigar e adicionar controles a partir do teste efetuado, além do resumo do que foi realizado ao longo do teste.
- Avaliação: Basicamente, a alternância de vetores de ataque conforme os artefatos que iam surgindo com exploração realizada com sucesso. Não seria possível avaliar o impacto real de um ataque no caso de teste proposto se não houvesse uma continuidade de exploração por parte do *tester*.

5.2. Limitações

É necessário considerar a omissão de informações como os estabelecimentos contratuais inicias realizados juntamente a empresa alvo. Isso impacta diretamente em uma avaliação não tão completa sob a ótica da Tramonto. De qualquer maneira, o fato da empresa ter

informado apenas o nome do domínio fornece o entendimento essencial para a etapa de Adequação da Tramonto.

Da mesma forma, a falta de uma explicação detalhada sobre as ferramentas utilizadas no teste também limitam o confronto com a Tramonto. Contudo, pode-se considerar isto como uma vantagem da Tramonto em relação ao caso de teste apresentado, uma vez que a estratégia se preocupa em apresentar claramente o kit de ferramentas idealizado para o cenário.

Por mais que o teste tenha seguido uma parte das metodologias NIST e OSSTMM, conforme informado pelo *tester*, há uma falta de padronização para o prosseguimento do fluxo do teste, no qual aparece quando não existe um planejamento prévio do caminho executado no caso em questão. O conflito entre padronizar um teste por completo e permitir uma atuação autônoma do *tester* é um problema no qual a Tramonto, desde sua idealização, procura resolver.

5.3. Principais contribuições da Tramonto para o Caso de Teste

Inicialmente, é possível discutir a atuação da Tramonto a partir da apresentação do caso. Os relatórios que a Tramonto propõe como saída final dos testes apresentam um resumo geral de cada detalhe da avaliação de segurança executada: ferramentas, técnicas, soluções e alvos definidos. Além disso, a contribuição de oferecer um relatório com aspecto mais técnico para o *tester* que efetuou o teste de intrusão também é um item a ser considerado de maneira relevante.

Ao início do teste de intrusão realizado, não há determinação dos rumos possíveis que o mesmo pode deter a partir do que é informado pela empresa. Mesmo que isso possa ser refutado em virtude do estabelecimento de um teste do tipo *black-box* (foi fornecido apenas o nome do domínio), seria interessante possibilitar ao *tester* as principais tarefas iniciais a serem realizadas. Como se sabe, uma das principais contribuições da Tramonto é permitir alternativas e sugerir, durante as etapas, atividades devidamente planejadas.

Em determinado momento do teste, mesmo que em caráter de tipo *black-box*, o *tester* ratifica com o responsável da empresa uma lista de hosts que foram identificados. Esse processo de verificação se encaixa diretamente em uma das etapas da Tramonto, e poderia ter sido feito com melhor organização, gerando uma saída para a etapa seguinte e conteúdo direto para o relatório final. O escopo, mesmo que bem definido, poderia ter indicado ao *tester* que alguns procedimentos implementados fossem previamente verificados para tornar o teste melhor constituído. Assim, quando da enumeração e da obtenção de informações, a estratégia permitiria um controle maior do *tester* sobre suas ações.

Não há, no caso apresentado, uma discussão sobre a conformidade do teste e seus aspectos legais. A Tramonto, antes do início do teste, exige a listagem de aspectos contratuais, sendo necessário delimitar em contrato qual o teste será efetuado, os limites que a intrusão não pode ultrapassar ou alcançar (há uma breve afirmação de que sistemas de terceiros não podem ser atingidos no caso), o acordo de confidencialidade e por fim, os custos envolvidos em toda a operação.

Ademais, a Tramonto poderia contribuir também através da explicação e indicação de estratégias e técnicas para o teste de intrusão, atividade que ocorre na primeira etapa da Tramonto. Essas possibilidades são criadas com o intuito de auxiliar o teste, permitindo

que o *tester* possa relatar suas preferências. Contudo, cabe ressaltar que essas estratégias e técnicas, para a Tramonto, são determinadas não só pelos padrões indicados pelas metodologias que constituem a Tramonto, mas também através de testes de intrusão anteriores. É importante considerar, neste ponto, que não há informações de outros testes realizados pelo mesmo *tester* do caso relatado. Com base nisso, seria possível que avaliações de técnicas de segurança a partir da rede interna e de teste de segurança físico fossem consideradas para o caso, já que ao término no mesmo percebeu-se as ações como usuário da rede interna, por exemplo.

6. Considerações Finais

A execução de testes de intrusão apresenta particularidades e detalhes que fazem com que esse tipo de teste promova uma série de desafios tanto para quem o executa (*tester*) como para o alvo. Inicialmente, percebe-se que os inúmeros cenários, controles e recursos que podem ser estabelecidos como alvos permitem uma ampla variação nas atividades que são desenvolvidas no teste de intrusão. Além disso, sabe-se que a tentativa de padronização dessas atividades tem sido discutida e encarada como uma problemática nas pesquisas recentes envolvendo a área de testes de segurança. A Tramonto, desde a sua concepção, é uma estratégia de recomendações que procura atender os requisitos necessários para a aplicação de testes de intrusão mas também idealiza, e objetiva, a flexibilidade do *tester*. Esse quesito é um aspecto essencial para a autonomia do teste, ao mesmo tempo em que são fornecidos procedimentos e atividades padrão para os diversos alvos possíveis. No estudo de caso previamente discutido nesse trabalho, alguns traços normatizam o plano do teste de intrusão, mas há uma carência na formatação desse teste. É possível afirmar que esse estudo de caso discute detalhes que comumente são vistos em grande parte dos testes de intrusão: falta de detalhamento técnico dos procedimentos, discussão superficial sobre os vetores de ataque encontrados e novas possibilidades de extração de conhecimento a partir do teste efetuado. A partir da análise deste estudo de caso surgem novas possibilidades para os trabalhos futuros, envolvendo outros casos específicos de cenários pré-determinados que representem as principais vulnerabilidades em algum recurso e/ou ativo. Assim, a Tramonto procura, dentre outros objetivos, tratar as problemáticas relacionadas aos testes de intrusão e seus demais aspectos.

Referências

Bechtsoudis, A. and Sklavos, N. (2012). Aiming at higher network security through extensive penetration tests. *IEEE Latin America Transactions*, 10(3):1752–1756.

Bishop, M. (2007). About penetration testing. *IEEE Security & Privacy*, 5(6):84–87.

Dalalana Bertoglio, D. and Zorzo, A. F. (2016). Tramonto: Uma estratégia de recomendações para testes de penetração. In *Simpósio Brasileiro de Segurança de Sistemas*, SBSeg, pages 366–379. SBC.

Dalalana Bertoglio, D. and Zorzo, A. F. (2017). Overview and open issues on penetration test. *Journal of the Brazilian Computer Society*, 23(1):1–16.

Fonseca, J., Vieira, M., and Madeira, H. (2010). The web attacker perspective - a field study. In *2010 IEEE 21st International Symposium on Software Reliability Engineering*, pages 299–308.

Geer, D. and Harthorne, J. (2002). Penetration testing: a duet. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 185–195.

Henry, K. M. (2012). *Penetration Testing: Protecting Networks and Systems*. IT Governance Publishing, UK.

Hertzog, P. (2010). *OSSTMM - Open Source Security Testing Methodology Manual*. Institute for Security and Open Methodologies (ISECOM).

Igure, V. M. and Williams, R. D. (2008). Taxonomies of attacks and vulnerabilities in computer systems. *IEEE Communications Surveys Tutorials*, 10(1):6–19.

ISSAF (2006). Information systems security assessment framework.

Jajodia, S., Noel, S., and O'Berry, B. (2005). *Managing Cyber Threats: Issues, Approaches, and Challenges*, chapter Topological Analysis of Network Attack Vulnerability, pages 247–266. Springer US, Boston, MA.

Lam, K., LeBlanc, D., and Smith, B. i. (2004). *Assessing network security*. Redmond, Wash. Microsoft Press, Washington, USA.

Line, M. B., Jaatun, M. G., Cheah, Z. B., Faruk, A. B. M. O., Garnes, H. H., and Wedum, P. (2008). *Ubiquitous Intelligence and Computing: 5th International Conference, UIC 2008, Oslo, Norway, June 23-25, 2008 Proceedings*, chapter Penetration Testing of OPC as Part of Process Control Systems, pages 271–283. Springer Berlin Heidelberg, Berlin, Heidelberg.

Meucci, M., Keary, E., and Cuthbert, D. (2008). *OWASP Testing Guide v.3*. OWASP Foundation.

Nickerson, C., Kennedy, D., Smith, E., Rabie, A., Friedli, S., Searle, J., Knight, B., Gates, C., and McCray, J. (2014). *Penetration Testing Execution Standard*. PTES.

Sarraute, C., Richarte, G., and Lucángeli Obes, J. (2011). An algorithm to find optimal attack paths in nondeterministic scenarios. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, AISec '11, pages 71–80, New York, NY, USA. ACM.

Stouffer, K., Falco, J., and Scarfone, K. (2008). *NIST SP 800-115: Technical Guide to Information Security Testing and Assessment*. National Institute of Standards and Technology, Maryland, USA.

Whitaker, A. and Newman, D. (2005). *Penetration Testing and Cisco Network Defense*. Cisco Press, Indianapolis, USA.

Realização

Apoio Fomento

Apoio Institucional

Patrocinador Diamante

Núcleo de Informação e Coordenação do Ponto BR

Comitê Gestor da Internet no Brasil

Patrocinador Ouro

Patrocinador Bronze