Frontmatter

Editors

SIGAPP FY’14 Quarterly Report

A Message from the Editor

S. Shin

S. Shin

H. Haddad

SAC 2015 Preview

Selected Research Articles

Opinions of People: Factoring in Privacy and Trust

A. Basu, J. Vaidya, J. Corena,
S. Kiyomoto, S. Marsh, G.
Guo, J. Zhang, and Y. Miyake

Hardware-Software Collaboration for Secure Coexistence
with Kernel Extensions

D. Oliveira, N. Wetzel, M.
Bucci, J. Navarro, D. Sullivan,
and Y. Jin

Early Response Markers from Video Games for
Rehabilitation Strategies

R. Davison, S. Graziadio, K.
Shalabi, G. Ushaw, G. Morgan,
and J. Eyre

Deployment and Activation of Faulty Components at
Runtime for Testing Self-Recovery Mechanisms

K. Gama and D. Donsez

Error Control Based Energy Minimization for Cooperative
Communication in WSN

S. Ahmed, D. Kim, S. Bouk,
and N. Javaid
SIGAPP FY’14 Quarterly Report

July 2014 – September 2014
Sung Shin

Mission

To further the interests of the computing professionals engaged in the development of new computing applications and to transfer the capabilities of computing technology to new problem domains.

Officers

<table>
<thead>
<tr>
<th>Role</th>
<th>Name</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chair</td>
<td>Sung Shin</td>
<td>South Dakota State University, USA</td>
</tr>
<tr>
<td>Vice Chair</td>
<td>Jiman Hong</td>
<td>Soongsil University, South Korea</td>
</tr>
<tr>
<td>Secretary</td>
<td>Michael Schumacher</td>
<td>University of Applied Sciences Western Switzerland, Switzerland</td>
</tr>
<tr>
<td>Treasurer</td>
<td>Tei-Wei Kuo</td>
<td>National Taiwan University, Taiwan</td>
</tr>
<tr>
<td>Webmaster</td>
<td>Hisham Haddad</td>
<td>Kennesaw State University, USA</td>
</tr>
<tr>
<td>Program Coordinator</td>
<td>Irene Frawley</td>
<td>ACM HQ, USA</td>
</tr>
</tbody>
</table>

Notice to Contributing Authors

By submitting your article for distribution in this Special Interest Group publication, you hereby grant to ACM the following non-exclusive, perpetual, worldwide rights:

- to publish in print on condition of acceptance by the editor
- to digitize and post your article in the electronic version of this publication
- to include the article in the ACM Digital Library and in any Digital Library related services
- to allow users to make a personal copy of the article for noncommercial, educational or research purposes

However, as a contributing author, you retain copyright to your article and ACM will refer requests for republication directly to you.
A Message from the Editor

I hope all of you had a great summer, and I am happy to release the fall issue of Applied Computing Review. This issue includes four selected papers presented at the 29th ACM SAC (Symposium on Applied Computing) conference, and one selected paper from the 2013 ACM RACS (Research in Adaptive and Convergent Systems). All the selected papers have been revised and expanded for inclusion in ACR, and I can proudly tell you that each and every one of them maintains high quality.

I am grateful to the authors for contributing the state-of-the-art methods in their research area and to the highly qualified editorial board members who coordinated an outstanding lineup of technical paper reviews. ACR is available to everyone who is interested in the modern applied computing research trends. Our goal is to provide you with a platform for sharing innovative thoughts among professionals in various fields of applied computing.

I would like to take this opportunity to announce that the 5th ACM BCB (Bioinformatics, Computational Biology, and Health Informatics) conference will be held in Newport Beach, CA, from September 20th to 23rd. You will find more information about BCB below. Most importantly, I would like to remind you once again that the 30th ACM SAC will be held in Salamanca, Spain. Please join us and enjoy the conference. See you in Salamanca.

Sincerely,

Sung Shin
Editor in Chief & Chair of ACM SIGAPP

ACM BCB 2014

BCB is the main conference of the ACM SIGBio (Special Interest Group on Bioinformatics, Computational Biology, and Biomedical Informatics). BCB provides a forum for interdisciplinary and multidisciplinary research encompassing disciplines of computer science, mathematics, statistics, biology, bioinformatics, and health informatics. Details about the conference can be found at http://www.cse.buffalo.edu/ACM-BCB2014/.

Next Issue

The planned release for the next issue of ACR is December 2014.
SAC 2015 Preview

The 30th Annual Edition of the ACM Symposium on Applied Computing (SAC) will be held in Salamanca, Spain, on the campus of Salamanca University. The conference starts on Monday April 13th with the Tutorials Program and continues with the Technical Program till Friday April 17, 2015. The Student Research Competition (SRC) Program is planned for Tuesday (display session) and Wednesday (presentations session), respectively, and the Posters Program will take place on Thursday. The paper submission system is now open for all tracks at http://www.acm.org/conferences/sac/sac2015/. Details of paper submissions, acceptance rate, and country of participation will be reported in the next issue of ACR.

The conference will include the third annual Student Research Competition (SRC) program. SRC is designed to provide graduate students the opportunity to meet and exchange ideas with researchers and practitioners in their areas of interest. Active graduate students seeking feedback from the scientific community on their research ideas are invited to submit research abstracts of their original un-published and in-progress research work in areas of experimental computing and application development related to SAC 2015 Tracks. Accepted research abstracts will be published in the conference proceedings (both CD and printed book). Authors of accepted abstracts are eligible to apply for the SIGAPP Student Travel Award Program (STAP). A designated committee will judge the display and presentations of abstracts and select the top three winners for medallions and cash prizes ($200, $300, and $500) provided by ACM. The winners will be recognized during the banquet event. We encourage you to share this information with your graduate students.

The local organizing committee is composed of Professor Juan Manuel Corchado (University of Salamanca), Professor Javier Bajo (Technical University of Madrid), and Professor Sara Rodriguez (University of Salamanca). To facilitate attendees travel, the local organizing committee will provide plentiful of information and help. Once finalized, details will be posted on SAC 2015 website. As the planning is underway, we are excited to have SAC 2015 in Salamanca - A UNESCO Declared World Heritage City. We invite you to join us this April to meet other attendees, enjoy the conference programs, and have a pleasant stay in Salamanca. We hope to see you there.

On Behalf of SAC Steering Committee,

Hisham Haddad
Member of the Steering Committee
Member of SAC 2015 Organizing Committee
Opinions of People: Factoring in Privacy and Trust

Anirban Basu, Jaideep Vaidya, Juan Camilo Corena, Shinsaku Kiyomoto, Stephen Marsh, Guibing Guo, Jie Zhang, Yutaka Miyake

ABSTRACT
The growth of online social networks has seen the utilisation of these network graphs for the purpose of providing recommendations. Automated recommendations, however, do not take into account inter-personal trust levels that exist in a social network. In this article, we propose a privacy-preserving trusted social feedback (TSF) scheme where users can obtain feedback on questions from their friends whom they trust. We show that the concept can be extended to the domain of crowdsourcing – the trusted crowdsourcing (TCS) scheme. In crowdsourcing, instead of asking friends, one can solicit opinions from experts in the crowd through a privacy preserving trusted feedback mechanism. Our proposal supports categorical answers as well as single-valued numerical answers. We evaluate our proposals in a number of ways: based on a prototype implementation built atop the Google App Engine, we illustrate the performance of the trusted social feedback. In addition, we present a user study to measure the impact that our trusted social feedback proposal has on users’ perception of privacy and on foreground trust. We also present another user study to capture a model for user acceptance testing of the trusted crowdsourcing.

Categories and Subject Descriptors
H.3.3 [Information Search and Retrieval]: Information Filtering

General Terms
Algorithms, Human Factors, Security

Keywords
privacy, trust, social network, crowd sourcing, recommendation, category

1. INTRODUCTION

Copyright is held by the authors. This work is based on an earlier work: SAC’14 Proceedings of the 2014 ACM Symposium on Applied Computing. Copyright 2014 ACM 978-1-4503-2469-4/14/03. http://dx.doi.org/10.1145/2554850.2554860.

Social aware recommendation – a recent phenomenon in recommender systems – has made use of online social networks to provide better, arguably more accurate, recommendations. However, many such automated recommender systems fail to consider the inter-personal context-sensitive trust that exists between individuals in the social network, which can affect the way recommendations are made and interpreted. We see social aware recommendation from a different angle, one that is modelled after the word-of-mouth concept in a human society. We also show that our work can be generalised and is applicable to crowdsourcing. We envisage that inter-personal trust is sensitive information, which the truster may wish to keep private; thus requiring the social aware recommendation mechanisms to be privacy-preserving.

We postulate that the strength of a social relation is often one’s asymmetric personal perception of another in a particular context that changes over time. We refer to this as trust in this paper. The asymmetric nature of personal perception means that a’s trust in b is likely to be different from b’s trust in a. This should affect the way one believes a recommendation from a friend. Recommendations based on a community of opinions do not generally consider this interpersonal and contextual trust. In tune with this understanding of trust and modelling after the real society, the user asks for the aggregate feedback from her friends, in her social network, regarding the item of interest. She attaches a certain level of contextual trust to each friend that she asks the question. This non-automated second stage is what we call the trusted social feedback (TSF). Assuming that such a recommender system will be deployed on a cloud, the TSF proposal must be privacy preserving. Building on this idea, one can ask a group of domain experts (instead of friends), for feedback on questions. We explore this with what we term as trusted crowdsourcing (TCS). The TCS proposal also caters for privacy. Furthermore, we extend both TSF and TCS proposals to support multi-valued categorical answers instead of single-valued numerical answers.

Note that automated personal trust transitivity – an idea closely related with the use of trust in the context of social network graphs – is debatable and subjective. It exists but modelling it is difficult. Josang et al. in [21] go as far as
saying “[...] all mathematical operators for trust transitivity proposed in the literature must be considered ad hoc; they represent attempts to model a very complex human phenomenon as if it were lendable to analysis by the laws of physics”. The authors propose a radically different interpretation of trust transitivity based on subjective logic. The authors observe that in order for transitivity to function, the advisor must, in some way, communicate his/her trust on the trust target to the originator relying party. Thus, in our proposals, we rule out automatic estimation of propagated trust.

The rest of the paper is organised as follows. Before delving into describing our proposal, we present the state-of-the-art in section 2 about recommendations using collaborative filtering as well as question-answer services related to privacy and trust; and privacy aware crowdsourcing. Then, we describe the trusted social feedback (TSF) proposal in sub-section 3.1, the trusted crowdsourcing (TCS) proposal in sub-section 3.2 and the capability of both these proposals to handle multi-valued categorical answers in sub-section 3.3. A security analysis is presented in section 4 and the evaluation results on our model in section 5 before concluding in section 6.

2. THE STATE-OF-THE-ART

Herlocker et al.’s work [16] is one of the older works on automated collaborative filtering algorithms. Golbeck’s work [12] on FilmTrust utilised trust in social networks for movie recommendations. Guo’s work [15] is the closest to ours in the way they combined opinions of neighbours in a social network, weighted by trust values. Unlike our proposal, the paper used the concept of trust propagation and it does not preserve privacy in the aggregation process. Trust propagation is a hard-to-model subjective concept. Two recent proposals: [21] and [27] describe interesting ways of looking at trust propagation. Jamali and Ester [20] employed matrix factorisation to deduce trust propagation, which was then used in collaborative filtering. TidalTrust [13] and MoTeTrust [26] are similar with the latter considering ratings from a maximum depth only, in a breadth first search over a trust network to compute a prediction. In [28], authors suggested that the traditional emphasis on user similarity in recommender systems was overstated, and proposed two trust based recommender system solutions. TrustWalker [19] used a random walk method to combine item-based collaborative filtering with trust-based recommendation.

Privacy preserving collaborative filtering (PPCF) has been studied by many [3, 4, 7, 5, 33, 32, 14]. In the context of medical recommendation systems, authors in [17] propose a privacy-friendly architecture where patients submit their ratings in a protected form and the computation of recommendations is done using secure multiparty computation techniques. Existing work can be classified into using either cryptographic or perturbation techniques to preserve privacy. Very few of these proposals have been tested on real world cloud platforms. Canny’s work [6] utilised factor analysis and homomorphic encryption for PPCF; while in [14], the authors computed PPCF from a combination of random perturbation and secure multiparty computation. Polat’s works [31, 33] have used randomisations to preserve privacy. Several question-answer services exist, including commercial ones, such as Yahoo! Answers, Aardvark. Fleming [10, 11] proposed a privacy enhanced question-answer system based on stigmergic routing where privacy is provided by plausible deniability in a decentralised network.

In [34], the author focuses on the problem of reliability of the work submitted to anonymous members of paid microtask crowdsourcing platforms (e.g., Amazon Mechanical Turk) as well as the privacy of the input given to them. The solution involves generating perturbations of the data to preserve anonymity and a majority voting rule to determine the correct output. Other works involved in achieving reliability for human computations can be found in [22, 23]. A line of research more akin to ours is presented in [35], where the authors introduce a framework to preserve user privacy in geographic crowdsourcing applications by performing user anonymization and delayed upload of information. The limited existing work in privacy tends to focus on privacy of the questions themselves instead of the privacy of the interpersonal trust and that of responses, which are what we concentrate on. In [18], the authors identify two different concerns for privacy in reputation systems and propose a solution based on electronic cash technology and designated verifier proofs.

The field of social aware recommendation is relatively new in comparison with traditional recommender systems. In [1], we developed the first privacy-preserving solution for trusted social feedback. In this article we expand on that work and apply it to the problem of crowdsourcing. Our prior work was limited to single valued numeric responses, where as the current article extends the work to multivalued responses as well, which is much more realistic. We also expand on the security analysis, and perform a completely new evaluation of privacy and user acceptance. Our proposals are about trust empowerment because we see trust as an idiosyncratic, context sensitive, neither entirely rational nor subjective feeling that changes over time [9]. Our privacy preserving solutions also consider privacy from a user-centric perspective.

3. OPINIONS OF PEOPLE

In this section, we describe two methods of obtaining opinions from people: (a) asking trusted friends and (b) asking domain experts through crowdsourcing.

3.1 Trusted social feedback: finding out what friends think

Modelled after a likeness of the word-of-mouth in a human society, the user asks people in her virtual social network for a trusted social feedback (TSF) on a query. For the sake of simplicity at this point, a feedback is a numeric rating in response to a query. In a later section, we will illustrate that the feedback could also be a categorical answer. A query is defined as a question for soliciting an opinion on an item or topic of interest. For instance, a query could be “What is your opinion on the Canon 5D Mark III DSLR camera?”

The feedback acts as a trust empowering information aid to the user in making a choice. In the simplest case, the feedback is an average of the feedback inputs from all friends within one degree of separation, each weighted by the directional trust the user has on that friend. This is similar to the model presented in the FilmTrust work by Jennifer Golbeck [12]. The feedback is obtained per query. Because of the dynamic nature of queries as well as the trust levels specified during queries, no feedback can be pre-defined or stored on the cloud platform that hosts the social network.

In order to preserve privacy, TSF must ensure the non-

APPLIED COMPUTING REVIEW SEP. 2014, VOL. 14, NO. 3
The directed social connections of $u$ to any friend. (c) the aggregated feedback result to the social network and the friends and to the social network; and (b) the feedback disclosure of: (a) the directional trust values in a query to the friends and to the social network; and (b) the feedback from a particular friend of the user to the social network and the user. (c) the aggregated feedback result to the social network or to any friend.

3.1.1 Additively homomorphic cryptosystem – Paillier

Before delving further into the proposal, here we briefly introduce the notion of an additively homomorphic cryptosystem. The Paillier public-key cryptosystem [29] exhibits additively homomorphic properties, which we utilise in our proposals. Denoting encryption and decryption functions by $E()$ and $D()$ respectively, the encryption of the sum of two plaintext messages $m_1$ and $m_2$ is the modular product of their individual ciphertexts:

$$E(m_1 + m_2) = E(m_1) \cdot E(m_2) \tag{1}$$

while, the encryption of the product of one plaintext messages $m_1$ and a plaintext integer multiplicand $\pi$ is the modular exponentiation of the ciphertext of $m_1$ with $\pi$ as the exponent:

$$E(m_1 \cdot \pi) = E(m_1)\pi \tag{2}$$

With such an additively homomorphic cryptosystem at our disposal, let us denote the directional trust from user $a$ to friend $b$ as $T_{a\rightarrow b}$, the feedback from a friend $i$ on a query $k$ as $\omega_{i,k}$ and the total number of friends responding to the query as $n$. The trust value and the individual feedback value are discrete integers. The trusted feedback on query $k$ for user $u$ is given as:

$$\mathcal{F}_{u,k} = \frac{\sum_{i \neq j \neq u}^{n} \omega_{i,k} T_{u\rightarrow i}}{\sum_{i \neq j \neq u}^{n} T_{u\rightarrow i}} \tag{3}$$

This computation can be performed over the (additively homomorphic) encrypted domain for user $u$ as:

$$\mathcal{F}_{u,k} = \frac{D(\prod_{i \neq j \neq u} E(0,r_{i})E(T_{u\rightarrow i})^{\omega_{i,k}})}{D(\prod_{i \neq j \neq u} E(T_{u\rightarrow i}))} \tag{4}$$

The encryption of zero performed by the friend $i$, (denoted as $E(0,r_i)$) ensures that the encrypted partial feedback from friend $i$, i.e., $E(0,r_i)E(T_{u\rightarrow i})^{\omega_{i,k}}$ does not reveal $\omega_{i,k}$ despite the cloud’s knowledge of $E(T_{u\rightarrow i})$, unless the user $u$ and the cloud collaborate. The formal proof is in section 4.2. The trusted social feedback mechanism is illustrated in figure 1 and is described in algorithm 1. While sending the question, the user attaches an encrypted trust value for each friend to the question such that when a friend responds, the response is homomorphically multiplied by the trust value. The cloud aggregates those individual responses from the friends and sends back the aggregate response to the user after a threshold number of friends have responded. The flow of information in TSF is shown in figure 2.

As trust is personal and idiosyncratic [9], our proposed feedback mechanism is only there for trust empowerment, not to enforce a trust decision on the user. What the user does with the feedback is solely her choice. Therefore, a mathematical model for trust transitivity over multiple degrees of separation in the social network graph is often inadequate and meaningless because the model would tend to suggest a particular trust level. Trust is also sensitive to changes over time and context. In our proposal, the trust values can be as short-lived as a single query, which caters for temporal changes. The user can solicit the response to her query from a selected group (based on any particular context) of friends, thereby enabling context sensitivity. Thus, the queries in TSF are short-lived and context sensitive.

Untrust [25], which can be expressed in our proposed feedback mechanism, is also context sensitive. This means that Alice could trust her friend Bob for an opinion on cloud security but at the same time untrust him regarding any opinion on quantum entanglement. Untrust can prove use-

3The notations $E(x,r_u)$ and $E(x)$ are synonymous, i.e., encryption performed by the user $u$. The random number notation is used only when the operation is performed by some other user $i$ with $u$’s public key, i.e., $E(x,r_i)$.

4The $\cdot$ is used to denote multiplication for the sake of readability.
ful to accept negative feedback or reject positive feedback from untrusted friends for specific queries. In our current prototype, we do not model untrust.

**ALGORITHM 1:** Computing the trusted social feedback for user u on item k.

Require: Additively homomorphic encrypted domain for user u, i.e., \( E \) and corresponding public key.

Require: Encrypted directional trust \( E(T_u \rightarrow i) \) from user u to each friend i.

1: for each encrypted directional trust \( E(T_u \rightarrow i) \) do
2:    if i wishes to respond then
3:        i computes encrypted partial feedback,
4:            \( \psi_i \leftarrow E(0, r_i) \cdot E(T_u \rightarrow i)^{\omega_{i,k}} \)
5:        social network updates encrypted trusted feedback,
6:            \( \Psi \leftarrow \Psi \cdot \psi_i \)
7:    end if
8: end for
9: return encrypted trusted feedback, \( \Psi \).
10: return encrypted response cardinality, \( \eta \).
11: user u obtains the trusted social feedback,
12: \( \mathcal{F}_{u,k} = \frac{D(\Psi)}{D(\eta)} \).

### 3.2 Trusted crowdsourcing: opinions of domain experts

An extension of the trusted social feedback is a concept, which we call the trusted crowdsourcing (TCS). Through TCS, the user no longer solicits opinions from friends but uses a crowdsourcing platform to pose questions to domain experts. We assume the existence of a reputation system for domain experts through which experts gain or lose reputation based on how their answers are received. A mechanism to infer reputation based on the TCS model is left for future work.

The TCS ensures the non-disclosure of: (a) the directional trust values in a query to the eligible responders and to the crowdsourcing platform; (b) the feedback from a particular responder to the crowdsourcing; and (c) the aggregated feedback result to the crowdsourcing platform or to any responder.

The key difference between TCS and TSF is that in TCS, the user does not know at the time of the query who in the crowd will be eligible and willing to respond. Furthermore, the absence of a social network connection between the user and the potential responder means that it is impossible for the user to specify directional trust values at the time of query as it was done in TSF. However, before letting a domain expert answer a query, the crowdsourcing platform can obtain, from the user, the encrypted trust that the user chooses to assign to the responder. Thus, before receiving an answer, the user knows the identities of the responders — this is in sharp contrast with the TSF model. From the perspective of the user, the trust may be specified based on the reputation of the responder. In keeping with the understanding that trust is personal and idiosyncratic, the trust in TCS is personal; reputation is a trust aid, not a trust enforcement. Apart from the reputation, the response eligibility criteria is similar to that used in most crowdsourcing platforms. For instance, if the user asks the question “I am going to Japan next week for the first time. Which places would you recommend visiting?” then a likely response eligibility criteria is that the responder must have visited Japan and/or have lived/currently living in Japan. The eligibility criteria is user-specified and may not exist if the user so chooses.

Responders express their wish to the crowdsourcing platform to respond to a particular question. If the eligibility criteria is met, the crowdsourcing platform is responsible for obtaining and sending the encrypted trust values to each el-
User u decrypts the numerator and denominator and divides to obtain the plaintext feedback value. Individual responses are also decrypted separately by u.

Figure 3. Overview of the trusted crowdsourcing mechanism.

Figure 4. The trusted crowdsourcing sequence.

eligible responder. From that point forward, TCS works like TSF. An overview diagram of TCS is shown in figure 3 while the operating steps are described in algorithm 2, which is only slightly modified in comparison with algorithm 1. The flow of information in TCS is shown in figure 4. Notice that the crowdsourcing platform sends back to the user, both the aggregated response as well as the individual responses.

Although the mechanism to build reputations of responders is left for future work, a potential way to infer reputations could be to compare individual answers with the collective answer. This might help the user form a better understanding of where each individual answer stands. In addition, the availability of individual answers also helps with forming a better view of the distribution of answers than just relying on the mean.

3.3 Numeric answers to categorical answers

Although, in an example in the previous section, we mentioned a question “I am going to Japan next week for the first time. Which places would you recommend visiting?”
sent out to the crowd, the TSF or TCS models described so far cannot support the answer to such a question. The reason is that the both the TSF and the TCS models only support single-valued numeric answers to questions such as “What is your opinion on the Canon 5D Mark III DSLR camera?”.

However, this can be easily extended to support a multi-valued categorical answer. Let us re-consider the question that requires a multi-valued answer: “I am going to Japan next week for the first time. Which places would you recommend visiting?”. Let us, now, assume that the potential places are to be chosen from a list of classes: Tokyo, Kyoto, Hakone and Sapporo. For simplicity, we will assume that each responder will have to include all these classes in the response. A system that allows for ranking only some classes by each user is to be investigated in the future, and may bear some similarity to our work on rank aggregation of partially ranked lists [2].

With each response including all classes, a responder \( p_1 \) may respond with the following class values
\[
\begin{align*}
\text{Tokyo} & = 0.4, \\
\text{Kyoto} & = 0.3, \\
\text{Hakone} & = 0.2, \\
\text{Sapporo} & = 0.1,
\end{align*}
\]
thus specifying a probability distribution with the class values summing up to 1. Someone else \( p_2 \) can specify:
\[
\begin{align*}
\text{Tokyo} & = 0.2, \\
\text{Kyoto} & = 0.3, \\
\text{Hakone} & = 0.3, \\
\text{Sapporo} & = 0.2.
\end{align*}
\]
We can apply the TSF or the TCS algorithm on each class independently of the others. Thus, the end result computed by the user, \( F_{u,k} \), will be per class, which can be normalised further to obtain the correct probability distribution. Table 1 is an example, in the plaintext domain, of a multi-valued answer. Even if the final weighted averages (Weighted mean in table 1) do not add up to unity, we do not need to normalise in order to find a ranking from the answer. In the aforementioned example, Tokyo is the most preferred place to visit followed, in order, by Kyoto, Hakone and Sapporo. Note that the implementation will have to scale the class values in the probability distribution to integer-only domain so that those values can be used by a homomorphic cryptosystem that works only with integers.

4. SECURITY ANALYSIS

In this section, we present the description of various attacks by honest-but-curious adversaries and discuss how our model copes. Furthermore, we also provide a proof of obfuscation by the encryption of zero. We also present a specialised partial response disclosure attack.

4.1 Honest but curious adversary model

In this discussion, the word cloud will be considered synonymous with either the social network or the crowdsourcing platform in terms of threats because both will usually utilise a cloud environment. Thus, the internal privacy threats to either can arise from the cloud infrastructure. We assume that the parties involved in this process are honest but curious. Therefore, attacks involving collaborations between the cloud and the attacker are not considered as realistic threats although we have described some such possible attacks. For a malicious user, a specialised attack for partial response disclosure is also described in section 4.3.

4.1.1 Curious user, multi-query and sybil attacks

The user can run multiple queries requesting the feedback on the same question from the same set of friends or domain experts. In doing so, and by varying the user’s directional trust on each responder, the user can acquire the information necessary to reveal the feedback provided by each responder. However, the feedback response is slow and some responders may choose not to respond. Furthermore, the feedback from the same person may vary over time. In addition, with crowdsourcing, each query may cost money making this an expensive attack. Therefore, using a multi-query attack is not guaranteed to succeed. To further enhance the privacy of the feedback, the responder can perturb his/her feedback input in bounded integral ranges – an avenue we have left open for future work.

However, in a sybil attack the user asks a question to one real person and a number of sybil identities. Upon receiving the responses, the asker can find out the exact response from the real person given the knowledge of those from the sybil identities. Our model is not resistant against this type of sybil attacks.

4.1.2 Curious cloud, man-in-the-middle attack

Despite the query itself being sent in clear text, the directional trust values from the user and the partial feedback from each responder are both in the encrypted domain of the user. Even though the cloud knows the encrypted directional trust value, it cannot decipher the actual feedback from any responder since encrypted zero, i.e., \( \mathcal{E}(0, r_i) \), is homomorphically added by each responder thus making the encrypted trusted feedback component probabilistic. The cloud, however, can tell who responded to the query.

4.1.3 Curious responder

Any particular responder cannot determine the directional trust value because it is encrypted by the user’s public key.
4.1.4 Collaborative attacks

If the user and the cloud collaborate then all the partial feedbacks can be deciphered since the cloud will be able to decrypt partial feedback values with the help of the user. If a responder and the cloud collaborate, the responder can learn how many other people responded to the query but it cannot decipher the actual individual feedback values. If the user and a responder collaborate, they can only learn about each others’ secrets – the directional trust value and the feedback.

4.1.5 Out-of-the-range attacks

Both the responder and the cloud can encrypt arbitrary numbers and send them to the user in the response. Homomorphic range check protocols [30] may be applicable to protect those scenarios but this falls within the remits of future work.

4.2 Proof of obfuscation by encryption of zero

Since the numeric feedback on item k from a responder, i, is in a fixed discrete integral range, the cloud can attempt to learn it by pre-computing all possible values5 of $E(T_{u \rightarrow i})^{\omega_i,k}$ using a trial-and-error method of dividing what the responder sends by the pre-computed value to eliminate the obfuscating encryption of zero. Let us assume that the correct value of $\omega_{i,k}$ in question is $\omega_1$ and a wrong value is $\omega_2$. This is what happens.

4.2.1 Case A: correct pre-computed value

If the cloud used the correct pre-computed value: $E(T_{u \rightarrow i})^{\omega_1}$, we have:

$$\frac{E(0, r_i)E(T_{u \rightarrow i})^{\omega_{i,k}}}{E(T_{u \rightarrow i})^{\omega_1}} = E(0, r_i)E(T_{u \rightarrow i})^{\omega_{i,k} - \omega_1}$$

$$= E(0, r_i)$$

Now, the cloud computes:

$$\frac{E(0, r_i)E(T_{u \rightarrow i})^{\omega_{i,k}}}{E(0, r_i)} = E(T_{u \rightarrow i})^{\omega_{i,k}} = E(T_{u \rightarrow i})^{\omega_i,k}$$

Thus, the cloud obtains the same value as the one it pre-computed.

4.2.2 Case B: wrong pre-computed value

If the cloud used a wrong pre-computed value: $E(T_{u \rightarrow i})^{\omega_2}$, we have:

$$\frac{E(0, r_i)E(T_{u \rightarrow i})^{\omega_{i,k}}}{E(T_{u \rightarrow i})^{\omega_2}} = E(0, r_i)E(T_{u \rightarrow i})^{\omega_{i,k} - \omega_2}$$

Now, the cloud computes:

$$\frac{E(0, r_i)E(T_{u \rightarrow i})^{\omega_{i,k}}}{E(0, r_i)} = E(T_{u \rightarrow i})^{\omega_{i,k} - \omega_i,k + \omega_2} = E(T_{u \rightarrow i})^{\omega_2}$$

Here again, the cloud obtains the same value as the one it pre-computed.

Since the results from both the right and the wrong guesses are indistinguishable, the cloud cannot guess which one is the true value of $E(T_{u \rightarrow i})^{\omega_{i,k}}$ and hence $\omega_{i,k}$.

4.3 A specialised partial response disclosure attack

Our construction is not inherently secure against a malicious user wishing to know the responses of her responders from the aggregate encrypted values. This attack consists of creating a vector with several coordinates inside a single encrypted value. These coordinates can be read independently by the malicious user. Consider an x-bit number and treat it as a vector of dimension y, where each coordinate is represented using $\frac{x}{y}$ bits. If operations are performed on this vector with no individual coordinate exceeding $2^\frac{x}{y} - 1$, then there is no loss of information for that coordinate. The following example illustrates this idea.

1. Assume $x = 16$ and $y = 4$, then each coordinate can represent values in the range [0 15].

2. The user asks four responders, i.e., $f_1 \ldots f_4$ a question using the following trust values (spaces introduced for readability), represented as bit sequences:

$$T_{u \rightarrow f_1} = 0000 0000 0000 0001 \text{ [decimal : 1]}$$
$$T_{u \rightarrow f_2} = 0000 0000 0001 0000 \text{ [decimal : 32]}$$
$$T_{u \rightarrow f_3} = 0000 0001 0000 0000 \text{ [decimal : 512]}$$
$$T_{u \rightarrow f_4} = 0001 0000 0000 0000 \text{ [decimal : 8192]}$$

3. Each responder provides his/her response in the range [1 15] weighted by the ingress trust value, i.e.,

$$E(T_{u \rightarrow i})^{\omega_{i,k}}$$

The encryption of zero is left out for simplicity because it does not stop this attack, which happens in the plaintext domain.

4. The cloud aggregates the resultant numerator as:

$$\prod_{i \neq u} E(T_{u \rightarrow i})^{\omega_{i,k}}$$

Table 1. An example of multi-valued answer.

<table>
<thead>
<tr>
<th>Person 1 ($p_1$)</th>
<th>Tokyo</th>
<th>Kyoto</th>
<th>Hakone</th>
<th>Sapporo</th>
<th>Trust</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>0.3</td>
<td>0.2</td>
<td>0.1</td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td>Person 2 ($p_2$)</td>
<td>0.2</td>
<td>0.3</td>
<td>0.3</td>
<td>0.2</td>
<td>0.5</td>
</tr>
<tr>
<td>Person 3 ($p_3$)</td>
<td>0.1</td>
<td>0.4</td>
<td>0.3</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>Person 4 ($p_4$)</td>
<td>0.5</td>
<td>0.1</td>
<td>0.2</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>Weighted mean</td>
<td>0.34615</td>
<td>0.25385</td>
<td>0.25</td>
<td>0.16538</td>
<td>–</td>
</tr>
<tr>
<td>Class rank</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>–</td>
</tr>
</tbody>
</table>
Since none of the coordinates in the numerator has a value greater than 15, the malicious user can extract the answer from each responder by reading the decrypted numerator, 4-bits at a time. The technique also works for trust values where the particular non-zero nibble is greater than 0001, for example 0010 0000 0000 0000 [decimal : 16384]. In that case, the malicious user simply needs to adapt the coordinate length accordingly and once extracted, divide it by the original trust value assigned to that particular responder.

To prevent this attack, a proof stating that the trust values are in a given range is necessary. Alternatively, if the number of responders asked is large enough in comparison with the bit space of the plaintext trust values then the bit manipulations will overlap, thus making it impossible for the attacker to identify individual ratings.

5. IMPLEMENTATION AND EVALUATION

![Graph](image)

(a) Pre: 1=“extremely bad”; 7=“extremely good”

![Graph](image)

(b) Post: 1=“extremely bad”; 7=“extremely good”

Figure 5. Responder’s change in the understanding of the TSF prototype (Q1), pre- and post- experiment.

In this section we present the results from: 1. the user studies for the trusted social feedback prototype; 2. the performance evaluation of the speed of cryptographic primitives on the web front-end; 3. the speed of the essential functions of the prototype at the back-end of the trusted social feedback prototype; and 4. the user studies for the trusted crowdsourcing model.

The experimental trusted social feedback prototype runs on the Google App Engine for Java. The application uses Facebook to perform user login and to determine social connections.

![Table](image)

Table 2. Change in uncertainties associated with 4 questions and 12 users.

(a) Change in response uncertainties for each question per user.

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>Reduced</td>
<td>No change</td>
<td>Reduced</td>
</tr>
<tr>
<td>U2</td>
<td>Reduced</td>
<td>Reduced</td>
<td>Reduced</td>
</tr>
<tr>
<td>U3</td>
<td>No change</td>
<td>Reduced</td>
<td>Reduced</td>
</tr>
<tr>
<td>U4</td>
<td>Reduced</td>
<td>Reduced</td>
<td>Increased</td>
</tr>
<tr>
<td>U5</td>
<td>Reduced</td>
<td>Reduced</td>
<td>Reduced</td>
</tr>
<tr>
<td>U6</td>
<td>No change</td>
<td>Reduced</td>
<td>No change</td>
</tr>
<tr>
<td>U7</td>
<td>Reduced</td>
<td>No change</td>
<td>Reduced</td>
</tr>
<tr>
<td>U8</td>
<td>No change</td>
<td>Reduced</td>
<td>Reduced</td>
</tr>
<tr>
<td>U9</td>
<td>Reduced</td>
<td>Reduced</td>
<td>Reduced</td>
</tr>
<tr>
<td>U10</td>
<td>No change</td>
<td>Reduced</td>
<td>Reduced</td>
</tr>
<tr>
<td>U11</td>
<td>Reduced</td>
<td>Reduced</td>
<td>Reduced</td>
</tr>
<tr>
<td>U12</td>
<td>Reduced</td>
<td>Reduced</td>
<td>No change</td>
</tr>
</tbody>
</table>

(b) Change in response uncertainties per question.

<table>
<thead>
<tr>
<th>Question</th>
<th>Increased</th>
<th>No change</th>
<th>Reduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>0 (0%)</td>
<td>4 (33%)</td>
<td>8 (67%)</td>
</tr>
<tr>
<td>Q2</td>
<td>0 (0%)</td>
<td>2 (17%)</td>
<td>10 (83%)</td>
</tr>
<tr>
<td>Q3</td>
<td>1 (8%)</td>
<td>2 (17%)</td>
<td>9 (75%)</td>
</tr>
<tr>
<td>Q4</td>
<td>1 (8%)</td>
<td>4 (33%)</td>
<td>7 (58%)</td>
</tr>
</tbody>
</table>

Table 3. Performances of Paillier in Javascript. Times are in milliseconds. KG: key generation, E: encryption, HA: homomorphic add; HM: homomorphic multiplication; D: decryption. The number suffixed to these abbreviations indicate cryptosystem bit size.

<table>
<thead>
<tr>
<th>Bit Size</th>
<th>Chrome</th>
<th>Firefox</th>
<th>IE</th>
<th>Safari</th>
</tr>
</thead>
<tbody>
<tr>
<td>KG-512</td>
<td>69</td>
<td>57</td>
<td>441</td>
<td>1032</td>
</tr>
<tr>
<td>E-512</td>
<td>23</td>
<td>16</td>
<td>195</td>
<td>367</td>
</tr>
<tr>
<td>HA-512</td>
<td>2</td>
<td>2</td>
<td>9</td>
<td>27</td>
</tr>
<tr>
<td>HM-512</td>
<td>11</td>
<td>8</td>
<td>104</td>
<td>214</td>
</tr>
<tr>
<td>D-512</td>
<td>23</td>
<td>15</td>
<td>195</td>
<td>371</td>
</tr>
</tbody>
</table>

Table 4. The average times taken for various servlet function calls. The profile servlet is responsible for user profile specific functions while qaserv deals with questions and their responses.

<table>
<thead>
<tr>
<th>Servlet:Action</th>
<th>Call count</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile:getProfile</td>
<td>121</td>
<td>3128</td>
</tr>
<tr>
<td>profile:getTopUsers</td>
<td>207</td>
<td>1816</td>
</tr>
<tr>
<td>profile:savePublicKey</td>
<td>36</td>
<td>2012</td>
</tr>
<tr>
<td>qaserv:answerQuestion</td>
<td>195</td>
<td>233</td>
</tr>
<tr>
<td>qaserv:askQuestion</td>
<td>81</td>
<td>1826</td>
</tr>
<tr>
<td>qaserv:myNotifications</td>
<td>552</td>
<td>783</td>
</tr>
<tr>
<td>qaserv:myQuestions</td>
<td>262</td>
<td>1779</td>
</tr>
</tbody>
</table>
5.1 TSF: measuring perception of privacy and foreground trust

The following questions were used to record user responses and measure changes in uncertainty in these responses. Each question was presented once before and once after each user had a chance to play with the experimental prototype.

Q1 How is your understanding about what you can do with this application?

Q2 How well do (did) you feel that the application will preserve (preserved) the privacy of the personal trust levels that you have on your friends, and the privacy of the responses from your friends?

Q3 How useful do you think is this application?

Q4 How likely are you to use such an application, if available publicly?

Each question was followed by a question to measure uncertainty: How certain are you about your previous response?. Responses to each question was recorded on a 7-point Likert scale [24].

Figures 5 through 8 show the user responses before and after the experiment in terms of understanding, privacy-preservation, usefulness and likelihood of use of the prototype. Together, the figures show the change in the user perceptions before and after the experiment. In figure 5, the results suggest that use of the prototype may have helped the users to understand the application better. Similarly, figure 6 shows that the users' perception of how well privacy is preserved increased after using the prototype. The users also perceived the application to be more useful after having used the prototype, which is illustrated by figure 7. According to figure 8, the likelihood of users using such a system in the future also slightly improved overall after having used the prototype. Thus, this suggests that our prototype was perceived by the users to be an effective tool for preserving privacy while obtaining feedback.

Closely related to the sense of privacy, this user study also inferred users’ trust in the application, which relates to the concept of foreground trust [8]. It is different from the trust between friends that we have discussed so far. Dwyer et al. in [9], suggested that a reduction of uncertainty is positively correlated with the increase of trust. Thus, a measure of uncertainty is used to infer trust. In our user study with 12 participating users, we have employed pre-use and post-use questionnaires to determine the changes in uncertainty. The users are highly technically competent and were aware of this research work before using the prototype. Table 2 shows that the uncertainty in the users’ responses usually declined, thus suggesting a likely increase in foreground trust.

5.2 TSF: measuring performance

The speed at which a feedback can be obtained depends almost entirely on the speed at which friends respond to the question; and to some extent on the speed of cryptographic operations and that too on the client-side because the speed of the limited cryptographic operations on the cloud-side is usually negligible compared to delays caused by network la-
tencies, cloud instance initialisations, and datastore access. For every partial response submitted by a friend, the cloud is responsible for exactly two homomorphic additions, see line 4 in algorithm 1. We present a comparison of performances of cryptographic primitives on the client side. We have built a Google Web Toolkit wrapper for an optimised Javascript implementation of the Paillier cryptosystem using the Stanford Javascript BigInteger library. The result of each test, in table 3, is a rounded-off average from 50 runs. The tests were carried out on Windows 8, running on a 64-bit 3.4GHz Intel i7-3770 dual quad-core processor with 16GB RAM. The versions of the browsers are: Chrome 28.0.150072m, Firefox 22.0, Internet Explorer (IE) 10.0.9200.16599 and Safari 5.1.7. IE and Safari failed to finish the tests when the cryptosystem was set to 1024 bits, so we used a 512 bits cryptosystem for our tests.

Using the F1 (600MHz) instance class and the high-replication datastore of the Google App Engine, the averages of the times taken for the different servlet calls are shown in table 4. The time taken for a particular function call also includes the time taken to execute any intermediate servlet filters, for instance the filter that verifies the logged-in users.

5.3 TCS: measuring user acceptance

Further to measuring foreground trust and sense of privacy for the trusted social feedback implementation, we also measured user acceptance of the trusted crowdsourcing model.

The public questionnaire was preceded by a brief description of the TCS model. The users were asked the following questions after reading the description of the model.

Q1 If you were Alice, how well do you feel that the privacy of your personal trust value in Bob is preserved by CrowdTrust?
Q2 If you were Alice, how important would this privacy be to you?
Q3 If you were Bob (the responder) how well do you feel that the privacy of your answer to Alice is hidden from the crowdsourcing platform by CrowdTrust?
Q4 If you were Bob, how important would this privacy be to you?
Q5 How likely would you be to use such a service to ask for opinions from the crowd?
Q6 How likely would you be to use such a service to answer questions, get remunerated and develop reputation?
Q7 Would you prefer CrowdTrust to be implemented as an end-to-end user communication where the crowdsourcing platform is oblivious to the privacy preserving layer? Or, do you prefer an existing crowdsourcing platform to integrate and provide CrowdTrust as a separate add-on service?

Responses to each question was recorded on a 7-point Likert scale. There were two other questions that were used to collect comments and concerns from the responders to the survey.

The public questionnaire6 was preceded by a brief description of the TCS model. The users were asked the following questions after reading the description of the model.

6. CONCLUSIONS

In this paper, we have presented a working prototype for obtaining feedback on queries from one’s trusted friends in a privacy preserving manner. We have implemented and tested the prototype on the Google App Engine with Facebook, and have run a user study to evaluate the perception of privacy as well as foreground trust in the prototype. The evaluation shows that the prototype was considered by the users to be an effective tool for preserving privacy while obtaining feedback; and that it can be inferred that the foreground trust in the prototype was high. A novel contribution of the paper is the observation that the technique for privacy-preserving social feedback can also be used in the domain of crowdsourcing. Based on this, we develop a framework for trusted crowdsourcing to enable users to obtain feedback from domain experts who may not necessarily be friends. We have evaluated users’ perception of privacy preservation and the acceptance of our trusted crowdsourcing model through a user survey. The results of the survey demonstrate that the users, in general, felt that the model was good at preserving the privacy.

In the future, we plan to expand this work by developing:

**Figure 10. The perception and importance of privacy from Bob’s viewpoint.**

**Q8** If you were Alice, would you have any concerns or suggestions?

**Q9** If you were Bob, would you have any concerns or suggestions?

The responses to Q8 and Q9 were used to refine the TCS model, while the analysis of the results of responses to Q1 through Q7 are presented below. At the time of this writing, the survey had 16 responses. Figure 9 shows to what extent Alice (i.e., the person who uses the crowdsourcing platform to send her query) was convinced that her privacy was preserved in the model, and how important that privacy is to her. Similarly, figure 10 correspondingly shows to what extent Bob (i.e., the responder’s) was convinced that her privacy was preserved in the model, and how important that privacy is to him. Figure 11 illustrates to what extent a user would use such a system if they were playing the role of Alice/Bob, and also shows their preferences regarding the form of implementation (end-to-end/integrated) of the TCS framework.

**Figure 11. Alice’s and Bob’s preferences on the implementation of privacy-preserving trusted crowdsourcing and the likelihood of use.**
7. REFERENCES


ABOUT THE AUTHORS:

Dr. Anirban Basu is a Senior Researcher at KDDI R&D Laboratories in Japan. Prior to that, he worked as a Post-doctoral Researcher at Tokai University. He is also a Visiting Research Fellow at the University of Sussex. He holds a Ph.D. in Computer Science and a Bachelor of Engineering (Hons.) in Computer Systems Engineering from the University of Sussex. His research interests are in computational trust, privacy and security and peer-to-peer networks. He is particularly active within the IFIPTM computational trust management community.

Jaideep Vaidya is an Associate Professor in the MSIS Department at Rutgers University. He received the B.E. degree in Computer Engineering from the University of Mumbai, the M.S. and Ph.D. degree in Computer Science from Purdue University. His general area of research is in data mining, data management, security, and privacy. He has published over 100 technical papers in peer-reviewed journals and conference proceedings, and has received several best paper awards from the premier conferences in data mining, databases, digital government, and informatics.

Juan Camilo Corena is a PhD candidate at Keio University in Japan. In the past, he has been an information security researcher at KDDI R&D Laboratories in Japan and an instructor and researcher at Politécnico Grancolombiano in Colombia. He holds a master and a bachelor degree in systems and computer engineering from University of the Andes (Colombia). His research interests include cryptography, erasure codes, algorithm design and network security.

Shinsaku Kiyomoto received his B.E. in engineering sciences and his M.E. in Materials Science from Tsukuba University, Japan, in 1998 and 2000, respectively. He joined KDD (now KDDI) and has been engaged in research on stream ciphers, cryptographic protocols, and mobile security. He is currently a senior researcher at the Information Security Laboratory of KDDI R&D Laboratories Inc. He was a visiting researcher of the Information Security Group, Royal Holloway University of London from 2008 to 2009. He received his doctorate in engineering from Kyushu University in 2006. He received the IEICE Young Engineer Award in 2004. He is a member of JPS and IEICE.
Steve Marsh (www.stephenmarsh.ca) is a trust scientist who works with the phenomenon of trust for computational systems. He is an Assistant Professor of Information Systems in the Faculty of Business and Information Technology, University of Ontario Institute of Technology. His PhD (University of Stirling, 1994) is a seminal work that introduced the first formalisation of the phenomenon of trust (the concept of 'Computational Trust'), and applied it to Multi Agent Systems. As a milestone in trust research, it brought together disparate disciplines and attempted to make sense of a vital phenomenon in human and artificial societies, and is still widely referenced today. Steve's current research builds extensively on this model, applying it to context-adaptive mobile devices, device comfort, protection of the user, and mobile device security and privacy.

Mr. Guibing Guo is a Ph.D student from Nanyang Technological University, Singapore. He received his bachelor degree in Computer Science and Technology and master degree in Computer Software and Theory in 2008 and 2011 respectively at Yanshan University, China. He also received another master degree in Computer Control and Automation in 2010 at Nanyang Technological University, Singapore. His current research is focused on building an effective recommender system to resolve the data sparsity and cold start problems for traditional recommender systems.

Jie Zhang is an Assistant Professor of the School of Computer Engineering, Nanyang Technological University, Singapore. He is also an Academic Fellow of the Institute of Asian Consumer Insight and Associate of the Singapore Institute of Manufacturing Technology (SIMTech). He obtained Ph.D. in Cheriton School of Computer Science from University of Waterloo, Canada, in 2009. During PhD study, he held the prestigious NSERC Alexander Graham Bell Canada Graduate Scholarship rewarded for top PhD students across Canada. He was also the recipient of the Alumni Gold Medal at the 2009 Convocation Ceremony. His research interests are in trust management and multi-agent systems.

Yutaka Miyake received the B.E. and M.E. degrees of Electrical Engineering from Keio University, Japan, in 1988 and 1990, respectively. He joined KDD (now KDDI) in 1990, and has been engaged in the research on high-speed communication protocol and secure communication system. He received the Dr. degree in engineering from the University of Electro-Communications, Japan, in 2009. He is currently a senior manager of Information Security Laboratory in KDDI R&D Laboratories Inc. He received IPSJ Convention Award in 1995 and the Meritorious Award on Radio of ARIB in 2003.
Hardware-Software Collaboration for Secure Coexistence with Kernel Extensions

Daniela Oliveira  
University of Florida  
Dept. of Electrical and Computer Engineering  
daniela@ece.ufl.edu

Nicholas Wetzel  
Bowdoin College  
Dept. of Computer Science  
nwetzel@bowdoin.edu

Max Bucci  
Bowdoin College  
Dept. of Computer Science  
mbucci@bowdoin.edu

Jesus Navarro  
NVIDIA  
jnavarro@nvidia.com

Dean Sullivan  
University of Central Florida  
Dept. of Electrical Engineering and Computer Science  
dean.sullivan@ucf.edu

Yier Jin  
University of Central Florida  
Dept. of Electrical Engineering and Computer Science  
yier.jin@eecs.ucf.edu

ABSTRACT

Our society is dependent upon computer systems that are the target of a never-ending siege against their resources. One powerful avenue for exploitation is the operating system kernel, which has complete control of a computer system’s resources. The current methodology for kernel design, which involves loadable extensions from third parties, facilitates compromises. Most of these extensions are benign, but in general they pose a threat to system trustworthiness: they run as part of the kernel and some of them can be vulnerable or malicious. This situation is paradoxical from a security point of view: modern OSes depend, and must co-exist, with untrustworthy but needed extensions. Similarly, the immune system is continuously at war against various types of invaders and, through evolution, has developed highly successful defense mechanisms. Collaboration is one of these mechanisms, where many players throughout the body effectively communicate to share attack intelligence. Another mechanism is foreign body co-existence with its microbiota. Remarkably, these properties are not leveraged in kernel defense approaches. Security approaches at the OS and virtual machine layers do not cooperate with each other or with the hardware. This paper advocates a new paradigm for OS defense based on close collaboration between an OS and the hardware infrastructure, and describes a hardware-software architecture realizing this vision. It also discusses the architecture design at the OS and hardware levels, including experimental results from an emulator-based prototype, and aspects of an ongoing hardware implementation. The emulator-based proof-of-concept prototype, Ianus, uses Linux as the OS and the Bochs x86 emulator as the architecture layer. It successfully minimized kernel extensions interactions with the original kernel. Its security was evaluated with real rootkits and benign extensions. Ianus' performance was analyzed with system and CPU benchmarks and it caused a small overhead to the system (approximately 12%).

Categories and Subject Descriptors

D.4.6 [Operating Systems]: Operating system security and protection

General Terms

Security

Keywords

HW-SW collaboration, OS defense, kernel extensions, immune system

1. INTRODUCTION

Our society has become dependent on increasingly complex networked computer systems which are the target of a never-ending siege against their resources, infrastructure and operability for economic and political reasons. Attacks on computer systems can have devastating consequences to our society, such as a nuclear power facility going rogue, an electrical grid shutting down an entire city, or the financial sector going down after a hit in a bank [1, 2]. One particularly powerful avenue for system exploitation is the compromise of the operating system kernel, which has complete control of a computer system’s resources and mediates access of user applications to the hardware. The current methodology for kernel design, which involves loadable extensions from third parties, facilitates system compromises. In this methodology the kernel has an original set of components that can be extended during boot time or while the system is running. This paradigm is adopted by modern OSes and represents a convenient approach for extending the kernel functionality.

Kernel extensions, especially device drivers, currently make up a large fraction of modern kernel code bases (approximately 70% in Linux and a larger percentage in Windows) [3]. Most of these extensions are benign and allow the system to communicate with an increasing number of diverse I/O devices without the need of OS reboot or recompilation. However, they pose a threat to system trustworthiness because they run with the highest level of privileges and can be vulnerable or malicious. This situation is paradoxical from a security point of view: modern OSes depend and must co-exist with untrustworthy but needed extensions.

Similar to the current scenario of computer security, the mammalian immune system faces the exact same challenges every day, continuously battling against various types of invaders. It is hitherto the most successful defense system and has been perfected by Nature through millions of years of evolution. Leveraging immune system defense mechanisms built-up over eons is the key to im-
prove computer systems security, particularly OS security. Among its various defense mechanisms, two are most relevant. The first is cooperation. Team work is vital to a properly functioning immune system, where many types of white blood cells collaborate across the entirety of the human body to defend it against invaders. The second is foreign body co-existence. The human body has ten times more microbes than human cells. Most of these microbes are benign and carry out critical functions for our physiology, such as aiding digestion and preventing allergic reactions. In spite of that, a fraction of these microbes pose a threat to our bodies as they can cause pathologies. The immune system has evolved so that it can maintain an homeostatic relationship with its microbiota, and this involves controlling microbial interactions with host tissues, lessening the potential for pathological outcomes [4].

Remarkably, these two highly successful immunity mechanisms have not been applied to OS security. The two key players that make up a computer system, OS and hardware, interact precariously with each other and do not cooperate. Security approaches employed at the OS and virtual machine (VM) layers do not cooperate with the hardware, nor do they communicate information to dynamically adapt to future incursions. Current virtualization-based security solutions do not rely on collaboration with the guest OS because they are based on the traditional paradigm for OS protection, which advocates placing security mechanisms in a VM layer, thereby leaving the OS with no active role in its own defense. This is because the current threat model only defines the VM and the hardware as trustworthy so that the guest OS is considered untrustworthy and easily compromised by malware [5, 6, 7, 8, 9, 10]. This traditional model suffers from two main weaknesses.

The first weakness is the semantic gap: there is significant difference between the high level abstractions observed by the guest OS and the low level abstractions at the VM. The semantic gap hinders the development and widespread deployment of virtualization-based security solutions because these approaches need to inspect and manipulate abstractions at the OS and architecture level to function correctly. To address the semantic gap challenge, traditional VM-based security solutions use a technique called introspection to extract meaningful information from the system they monitor [5]. With introspection, the physical memory of the guest OS is mapped at the VM address space for inspection. High level information is obtained by using detailed knowledge of the OS layout, algorithms and data structures [11].

The second weakness of traditional VM-based OS defense mechanisms is the introspection mechanism itself, which is a manual, error prone, and time consuming task that, despite being perceived as secure until recently, does rely on the integrity of the guest OS to function correctly. Traditional introspection solutions assume that even if the guest OS is compromised, their mechanisms and tools, residing at a lower-level (VM) will continue to report accurate results. However, Baram et al [12] argued that this security assumption does not hold because an adversary, after gaining control of an OS (e.g., through kernel-level rootkits), can tamper with kernel data structures so as a bogus view of the system is provided for introspection tools.

This paper advocates a new paradigm for OS defense: OSes should evolve to closely interact with the hardware playing an active role in maintaining safe interactions with their extensions. Similar to the immune system, computer systems should control the interactions between extensions and the original kernel lessening the potential for security breaches. In this paper, a hardware-software (HW-SW) collaborative architecture for OS defense is proposed. The main idea is that the OS will provide the hardware with intelligence needed for enforcement of security policies that allow for safe co-existence of the kernel and its extensions. Specifically, the following security policies are considered:

- Kernel extensions should never directly write into kernel code and data segments, including limited portions of the stack segment, except into their own address spaces.
- Kernel extensions should only interact with the kernel and other extensions through exported functions.²

Enforcing these policies requires architectural support so that extensions’ execution are monitored and stopped in case of a violation. Controlling the execution of kernel extensions lies at the heart of the proposed approach. HW-SW collaboration is necessary because enforcing these policies requires system-level intelligence that only the OS can provide and architectural capabilities that only the hardware can support. For example, only the OS knows the boundaries of extensions in main memory and addresses of kernel functions. However, only the hardware can interpose on low level abstractions such as writes into main memory, invocation of CALL instructions and origin of the instructions for enforcing the policies. The cooperation benefits are clear when one considers that the OS and the hardware have access to a distinct set of functionalities and information, which in combination allows for enforcement of security policies that control the execution of kernel extensions.

This paper discusses the challenges of designing a hardware architecture that allows for cooperation and communication with the OS. An emulator-based proof-of-concept prototype called Ianus ³ was developed to validate the hardware implementation and the paradigm. It uses the Bochs Intel x86 emulator [13] as the architecture layer and Linux Ubuntu 10.04 (kernel version 2.6.32) as guest OS. Ianus’ experimental evaluation showed it successfully confined extensions into their own address spaces and contained their interactions with other parts of kernel code and data. Ianus’ security was assessed with a set of real kernel rootkits which were all stopped before any malicious actions were performed and with benign extensions that could run normally. The overhead to the system was analyzed with a set of system and CPU benchmarks and was found to be low, approximately 12%.

This paper is organized as follows. Section 2 discusses the challenges of HW-SW collaboration for security and the main requirements for such architecture. Section 3 describes in details the design and implementation of an emulator-based proof-of-concept prototype validating this vision. In section 4 the paper shows the experimental analysis of the prototype in terms of security and performance. Section 5 brings a discussion of other aspects of the hardware implementation and future work. Section 6 summarizes relevant related work in kernel protection and hardware support for system security. Section 7 concludes the paper.

2. CHALLENGES FOR HARDWARESOFTWARE COOPERATION

The main challenge to HW-SW cooperation is that current security approaches are single-layered and separate the field into distinct realms, either hardware or software. These approaches are

²Exported functions are those required by extensions to perform their tasks and can be viewed as the kernel interface to extensions. Kernel extensions, when loaded into the system can also export a subset of their functions to other extensions.

³Ianus is an ancient Roman God who has two faces, each looking in the opposite direction. His image is associated with the ability of imagining two opposites or contradictory ideas existing simultaneously.

⁴The term “guest” is used here because the operating system is running on top of an emulator, which runs on a host OS.
isolated and work independently. Security solutions at the OS and VM layers do not cooperate with the hardware, nor do they communicate information about attacks. Hardware security approaches, on the other hand, mainly focus on Trojan prevention and detection and rarely leverage system level context. Further, OS defense approaches generally make the flawed assumption that the underlying hardware is trustworthy, while hardware malware has been found in many embedded systems from chips used by the military, to off-the-shelf consumer electronics [14, 15, 16].

The main step for allowing HW-SW cooperation is implementing the communication interface between OS and hardware following a pre-defined communication protocol. The goal is to provide a physical mechanism for hardware and software to communicate, apply security policies, exchange intelligence, and physically monitor system operations. This hardware component, named hardware anchor, is designed and implemented in the processor with the goals of minimizing the performance overhead and better utilizing existing hardware resources.

The hardware anchor is made of two main components: a cross-boundary interface and a software operation checker. The cross-boundary interface enables communication and information exchange between the OS kernel and the processor. The anchor receives and interprets security instructions from the OS kernel, collects system-level intelligence and enforces the security policies. The security instruction is a new instruction added to the processor and behaves like a software interrupt with parameters passed in general purpose registers. The intelligence information includes boundaries of extensions, kernel function addresses and types (exported or non-exported), user-defined sensitive information, and protected memory spaces. Since the security instruction fetching and decoding functionalities share the on-board processor resources with normal instructions, the cross-boundary interface will be seamlessly embedded with the processor. The information collected through the interface will be stored inside secure hash tables within the processor, and cannot be accessed by any other hardware module.

The second anchor component, the system operation checker, is also located in the processor and monitors OS operations based on information collected through the cross-boundary interface. For example, the OS will downcall the anchor to provide memory boundaries of extensions whenever they are installed in the system. The hardware anchor will then record this information and block all of the extension’s operations at the architectural level that violate the security policies.

The system operation checker performs security validations to make sure OS extensions operate within the restricted boundaries defined by the security policies. The checker operates in a preventive mode so that any operations issued by kernel extensions will be checked for trustworthiness before they are performed. Several types of security checks can be performed. For example, the checker can monitor reads and writes in kernel space and calls to kernel functions.

OS kernel security instructions are the software counterpart of the hardware anchor. The OS kernel is modified to include calls to the hardware at specific points in its execution. For example, immediately after boot, the OS will perform calls to the hardware to pass kernel function addresses. The kernel also calls the hardware every time a new extension is installed/uninstalled to pass/remove its boundaries in main memory. It also calls the hardware to pass addresses of functions added by the extension. Whenever an extension allocates memory, the kernel calls the hardware to update the extension boundary. A diagram of the proposed hardware anchor enhanced architecture is shown in Figure 1.

3. EMULATOR-BASED PROTOTYPE

The immune system-inspired kernel defense approach involves an OS that directly downcalls the hardware to pass information about loading, unloading and memory boundaries of extensions. Upon being processed by the CPU these downcalls are forwarded to handlers at the architecture layer, which are also responsible for maintaining the information passed. Figure 1 shows Ianus’ high level view, which has the following key features: OS downcalls, downcall processing handlers, extensions’ information kept in the architecture layer, and a checker for memory writes and function calls.

Downcalls are direct calls from the OS to the CPU (Step 1 in Figure 1) and can have a variable number and type of parameters. Ianus has downcalls for extension loading, unloading, and dynamic allocation and deallocation of kernel memory. Every time an extension is loaded, the OS downcalls the CPU to pass the extension’s name and the address and size of its object code. The extension’s name uniquely identifies it in the system. When an extension allocates memory, the OS downcalls the CPU passing information about the address and size of the area. Memory dynamically allocated by extensions, despite being part of the kernel, do not receive the same level of protection given to original kernel data areas. The security policy adopted is to not allow kernel code and data being overwritten (bypassing kernel exported functions) by extension’s instructions, which are considered low integrity. However, extensions cannot be prevented from writing into their own allocated memory areas, which requires tracking at the architecture layer. When an extension frees memory, the OS downcalls the CPU to provide it with the memory region address. Memory deallocated by an extension is considered again a high-integrity area of kernel space.

Upon receiving a downcall the CPU delegates its processing to specific handlers (Step 2 in Figure 1), which create objects representing extensions and their attributes in the architecture layer. Extensions’ memory boundaries (a range of linear addresses) are kept in an internal hash table per extension at the architecture layer. When an extension is unloaded the handler destroys the corresponding extension’s object and removes the extension’s corresponding linear addresses from the extension’s hash table. Whenever kernel memory is allocated the handler checks if the instruction performing the allocation belongs to any of the extensions and if it does, the handler inserts this area into the extension’s hash table. Finally, when kernel memory is deallocated, the handler
3.1 Assumptions and Threat Model

This paradigm assumes an active OS which, like the immune system, is in charge of its own protection against its community of extensions with the support of the architecture layer. It is assumed that most kernel extensions are benign, but a small fraction of them will attempt to compromise the kernel and execute malicious actions.

It is also assumed an establishment time immediately after boot and all code and data present in the system before it are considered trustworthy. All extensions installed in the system are monitored, but they do not suffer any restriction on their execution, as long as they do not attempt to bypass kernel exported functions and write into the kernel code and data segments.

3.2 Implementation

An emulator-based proof-of-concept prototype, Ianus, was implemented to evaluate this architecture. Ianus used the Bochs x86 32-bit emulator as the architecture layer and Linux Ubuntu 10.04 kernel version 2.6.32 as the guest OS. Bochs was used due to its flexibility for performing architectural changes. The modifications in the guest OS consisted of a total of seven downcalls added to the kernel as assembly instructions. Bochs was extended with downcall processing handlers, an anchor instruction (downcall), data structures for keeping extensions’ attributes and range of memory areas, and a checker invoked in all functions performing writes in main memory.

3.2.1 OS Downcalls and Handlers

The downcalls are implemented as an unused software interrupt in the Intel 32-bit x86 architecture (unused vector 15). The INT n instruction generates a call to the interrupt or exception handler specified by the destination operand. This operand (the interrupt vector) is an 8-bit number from 0 to 255, which uniquely identifies the interrupt. Vector 15 is reserved by Intel and is not in use. The INT n instruction was modified to handle vector 15 and this new software interrupt is handled similarly to how system calls are processed with parameters passed in general purpose registers (EAX, EBX, ECX, EDX, ESI, EDI, and EBP).

The extensions’ downcalls were placed in the system calls sys_init_module and sys_delete_module. It was necessary to insert two different downcalls in sys_init_module because during extension loading, after the initialization function is invoked, the memory area corresponding to the extension’s init part is freed. The first downcall is placed after the extension’s initialization function is invoked and passes the extension’s name, and the address and size in bytes of the extension’s init and core parts. The corresponding handler adds the memory range of the init and core parts (as linear addresses) into the extensions’ hash table at the architecture layer.

After sys_init_module invokes the initialization function, a second downcall signals the architecture layer that the extension’s init part will be freed. The corresponding handler removes the init memory region from the extensions’ hash table. A downcall is also inserted in the system call sys_delete_module (invoked when an extension is unloaded) to remove the extension’s information (attributes and memory regions) from the architecture layer.

Downcalls were also placed into the kernel functions kmalloc() and vmalloc() to handle memory dynamically allocated by extensions. The addresses of the callers of these allocation functions were obtained using the __builtin_return_address gcc hack to the Linux kernel [17], and allowed the architecture layer handlers to discover whether the caller function belonged to any of the active extensions. This strategy allows the distinction between memory allocations made by an extension and by the original kernel. If the address of the caller belongs to any of the extensions, the handler adds this newly allocated memory region to the extensions’ hash table. Downcalls were also inserted in the corresponding deallocation functions kfree() and vfree(). The corresponding downcall handlers check whether the caller’s address belongs to any of the active extensions tracked and if it does, remove the freed memory range from the extensions’ hash table.

The downcall handlers at the architecture layer must translate virtual addresses from the OS into linear addresses. Each virtual address is represented by a segment and an offset inside this segment. The virtual addresses are included in the machine language instructions to specify the address of an operand or instruction. For example, in the assembly instruction MOV EBX, [EBX], the content of memory location given by register EBX is stored into register EDI. In this case, register EBX contains the offset of a virtual address in a particular segment. However, the security mechanisms at the architecture layer deal with linear addresses. In the Intel x86 architecture (used in this work) a linear address is a 32-bit number used to address a memory range up to 4 GB (addresses 0 to 2^{32} − 1).

Linux employs a limited form of segmentation by using three registers (cs, ds, and ss) to address code (CS), data (DS) and the stack (SS) segments. Processes running at user-level mode use these registers to address respectively the user code, data and stack segments. Code executing at kernel-level use these registers to address the kernel data, code and stack.

Each handler, upon receiving a virtual address from the OS in one of the general purpose registers must translate it into a linear address. The virtual address (segment and offset) is forwarded to the segmentation unit in the architecture layer and translated into a 32-bit linear address that can be used to index the extensions’ hash table. Downcalls passing memory addresses can refer to data structures stored in the kernel DS or code in the kernel CS. For instance, the name of an extension or the address of a dynamically allocated memory region are located in the kernel DS. An extension’s core and init parts reside in the kernel CS.

3.2.2 OS and Downcall Integrity

A key requirement of the proposed architecture is to guarantee the integrity of the OS downcalls. A kernel extension does not have privileges to issue downcalls and should not tamper with downcalls issued by the original kernel. This policy prevents a malicious extension from issuing a bogus downcall, or tampering with information passed by the original kernel to the architecture layer. These goals are addressed through the verification of all writes into kernel code and data segments and the origin of a downcall instruction.

The first security policy adopted is that kernel extensions are not allowed to perform write operations into the kernel code and data segments. The architecture layer contains a module for checking the validity of all write operations performed in the kernel code and data segments using Algorithm 1. This check is performed immediately before an instruction attempts to write a value into a memory location (m.Addr). The architectural functions that perform writes into memory were instrumented to invoke the checker before any write is performed in main memory.

Whenever a write operation is about to be performed, it is checked whether the write is being attempted at kernel mode. This is done by checking the CPL value, which is represented by a 2-bit field in the cs register. Then it is checked whether the linear ad-
Algorithm 1 Checker for the validity of write operations in main memory

Input: mAddr, the address to be written in memory and iAddr the address of the instruction performing the store operation.

Output: An exception is raised if the write is invalid.

if (CPL == 0) && (isADDRESSFROMEXTENSION(iAddr)) && (!isADDRESSFROMEXTENSION(mAddr)) then
if (segment ≠ SS) then
exception
else
if (lAddr > EBP) then
exception
end if
end if
end if

function isADDRESSFROMEXTENSION(addr)
for i = 0 to Extension.length do
if Extension[iAddr] then
return true
end if
end for
end function

dress of the instruction storing data (iAddr) into memory belongs to any of the extensions' memory regions monitored at the architecture layer. Next, it is checked whether the memory address being written (mAddr) belongs to an extension itself, which is not considered a security violation. Following, the segment being written is checked. If the write is attempted at the data or code segments, an exception is raised because it is considered a security violation (Step 4 in Figure 1). If the segment is SS (stack) it is checked whether the target address is higher than the current value of register EBP. If it is higher, this is an indication of a stack corruption attempt and an exception is raised. The kernel has the discretion to treat this policy violation the way it finds most appropriate. One possible action is to unload the offending extension from the system.

The integrity of downcall instructions is checked with architectural support. Upon execution of the INT $15 instruction it is checked whether the instruction bytes come from an extension. This is done by hashing the current instruction linear address to the hash tables that maintain the memory regions for extensions. If the instruction belongs to any of the extensions, an exception is raised.

3.2.3 Monitoring Non-Exported Function Calls

In the proposed architecture extensions interactions with kernel functions are monitored. Extensions are supposed to only invoke kernel exported functions. A function call checker intercepts all CALL instruction invoked by an extension and verifies whether its target address belongs to an exported function. If the target address of the CALL instruction corresponds to a non-exported function (from the kernel or other extensions) or even to an address that does not correspond to a function (an indication of a return-oriented attack [18]), the CPU raises an exception.

The addresses of all kernel functions (exported and non-exported) are obtained from the System.map file created during kernel compilation and made available to the architecture layer. Functions are distinguished from static data structures by the symbol type or by where the symbol is located. For example, if the symbol belongs to the text section, it corresponds to a function. Exported symbols are identified by their prefix: in System.map all exported symbols start with the prefix *ksymtab*.

Extensions also contain their own symbols (static data structures and functions) and can export some of them. Whenever an extension is installed, the OS extracts information about its symbols (names, types, virtual addresses and whether they are exported) and executes a downcall to make this information available to the architecture layer. Extensions' symbols information in Linux can be obtained through the *sym* field in the *module struct* (exported symbols), and through the extension's symbol tables contained in the *__ksymtab, __ksymtab_gpl* and *__kstrtab* sections of the extension code segment (all symbols) [19]. When an extension is unloaded, its symbols are removed from the kernel and from the architecture layer. The complete information about kernel and extension's symbols is kept at the architecture layer in a hash table indexed by the symbol's address (linear address).

Whenever the CPU is about to execute a *CALL* instruction, the following checks are performed. First the function call checker determines if the function invocation is being performed at kernel level. Following, it determines whether the *CALL* function comes from an extension's code. Next, it is determined whether the *CALL* target can be found in the symbols table at the architecture layer and whether the symbol is exported. If the symbol is not exported it is verified whether the extension invoking the function actually owns it, which is allowed. If those checks do not pass, an exception is raised. Another possibility is the target of the *CALL* instruction not belonging to any symbol in the kernel or in its extensions. In this case, the symbol is not a function, which is an indication of a return-oriented attack [18].

3.2.4 Monitoring Extensions’ Access to Data Structures

Algorithm 1 can be extended for fine-grained monitoring of how extensions access kernel data structures. Data structures allocated by the kernel should only be modified by extensions indirectly through kernel exported functions. A direct access to a kernel data structure is considered a security vulnerability. Knowledge about the boundaries of all dynamic and static data structures in main memory is required for the monitoring of read/write access to kernel data structures by extensions. The boundaries of kernel static data structures can be found in the System.map file as explained in section 3.2.3.

Dynamically allocated data structures, however, cannot be found in a static file such as System.map as they are unknown at compilation time. In Linux dynamic data structures are created via the slab allocator [20], which divides groups of objects into *cache* that store a different type of object. For example, the task_struct cache is a group of data structures of this type. In the slab allocator, three functions control the dynamic allocation/deallocation of kernel data structures: *kmem_cache_alloc, kmem_cache_free, and kmem_cache_free*. The first function, *kmem_cache_alloc* is invoked only once when the system is booted and creates a new cache for a particular type of data structure, e.g., *task_struct*. The second function *kmem_cache_alloc* is used to allocate memory for a new data structure for the cache type passed as argument (the cache type determines the type of the data structure). For example, the cache *mm_struct* is used to allocate data structures of that corresponding type. The third function *kmem_cache_free* is used to return the memory area allocated to a data structure to the corresponding cache when it needs to be deallocated.

In Iansus the slab allocator’s functions from the OS are instru...
mented to inform the architecture layer whenever a data structure of a particular type is created and freed. Two hash tables at the architecture layer are used to keep this information. The first, the Data Structure Type, is indexed by the data structure’s type (the cache name) and also contains its size in bytes. Whenever a new cache is created, a new entry is inserted in this table. The second hash table, called Data Structure, is indexed by the data structure’s linear address and keeps up-to-date information about the active data structures in the kernel. Whenever a kernel data structure is created, a new entry is inserted in this table. When a data structure is deallocated its entry is removed from the table.

In this extension, Algorithm 1 works as follows. After verifying that the segment being written is not the stack (segment ≠ SS), it checks whether the segment being written is the data segment (DS). If it is, the memory checker records in a file, which is available to the system administrator, the type of the data structure being accessed.

### 3.3 Aspects of the Hardware Implementation

In the current hardware implementation, the anchor is embedded in the open-source SPARC V8 processor, as per the diagram shown in Figure 2. It includes the cross-boundary interface, the software security checker, and hash tables to store system level intelligence, such as extensions boundaries and addresses of kernel functions and static data structures. The hardware anchor monitors all traffic through the processor and takes control when a recognized down-call is issued through the interface. When a recognized down-call is fetched, the anchor halts the Integer Unit (IU), transfers the control of incrementing the program counter to the Anchor Control Unit (ACU), and ports the instruction-cache (I-cache) and data-cache (D-cache) data wires to the ACU. In this way, the SPARC architecture specific interrupt handling is suspended and subsequent fetch, decode, and execution stages are controlled by the ACU only.

Many OS downcalls will pass virtual addresses through the anchor, for instance the initial virtual address of a recently loaded extension. Therefore, the ACU also acts to control the Memory Management Unit (MMU) by severing the MMU input/output lines from the processor and porting them to the anchor. Subsequent MMU virtual-to-physical translations will operate on virtual addresses passed by the OS through the anchor. The physical addresses are returned to the ACU to be stored as system level intelligence needed for enforcement of security policies. Once the ACU finishes operating on the passed OS intelligence, control is given back to the IU and the input/output lines are returned to normal functionality. The ACU acts as the processor control unit by directing data flow when an OS downcall is issued. The hash tables that store system-level intelligence are part of the hardware anchor and are not accessible to any other component of the processor.

### 4. EXPERIMENTAL EVALUATION

This section presents the results of the experiments validating Ianus. All experiments were executed in an Intel quad core 3.8 GHz with 16 GB RAM and running Linux Ubuntu 12.04 as host OS. Each performance experiment was executed ten times and the results were averaged. The evaluation assessed Ianus security and performance. The security analysis investigated whether Ianus guaranteed the integrity of the downcalls and the information passed through them, the protection level against kernel-level malware (rootkits), and whether or not it caused disruption in the normal operation of benign modules (false positives). The performance analysis investigated Ianus’ overhead to the system (OS, architecture layer and the two combined).

### 4.1 Security Analysis

Ianus’ security was analyzed against real rootkits that exercised the following security concerns: (i) tampering with kernel code and data, (ii) tampering with downcall parameters, and (iii) issuing bogus downcalls. False positives were also evaluated with benign kernel modules and drivers. Table 1 shows the rootkits tested in this evaluation. The last two rootkits in the table were implemented by the authors and this section details their interactions with Ianus’ security mechanisms.

#### 4.1.1 Tampering with Kernel

The authors implemented a kernel rootkit (General Keylogger) as a loadable kernel module (LKM) that attempts, like most rootkits, to corrupt the OS’s system call table. In its initialization function the rootkit replaces a pointer to a legitimate system call function with a pointer to its malicious version of the system call. This is a common type of kernel attack in spite of recent Linux versions attempting to make the system call table inaccessible to kernel extensions. The rootkit has keylogging capabilities and was based on the system call hijacking approach described in [21]. The rootkit hijacks the system call table by first locating its address in the kernel through brute force and writes into the system call table by first setting the CR0 register’s first bit to 0, which changes the CPU from protected to real mode. After tampering with the system call table, the rootkit puts the CPU back to protected mode. These actions were done with the kernel functions read_cr0 and write_cr0.

When this malicious extension is loaded, the architecture layer has complete information about its boundaries in memory. When the initialization function is invoked, one of its instructions attempts to perform a write operation in an area which is part of the kernel data segment. The goal is to overwrite this area with a malicious address into one of the slots of the system call table. The write operation is checked at the architecture level and it is detected that (i) it is performed in kernel mode, (ii) the target memory location is in the kernel data segment, (iii) the instruction bytes come from the text of the extension’s init part, and (iv) the memory area being written is not part of any extension’s dynamically allocated memory region. The write operation is aborted (thus preventing any compromise) with the CPU raising an exception handled by the OS. All other rootkits that operate by tampering with the system call table were stopped similarly.

#### 4.1.2 Issuing a Bogus Downcall

Here the goal was to evaluate whether or not kernel-level malware could issue bogus downcalls to the CPU. The authors im-
implemented a rootkit that attempted to perform a downcall passing fabricated parameters to the CPU. The downcall was issued in the rootkit’s initialization function. As in the previous examples, immediately before the initialization function is invoked the architecture layer is keeping track of all memory areas in use by the extension. The extension’s initialization function is invoked and issues a downcall causing the CPU to execute instruction INT $15. Upon executing the interrupt instruction the origin of its bytes is verified at the architecture layer by hashing the instruction linear address to the hash tables that maintain the extensions’ memory regions. The hash is a hit, which shows that the downcall is being issued by an extension, and an exception is raised.

The only rootkit Ianus was not able to contain was the kbd_notifier keylogger [22], which operates without the need to tamper with kernel code and data. It is a stealthy rootkit that works by registering a malicious function with the kernel keyboard notifier chain, which is invoked whenever a keyboard event occurs and allows the malware to record the keys pressed by a user at the kernel level.

4.1.3 Extensions’ Access to Kernel Functions

Another important aspect of the evaluation was to analyze Ianus’ behavior when executing kernel extensions. The common assumption is that benign extensions will only access kernel exported functions to perform their tasks. Table 2 illustrates the evaluation done with benign drivers installed during boot and benign extensions from SourceForge [23]. From the set of extensions analyzed, three benign drivers invoke non-exported functions from other extensions and the kernel. These issues caused the CPU to raise an exception to the OS.

Table 3 shows how real rootkits access kernel functions. In general, rootkits need to invoke a great number of kernel non-exported functions to operate. The only exception was the kbd_notifier keylogger.

4.1.4 Extensions’ Access to Kernel Data Structures

Table 4 shows that, in general, benign extensions and drivers do not directly access kernel data structures. The only exceptions were the parport and floppy drivers, which access the task_struct of a process. Rootkits, on the other hand, need to tamper with some kernel data structure to succeed and the vast majority of them tamper with the system call table. The only exception is the kbd_notifier keylogger [22], which operates without the need to tamper with kernel code and data.

4.2 Performance Analysis

This section analyzes Ianus’ performance impact in the whole system using system microbenchmarks from Unixbench [24] and a subset of the SPEC CPUINT2006 benchmark suite [25]. The execution times were normalized to the execution time of the system without any modifications to the OS and the Bochs x86 emulator. Using the unmodified Bochs as a basis for normalization allowed
Table 3. Extensions’ access to functions in kernel space - Rootkits.

<table>
<thead>
<tr>
<th>Module</th>
<th>Number of exported symbols</th>
<th>Non-exported functions invoked</th>
</tr>
</thead>
<tbody>
<tr>
<td>rkit</td>
<td>0</td>
<td>native_read_cr0(kernel)</td>
</tr>
<tr>
<td>bROOTus</td>
<td>0</td>
<td>__ticket_spin_unlock(kernel) sys_read(kernel) sys_getdents64(kernel)</td>
</tr>
<tr>
<td>KBeast</td>
<td>0</td>
<td>native_read_cr0(kernel) sys_read(kernel) sys_write(kernel) sys_getdents64(kernel)</td>
</tr>
<tr>
<td>kbd_notifier</td>
<td>0</td>
<td>native_read_cr0(kernel) sys_read(kernel) sys_getdents64(kernel) sys_open(kernel) sys_unlink(kernel) sys_unlinkat(kernel) sys_rename(kernel) sys_rmdir(kernel) sys_delete_module(kernel)</td>
</tr>
<tr>
<td>LVTES</td>
<td>0</td>
<td>native_read_cr0(kernel) sys_read(kernel) sys_getdents64(kernel) sys_open(kernel) sys_close(kernel)</td>
</tr>
</tbody>
</table>

Table 4. Extensions’ access to kernel data structures.

<table>
<thead>
<tr>
<th>Module</th>
<th>Access type</th>
<th>Data Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drivers loaded during boot</td>
<td></td>
<td></td>
</tr>
<tr>
<td>i2c_piix4</td>
<td>N/A</td>
<td>none</td>
</tr>
<tr>
<td>serio_raw</td>
<td>N/A</td>
<td>none</td>
</tr>
<tr>
<td>floppy</td>
<td>Read</td>
<td>task_struct</td>
</tr>
<tr>
<td>parport_pc</td>
<td>N/A</td>
<td>none</td>
</tr>
<tr>
<td>parport</td>
<td>Read</td>
<td>task_struct</td>
</tr>
<tr>
<td>psmouse</td>
<td>N/A</td>
<td>none</td>
</tr>
<tr>
<td>ppdev</td>
<td>N/A</td>
<td>none</td>
</tr>
<tr>
<td>lp</td>
<td>N/A</td>
<td>none</td>
</tr>
<tr>
<td>Benign modules from sourceforge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>frandom</td>
<td>N/A</td>
<td>none</td>
</tr>
<tr>
<td>tier</td>
<td>N/A</td>
<td>none</td>
</tr>
<tr>
<td>rxdsk</td>
<td>N/A</td>
<td>none</td>
</tr>
<tr>
<td>usb_vhci_ioclfc</td>
<td>N/A</td>
<td>none</td>
</tr>
<tr>
<td>usb_vhci_hcd</td>
<td>N/A</td>
<td>none</td>
</tr>
<tr>
<td>tty0tty</td>
<td>N/A</td>
<td>none</td>
</tr>
<tr>
<td>Rootkits</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rkit</td>
<td>Write/Read</td>
<td>sys_call_table</td>
</tr>
<tr>
<td>brootus</td>
<td>Write/Read</td>
<td>sys_call_table</td>
</tr>
<tr>
<td>ipsecs_kbeast_v1</td>
<td>Write/Read</td>
<td>sys_call_table</td>
</tr>
<tr>
<td>lpscs_kbeast_v1</td>
<td>Write/Read</td>
<td>sys_call_table</td>
</tr>
<tr>
<td>new_read</td>
<td>Write/Read</td>
<td>sys_call_table</td>
</tr>
<tr>
<td>kbd_notifier</td>
<td>N/A</td>
<td>none</td>
</tr>
<tr>
<td>hijack_syscall</td>
<td>Write/Read</td>
<td>sys_call_table</td>
</tr>
<tr>
<td>lvttes</td>
<td>Write/Read</td>
<td>sys_call_table</td>
</tr>
</tbody>
</table>

the evaluation to be focused on the actual overhead of the security mechanisms and not on the Bocsh overhead as an Intel x86 emulator.

Figure 3(a) shows the performance overhead of the OS downcalls during normal OS operation for Unixbench. These benchmarks exercised exec calls, file reads and writes (fsbuffer, fsys and fsalt), pipe throughput (pipe) and pipe-based context switch (context), process creation (apawn) and system call overhead (syscall). For these experiments Unixbench ran with the modified and the unmodified version of the guest OS. The goal here was to isolate the performance overhead of downcall issuing at the OS for intense system-level activity. Figure 3(a) shows that the overhead of downcall issuing at the OS is negligible (average 2%) for most of the system benchmarks.

Figure 3(b) shows the performance analysis for five benchmarks from SPEC CPUINT2006. The overhead was measured for two different situations. In the first (OS Downcalls), the system was running the modified version of the OS containing all downcall issuing. Here the goal was to evaluate the overhead to the OS for a CPU intensive benchmark. The second setting (Downcall handling) had the same configuration as the first, but now the downcalls were being processed by the handlers at the architecture layer. Figure 3(b) corroborates Figure 3(a) results in the sense that the downcall issuing overhead at the OS is very low. Downcall processing caused an increase of 12% on average to the execution time of the SPEC CPUINT benchmarks at the architecture layer. The overhead of 12% is low when we consider that certain types of applications that require a high level of security, (e.g., a power grid or a server at a national security agency), can trade performance for security. The hash tables at the architecture layer required less than 5 MB of main memory.

5. DISCUSSION

Similar to OS kernels, modern hardware design has increasingly relied on third party extensions. Under the current hardware design methodology, only the core processor/microprocessor designed in-house goes over full functionality and security testing and is considered trustworthy. Peripheral IP (Intellectual Property) modules and firmware extensions are often not fully tested to save cost, to shorten time-to-market, or to increase the reusability of the designed systems. However, third-parties IP cores can contain malicious logic and hardware also needs to co-exist with untrustworthy but needed extensions [26, 15, 27].

The proposed architecture can be broadened to include the hardware itself in the protection mechanisms. Hardware policies can be added to the architecture so that the hardware can be protected holistically with the aid of OS intelligence. OS intelligence can help define the trusted boundaries for legitimate operations of each IP module as well as the nominal time lengths of bus occupation in order to prevent false positives when detecting DoS (denial-of-service) style Trojans. For example, if data encryption is performed, then the key, the plaintext, and the intermediate results are sensitive information that only the cryptographic co-processor can access. The addresses of the sensitive information in memory are then passed to the hardware anchor as part of the OS intelligence. All other IP modules are prohibited from visiting the memory space where the sensitive data is stored. Including the hardware in the security policies also addresses concerns of HW-SW integration vulnerabilities: modern computer systems are not prepared to counter attacks that rely on the interaction between HW-SW pairs.

Another key innovation of the proposed HW-SW collaborative architecture is the ability to build a system with any combination
of OS and hardware security policies chosen by a system designer or integrator at the time the system is built. Each policy implementation contains an OS and a hardware component. At the hardware side, the security policies are all part of the anchor functionality, but are only effective if explicitly applied by the OS through the anchor. Under this architecture a security policy can be explicitly disabled by the OS due to performance or false positive concerns.

Benign kernel extensions coming from the network that modify kernel data structures without using an exported kernel function will have their actions reported as a kernel integrity violation. This situation can be overcome if such extensions are obtained from a network interface that the system considers high integrity or if they are installed in the system before the establishment time.

Attacks that do not need to write into kernel space to succeed [28] or that compromise a hypervisor [29, 30] or that write into kernel memory through trusted code paths in the kernel are beyond the scope of Ianus. Further, the target of JMP instructions, which can be leveraged by a rootkit to bypass Ianus’ checks, are not currently checked in the prototype.

6. RELATED WORK

A great amount of research has been done regarding hypervisor-based kernel protection and security solutions leveraging traditional introspection. This section discusses relevant work in VM-based introspection, kernel integrity defenses, and hardware support for system security.

6.1 Virtual Machine Introspection

VM-based intrusion detection systems leverage introspection in two ways: passive [5, 31] and active [9, 32, 33]. Passive introspection accesses a guest OS memory to reconstruct its state and abstractions. The OS state is recreated from low level data such as memory page frames.

Active introspection addresses better the semantic gap problem by allowing a more up-to-date view of a guest OS state. Xenprobes [32] and Lazes [9] place hooks inside the guest OS to intercept some key events, and invoke a security tool residing at VM level to treat the event. HIMA [33] is a VM-based agent to measure the integrity of a guest OS by intercepting system calls, interrupts and exceptions. All of these approaches (passive and active) consider the guest OS untrustworthy and do not actively interact or leverage it in the introspection process. This limits the amount and variety of system-level information that can be collected. L4 microkernels ([34]) functioning as a hypervisor also requires a L4-aware OS which downcalls the hypervisor and resembles the idea of downcalls of this work. The OS modifications turn system calls, memory and hardware accesses into calls to the hypervisor. Differently from L4 microkernel, the purpose of the OS downcalls in Ianus is exclusively for aiding security.

Recently, researchers have been working on better ways to perform traditional introspection, which is an error-prone and manual process. Chiueh et al [35] propose to inject stealthy kernel agents to a guest OS to enable virtual appliance architectures to perform guest-OS specific operations. Virtuoso [11] creates introspection tools for security applications with reduced human effort. SIM [36] enables security monitoring applications to be placed back in the untrusted guest OS for efficiency. It still suffers from the same semantic gap challenges as traditional introspection approaches because it was not designed to rely on data from the guest OS. Fu and Lin [37] apply system-wide instruction monitoring to automatically identify introspection data and redirect it to the in-guest kernel memory. A limitation is that certain types of data cannot be redirected, limiting the amount of guest OS information that can be obtained. Other line of work [38, 39], based on process migration, proposes to relocate a suspect process from inside the guest OS to run side by side with an out-of-VM security tool. The challenge is
that some processes are not suitable for migration.

### 6.2 Kernel Integrity Defense

Many authors have previously addressed kernel protection. The works focusing on prevention use some form of code attestation or policy to decide whether or not a piece of code can be executed in kernel mode. SecVisor [7] employs a hypervisor to ensure that only user-approved code executes in kernel mode: users supply a policy, which is checked against all code loaded into the kernel. NICKLE [8] uses a memory shadowing scheme to prevent unauthorized code from executing in kernel mode. A trusted VM maintains a shadow copy of the main memory for the guest OS and performs a shadow copy of the main memory for the guest OS and performs kernel code authentication so that only trusted code is copied to the shadow memory. During execution, instructions are fetched only from the shadow memory. Code attestation techniques [6] verify a piece of code before it gets loaded into the system.

Some approaches can offer some protection against non-control data attacks [40] that tamper with kernel data structures by directly injecting values into kernel memory. Livewire [5] is a VM architecture with policies for protecting certain parts of the kernel code section and the system call table. KernelGuard [41] prevents some dynamic data rootkit attacks by monitoring writes to selected data structures. Oliveira and Wu [42] used a performance expensive dynamic information flow tracking system (DIFT) and a set of shadow memories to prevent untrusted bytes to reach kernel space.

There are also many works addressing detection. Copilot [43] uses a PCI add-in card to access memory instead of relying on the kernel. Lykosid [44] and VMWatcher [45] perform detection based on a cross-view approach: hiding behavior is captured by comparing two snapshots of the same state at the same time but from two different points of view (one from the malware and the other not). OSck [46] protects the kernel by detecting violation in its invariants and monitoring its control flow.

The difference between this work and previous research in VM introspection and OS protection is that here the OS, like our immune system, has an active role in its protection against compromise from kernel extensions. The architectural layer acts only as a collaborative peer leveraging the key information about extensions passed by the OS to monitor the interactions of extensions and the original kernel. Having the OS in charge of monitoring itself streamlines kernel defense when compared to related work based on manual and error-prone introspection.

### 6.3 Extension Isolation

A great body of work in the literature focus on isolating or containing the effects and execution of device drivers and modules. Nooks [47] introduced the concept of shadow drivers and isolate them in a separate address space so that they can be recovered after a fault. HUKO [48] and Gateway [10] built on this idea by leveraging hypervisor support to protect the approach that define modules in a separate address space from a compromised OS. Ianus’ goals are similar to Nooks, HUKO, Gateway in the sense of protecting the kernel from malicious or misbehaving extensions. However, Ianus does not attempt to physically isolate the kernel from its extensions, but provide a way for them to co-exist. This provided much more flexibility to the system. For example, Section 4 showed that many drivers do invoke kernel (and other module’s) non-exported functions and their execution would be disrupted in HUKO or Gateway. Ianus can be fine tuned to allow more privileges to certain modules known to be benign. Also, the proposed architecture can be extended to include the hardware itself in the protection mechanisms.

Some lines of work advocate running drivers partially or entirely in user space. Ganapathy et al. [49] introduced the idea of micro-drivers in which drivers execute partly in user space and partly in the kernel. Nexus [50] and SUD [51] confine buggy or malicious device drivers by running them in user-level processes. Some works attempt to achieve driver isolation in software, such as SFI [52], where the object code of untrusted modules are rewritten to prevent their code from jumping to an address outside of their address space. The proposed architecture allows extensions to be executed without any modifications.

### 6.4 Hardware Infrastructure for System Security

Besides the software approaches, researchers have also proposed to rely on enhanced hardware infrastructure to protect system security. Hardware architectures to prevent memory corruption bugs and to prevent information leakage were developed for information flow integrity within computer systems. However, vulnerabilities were detected in these methods through which attackers can bypass the protection schemes. Chen et al. proposed a HW-SW architecture supporting flexible and high-level software policies for sensitive data protection. Similar approaches were also developed in the area of embedded systems, where limited on-chip resources are available [53, 54, 55, 56, 57, 58, 59, 60].

The main difference of these works and the proposed architecture is that these approaches work in isolation with the hardware and the OS. As discussed in Section 5, this collaborative architecture can be extended to include security policies to protect the hardware itself and offer a flexible set of security policies that can be customized during system build time. The proposed architecture offers a system builder great flexibility for balancing security, performance and false positives.

### 7. CONCLUSIONS

Current OS defense approaches are single-layered and operate in isolation with the hardware, failing to recognize its key capabilities for enforcing security policies. HW-SW cooperation is a promising approach to improve system trustworthiness because each layer has access to a distinct set of intelligence and functionality, which in combination can offer higher security guarantees than when operating independently. This paper discussed a HW-SW architecture for allowing an OS to safely co-exist with its extensions. This architecture is inspired by the immune system collaborative defense mechanisms that maintain a symbiotic relationship with its needed but untrustworthy microbiota.

A proof-of-concept prototype for this architecture, named Ianus, was implemented with Linux and the Bochs x86 emulator as the collaborative architecture layer. Ianus’ was studied with several real rootkits and benign extensions. In the experiments all malicious rootkits were stopped and Ianus caused no false positives for benign modules. Ianus’ performance was analyzed with system and CPU benchmarks and the system overhead was low (12% on average).

### Acknowledgments

This research is funded by the National Science Foundation under grant CNS-1149730.

### 8. REFERENCES


ABOUT THE AUTHORS:

Daniela Oliveira received her B.S. and M.S. degrees in Computer Science from the Federal University of Minas Gerais, Brazil, in 1999 and 2001, respectively, and the Ph.D. degree in Computer Science in 2010 from the University of California at Davis. She is an Associate Professor in the Department of Electrical and Computer Engineering at the University of Florida. Her main research interest is interdisciplinary computer security, where she employs successful ideas from other fields to make computer systems more secure. Her current research interests include employing biology and warfare strategies to protect operating systems in cooperation with the architecture layer. She is also interested in understanding the nature of software vulnerabilities and social engineering attacks. She is the recipient of the 2012 NSF CAREER Award and the 2012 United States Presidential Early Career Award for Scientists and Engineers (PECASE).

Nicholas Wetzel received his B.S. degree in Computer Science from Bowdoin College in 2014. He worked as an undergraduate research assistant at Daniela Oliveira’s cyber security lab at Bowdoin for one year before she moved to the University of Florida.

Max Bucci is currently a Computer Science major at Bowdoin College and expects to receive his B.Sc. degree in Computer Science in 2015. He worked as an undergraduate research assistant at Daniela Oliveira’s cyber security lab at Bowdoin for one year before she moved to the University of Florida.

Jesus Navarro received his B.S. degree in Computer Science from Bowdoin College in 2013. He worked as an undergraduate research assistant at Daniela Oliveira’s cyber security lab at Bowdoin for two years before she moved to the University of Florida. He works now as a software engineer at NVIDIA in Santa Clara, California.
Dean Sullivan received the B.S. degree in Electrical Engineering in 2013 and is currently pursuing the Ph.D. degree in Computer Engineering from the University of Central Florida. His current research focuses on designing security-enhanced computer architecture that protects both software and hardware. His research interests include computer architecture, secure HW/SW co-design, hardware security and trust, and VLSI design.

Yier Jin received his B.S. and M.S. degrees in Electrical Engineering from Zhejiang University, China, in 2005 and 2007, respectively, and the Ph.D. degree in Electrical Engineering in 2012 from Yale University. Since 2012, he has been an assistant professor in the EECS Department at the University of Central Florida. His research focuses on the areas of trusted embedded systems, trusted hardware intellectual property (IP) cores and hardware-software co-protection on computer systems. He is also interested in the security analysis on Cyber-Physical Systems (CPS) and Internet of Things (IoT) with particular emphasis on information integrity and privacy protection.
Early Response Markers from Video Games for Rehabilitation Strategies

Richard Davison², Sara Graziadio¹, Kholood Shalabi¹, Gary Ushaw², Graham Morgan², Janet Eyre¹

¹Institute of Neuroscience, ²School of Computing Science, Newcastle University, UK

{richard.davison4, sara.graziadio, k.m.a.shalabi, gary.ushaw, graham.morgan, janet.eyre}@newcastle.ac.uk

ABSTRACT
Stroke commonly leads to partial or complete paralysis of one side of the body and there is limited availability of therapists to provide rehabilitation. It is a priority therefore to identify the most effective rehabilitation strategies and/or pharmacotherapies. Motor learning, the essential process underpinning rehabilitation, can be assessed more quickly and robustly than outcomes from rehabilitation. In this paper we describe a proof of concept system utilising a commodity input device to play a bespoke video game to measure the critical components of motor learning. We demonstrate that we can detect how simple changes in therapist instruction significantly change motor performance and learning. Although video games have been shown to aid in rehabilitation, this is the first time video games have been used to derive early response markers, based on the measurement of performance and motor learning, for use in the evaluation of the efficacy of a rehabilitation strategy.¹

Categories and Subject Descriptors
J.3 [LIFE AND MEDICAL SCIENCES]: Health

General Terms
Measurement, Design, Experimentation, Human Factors

Keywords
Rehabilitation, video games, motor learning

1. INTRODUCTION
Stroke is a major global problem; the current prevalence of 60 million stroke survivors is predicted to rise to 77 million by 2030. Hemiparesis, a detrimental consequence that many stroke survivors face is the partial or complete paralysis of one side of the body that occurs due to brain injury. It is remarkably prevalent occurring acutely in 80% [1, 2]. Unfortunately, upper limb recovery is unacceptably poor with persisting impairments in 50-70% of stroke survivors [2, 3]. Although it has been established that intense rehabilitation therapy increases upper limb recovery, resource limitation is the main barrier to implementation of this evidence-base. Given the increasing prevalence of stroke survivors, it is a priority to identify the most effective rehabilitation strategies and/or pharmacotherapies that augment neuroplasticity in order to maximise a patient’s response to the available therapy time.

Motor learning is the essential process underpinning recovery after hemiplegia, either through relearning to use the paretic arm and hand and/or learning to compensate with the lesser-affected side. Furthermore, motor learning in normal subjects and functional neuroplasticity leading to post-stroke motor recovery have been shown to share the same underlying molecular and genetic substrates and brain networks [4]. Since motor learning can be assessed more quickly and more robustly than behavioural outcomes from rehabilitation of stroke survivors, assessment of performance and motor learning can provide an ideal marker of the biological system underpinning rehabilitation.

In this paper we describe a proof of concept system that utilises a bespoke video game to generate high spatial-temporal resolution data from players. Such data is key to measuring the critical components of motor learning that in turn provide early response markers. We demonstrate how our system is sufficiently sensitive to detect how simple changes in therapist instruction significantly influence the motor performance exhibited within-game player performance.

The literature presents numerous works indicating how serious games may aid in rehabilitation. However, the main focus of this paper is not to provide a rehabilitative game (where encouraging while recognising broad movement is sufficient), but to show, for the first time, an ability to measure early response markers to a clinical standard that are key to informing rehabilitative strategies. Such markers would otherwise be derived through time consuming, and costly, therapist observation.

The capacity to automate evaluation brought about by the utilisation of the approach described in this paper brings forward additional opportunities. One such opportunity is in allowing targeted screening of candidate drugs for repurposing into rehabilitation. This could be prior to initiating phase 2 or phase 3 trials, or patient stratification for clinical trials, based on level of performance and indices of motor learning.

2. SYSTEM OVERVIEW
Nearly all manipulation activities of daily living require the ability to flexibly perform multiple steps to achieve a unitary task. Thus, regaining the ability to perform efficiently linked action phases in manipulation tasks is highly important for rehabilitation.

In this paper we illustrate the use of the proposed system to assess whether the instructions given by a therapist to subjects learning a task requiring two action phases, either describing the task as having a single objective or by breaking the task down into

¹Copyright is held by the authors. This work is based on an earlier work: SAC'14 Proceedings of the 2014 ACM Symposium on Applied Computing, Copyright 2014 ACM 978-1-4503-2469-4/14/03. http://dx.doi.org/10.1145/2554850.2554953
its two sequential action phases, significantly affect performance and learning.

2.1 Bespoke video games

We have developed a series of bespoke video games that require learning of sequentially linked action phases and capture key features of natural manipulation tasks, namely spatial and temporal control and the requirement that each phase is completed before the next phase can be executed. Furthermore, each action phase utilises characteristics of movements performed in real life (controlled application and adaptation of forces, visuomotor integration and adaptation, feed forward planning and feedback correction).

The game (sample screenshots shown in figure 1) used in this evaluation involves moving a spacecraft (avatar) around a screen to destroy meteorites (targets). A patient controls the movement of the avatar via isometric forces applied to a joystick. In essence, the patient must move their avatar to the same location as a target when it appears. The destroy action is a gameplay element acted out onscreen to increase interest for the patient.

Two sequentially linked action phases (transfer phase and lock and track phase) are embedded in the game. Players initiate a trial by relaxing their hand to bring the avatar to a central home zone. Targets are then randomly presented at one of three locations, namely to the right, to the left, or centrally above the home position. The first phase comprises moving the avatar towards the target (transfer phase), once having achieved the target the player must then hold the avatar within the trajectory of the target for 1s (lock and track phase). Feedback of performance is provided as a score that builds up on the screen during each trial of 12 target presentations. The final score is then presented at the end of each trial.

2.2 The joystick

In our experiments we wanted to create a cost efficient solution using only commodity hardware. Our choice was a video game joystick easily available to purchase from most retail outlets.

The joystick chosen was a Saitek Pro Flight X-65F Combat Control System (PC) Joystick. This joystick is considered a “high-end” gaming device costing approximately $500. However, unlike the typical joystick most gamers are familiar with, the Saitek Pro Flight does not measure the degree of movement of the joystick itself (typically found in controllers for consoles such as PS3, and Xbox). Instead, the pressure and the direction of such pressure are measured (the joystick is actually unmoveable). For the purposes of our game we are concerned with the joystick measurements associated to kilogram-force (kgf), and the degree to which this force is measured in the x axis (left or right on joystick) and y axis (up and down on joystick).

Understanding how the joystick works is important for realising the actions patients are required to undertake to successfully complete the game. The joystick informs the game of both force and position via a kgf value in both the x and y coordinate. This is then mapped into 2D space \( \{x, y\} \) floating-point coordinates between 1 and -1 to represent points that exhibit both direction and magnitude (which is the force). Assume figure 2.a indicates the joystick at rest (no force applied) and the outer circle represents the most force the joystick could recognise in all x and y directions. Figure 2.b indicates force applied in the negative x (left) and positive y (up) directions. Figure 2.c indicates a more severe force than that applied in 2.b (due to its proximate to the outer circle) in the positive x (right) and negative force and direction.
y (down) directions. Therefore, a steady force maintained in a single, unwavering, direction will provide an unmoving cursor at some point, say \( \{P_x, P_y\} \), with the distance from the origin \( \{0, 0\} \) indicating force (relative to the maximum force achievable).

To ultimately accommodate patients with movement difficulties the joystick has been adapted so forces orientated to the right, to the left, up or down can be generated by movements of the supported, out-stretched hand and/or by the arm and shoulder. The patient does not grip the joystick, but rests their hand on a custom made support on top of the joystick. This allows patients with severe disability to apply pressure and participate in the game. The pressure generates an \( \{x, y\} \) coordinate indicating force and direction (as mentioned earlier). A translation is applied to these coordinates to map them to "screen space".

Screen space is not circular (reachability of \( \{x, y\} \) by joystick), and represents the resolution of the game graphics (1280 pixels wide by 800 pixels in height). Therefore, a basic scalar translation is applied to enlarge the joystick \( \{x, y\} \) coordinate beyond that of the screen area. As the derived coordinates of \( x \) and \( y \) generated by the joystick is a floating-point number between -1 to 1 in both \( x \) and \( y \), this can be achieved by multiplying \( x \) and \( y \) by 1280 and casting to integer values (for pixel alignment). If full force could be achieved this could result in \( y \) coordinates occurring “off-screen” (i.e., beyond 800 and -800). However, this does not occur as input from the joystick relating to magnitude/force (distance from origin) is capped to retain the avatar in-screen.

### 2.3 Joystick accuracy

We are primarily concerned with accuracy achieved by the joystick in the presence of wavering (small variable/changing weighting over time). This is because such weighting is expected from human provided force. Therefore, we took the technical specification as a guide and created our own experiments with weights to judge the accuracy of the joystick. The weights were applied and measurements taken.

The maximum amount of force detected on the \( x \) axis and \( y \) axis are selectable (that is, the joystick can be calibrated to the strength of an individual). The range of calibration is from 0.2 kilograms of force (kgf), to 10 kgf. As the joystick has a fixed output range of 12 bits in the driver software, higher maximum forces will result in a disproportionately lower precision of the reported force value. Hence, setting a maximum force close to that of the patient's maximum pushing force is necessary to achieve maximum fidelity of data output from the joystick; any less and we can’t detect maximum force and any more we waste bits of accuracy.

While stable the values reported from the device were found to have a small shifting offset. This necessitated tracking idle movement and running a continuous correction algorithm in software.

We detected the degree of shifting offset by considering joystick measurements at intervals of the following steps: (1) joystick at rest; (2) applying force; (3) release force; (4) measure; (5) return to step (1).

A 30 second sample of the device recorded at rest, with axis set to a force rating of 10kgf revealed a mean \( x \) axis position of -22.54 (stdev 1.24), and \( y \) axis position of -39.48 (stdev 1.45). The -22.54 offset on the \( x \) axis equated to the mechanism being off by approximately 0.1 kgf. By leaving the joystick alone and changing its force setting to have a maximum of 0.2kgf movement the offset rose to approximately 50% inaccuracy at the lowest setting.

As the device is handled and re-centered (used over time), it was discovered that the offset shifts slightly. A simple experiment was created to measure this secondary variation of the offset: pushed hard in one direction for 2 seconds, left idle for 2 seconds, and then pushed hard once more for 2 seconds in the alternate direction. Unfortunately, after two pushes to the maximum positive value, the idle value on release cannot be predicted (could be negative or positive). This means we could not accurately use an automated solution.

To accommodate the need to handle the non-determinism found in the offset not being predictable after joystick force a calibration stage was added to the game. During calibration the patient is required to take their hand off the joystick entirely for a short period of time. During this period of time the mean position is recorded on each axis. As we know that the joystick is now at rest the appropriate offset can be used (known for at rest) to derive the required alteration to the input from the joystick. This calibration period takes place after every target is completed, to minimize the effect of the shifting offset as the joystick is moved.

To compensate for players not paying attention to the instructions and not lifting their hand off correctly, or simply taking too long to perform the required action, the derivatives of the joystick values are recorded during this calibration period. Only samples with a low acceleration and jerk are considered for the mean calculation, within a specified Euclidean distance from the axis origin.

Once all possible solutions to managing the offset of the joystick are applied we proceeded to measure the actual accuracy of the kgf values reported. This was achieved by attaching weights around the joystick. We used 10 kgf as the calibration setting (as this sensitivity is within our requirements).

The device was recorded at idle in this state to create an offset to account for the weight of the attaching cable (used to place weights appropriately) and the effect of gravity on the stick itself. The results of this experiment are described in table 1. The table shows:

<table>
<thead>
<tr>
<th>True Value (g)</th>
<th>Mean (g)</th>
<th>Difference</th>
<th>Inaccuracy %</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>12.7994059</td>
<td>7.200594099</td>
<td>36.0029705</td>
</tr>
<tr>
<td>100</td>
<td>88.6966656</td>
<td>11.3033344</td>
<td>11.3033344</td>
</tr>
<tr>
<td>500</td>
<td>389.3997906</td>
<td>110.6002094</td>
<td>22.12004188</td>
</tr>
<tr>
<td>1250</td>
<td>1139.409556</td>
<td>110.5904438</td>
<td>8.847235502</td>
</tr>
<tr>
<td>2500</td>
<td>2274.842334</td>
<td>225.1576658</td>
<td>9.00630663</td>
</tr>
<tr>
<td>5000</td>
<td>4994.192271</td>
<td>5.807728934</td>
<td>0.116154579</td>
</tr>
</tbody>
</table>

We used the calculated weights to create a new driver software for the joystick. We found that the human provided force was not sufficient to move the joystick completely (in both axes), thus we were able to shift the joystick manually to achieve full movement. We found that the joystick was able to move approximately 50% of the time, and that this could be increased to 100% with the use of weights.
• Inaccuracy %: difference as a percentage

The difference between the hung weight and reported weight grows non-linearly, suggesting a limitation in detecting force. Although sensitive enough for our purposes, the accuracy of the joystick is certainly not of a level one would expect of an equivalent medical device. However, we have shown that we can correct these offset issues in software to enable the joystick to operate at a level of accuracy appropriate for our analysis.

2.4 Capturing data

Assuming that each hand will deviate in ability due to stroke the game calibrates separately for each hand. This allows further refinement of the coordinate system. The side more severely affected will not be able to reach the same degree of coordinates as the less affected side. Therefore, calibration takes this into account and applies an appropriate ratio to allow the same degree of “on-screen” movement for both hands. For example, if the left side can only apply a maximum of 5 kgf whereas the right side can apply a maximum of 20 kgf, then the left phase of gameplay will apply a 4:1 ratio multiplier to attain the same degree of movement on the screen.

Theoretically, when a target (meteorite) appears it is possible to apply the correct force and direction to the joystick so as to make the avatar appear immediately over the target. This is because we are not “moving” in the direction indicated by the joystick, rather we are placing the avatar on screen as directed by the \{x, y\} coordinates produced by the joystick. However, as humans naturally take time to react and build up force the avatar appears to move across the screen.

To achieve an appropriate fidelity of sampling to ensure no significant movement data is missed we sample the joystick at a high rate (124 times a second) with the game loop having the potential to achieve higher rates (500 times a second). A commercial video game typically runs at 60 frames a second. However, our desire for a much higher rate is driven by the need to rule out the loss of outlier measurements (sudden increase or decreases in pressure) that may occur. This also has the result of allowing a high fidelity of movement data to be considered within the gameplay itself, allowing the data that is used clinically to drive the game. Although the monitor/TV cannot show 500 full frames a second, we can still run the game at such a rate for accuracy of tracking the joystick and refresh the screen as and when required.

Accuracy tracking per-target is calculated as the mean distance the joystick \{x, y\} is from the actual target coordinates over the time the target is present. This measurement is returned to the coordinate system between (-1, 1) to trivialise comparisons across all targets. Time (as presented in the graphs) is measured in seconds.

2.5 Analysis system

Prior to starting the game each player undertakes a calibration procedure. The maximum pressure that the subject is able to generate by rotating their hand to the right or left and by palmer flexion is recorded for each hand separately. The pressure required in the task to reach the target at each position is then automatically set to be 10% of maximum respective pressure, to avoid fatigue during game play.

1) Transfer Time - the time between the appearance of the target on the screen and the avatar reaching the target. This index reflects predominantly feed-forward generation of movement with little opportunity to correct errors based on feedback.

2) Distance - the mean distance between the target and the avatar during the lock and track phase. This index reflects predominantly feedback mechanisms and error correction.

For both indexes lower values are associated with higher performances.

3. METHOD

The ethical committee of Newcastle University approved the study and written informed consent was obtained from participants. All participants were naïve to the experimental setup and objectives of the study.

3.1 Subjects

We compared two groups of 12 right-handed, young adult subjects (Group 1: mean age, 27 years, range 20-35 years; Group 2: mean age, 27 years, range 20-36 years). The video game and the controllers used and the environment in which the game was played were the same for both groups.

Group 1 (Single objective instruction group) were asked to play the video game with the single instruction to follow the target as accurately as possible; the feedback score reflected the accuracy of tracking the target.

Group 2 (Two step instruction group) were asked to play the video game with the instructions to move the avatar to the target and then to follow the target as accurately as possible. The feedback score for this group reflected both the time taken to transfer the avatar to the target and the accuracy of tracking thereafter.

3.2 Protocol

Initially the subjects play three trials within the game with each hand (non-dominant hand first) to assess their pre-training performance levels. This is followed by a session of training for the dominant hand, when the player undertakes 15 further trials. After completing training, the player undertakes a final 3 trials with their dominant (trained) hand to determine their post training performance levels. This is followed by 3 trials with their non-dominant hand (un-trained hand) to assess inter-limb transfer of skill from their trained to their untrained hand (a measure of generalisation of learning).

3.3 Data Analysis

Pre and Post training performance for each hand was assessed as the mean Transfer Time and the mean Distance for the first and last 3 trials respectively. Motor learning was assessed as the difference between pre-training performance and post-training performance for the trained hand. Inter-limb transfer of training from the trained hand to the non-trained hand was assessed as the difference pre and post-training performances for the untrained hand.
inter-limb transfer of dominant hand training occurred in both groups for both components of the task. A significant Group*Time post-training. There was also a main effect of performance for Group I (perceived single objective) both pre and (p=0.040) with greater transfer in Group 2 compared to Group 1.

4. RESULTS

4.1 Trained hand (TH)
There was a main effect of Group for Distance (p=0.012), with a significantly better performance in Group 1 (perceived single objective) than Group 2 (perceived two step task). There was a main effect for Time for both indexes (p<0.001 for Transfer Time and Distance), indicating motor learning for both groups. Whilst there was no main effect of Group for Transfer Time (p=0.256), there was a significant Group*Time interaction (p=0.015) with a trend towards a better performance for Group 1 (perceived single objective) prior to training (p=0.089) but no difference between groups after training (p=0.831).

4.2 Non-trained hand (nTH)
There was a main effect of Group for both indexes (Transfer Time, p=0.034; Distance, p=0.001) with significantly better performance for Group 1 (perceived single objective) both pre and post-training. There was also a main effect of Time in both indexes (Transfer Time, p<0.001; Distance, p<0.001) indicating inter-limb transfer of dominant hand training occurred in both groups for both components of the task. A significant Group*Time interaction was observed in the Transfer Time index (p=0.040) with greater transfer in Group 2 compared to Group 1.

5. DISCUSSION
In almost any training situation where action goals are to be learnt, including rehabilitation after stroke, instructions are given. In the present study we presented to both groups exactly the same sequences of stimuli and in response the participants performed the same two sequentially linked action phases. When the instructions and feedback emphasised the two subcomponents of the task rather than focusing on the single action goal (tracking the target), the performance of both action phases was significantly degraded. These findings add to a growing literature that the nature of the instructions given has a decisive influence on performance and/or learning of the action goal. For example, there is converging evidence that an external focus for instructions (i.e., a focus on the movement effect) is more effective than an internal focus (i.e., focus on the muscles activated to achieve components of the movements themselves) for both performance and learning (for review see [5]). In this study the two groups were given an external focus since both instructions focused on the desired movement effect.

The task studied in the present study involved two sequentially linked action phases. Although most manual tasks involved in activities of daily living comprise sequentially linked action phases, nearly all studies of manual control and learning concern single actions, such as simple reaction times or moving the hand between two positions. Thus, our understanding of how action phases are linked to perform the overall action goal, and how such linking affects learning, is limited.

Bernstein [6] first argued that action goals correspond to a pattern to be executed in external space rather than a sequence of specified muscle patterns. An easily verified demonstration that action goals are an abstract pattern is that one’s written signature has the same unique pattern whether generated by shoulder and arm muscles to write on a blackboard or by forearm and finger muscles to write on paper [7]. Klapp and Jagacinski [8] summarised recent research that supports action goals being represented as an abstract code that does not incorporate details of the sub-components required, but rather involves a single motor gestalt or chunk that is processed holistically. Additional findings indicate that the organization of action goals into chunks can be changed by instructions. For example, when reaction time was measured prior to the articulation of pseudowords [9] instructions that encouraged separate articulation of each of the syllables resulted in reaction times that increased as a function of the number of syllables. However, when the instructions favoured combining the syllables to form a single word, the reaction time did not increase as a function of the number of syllables implying that combining the syllables created a single motor gestalt so that the number of chunks is one, regardless of the number of syllables.

The results of our study mirrors these findings but in relation to a manipulative task; the transfer time was prolonged when the instructions favoured viewing the action goal as two separate tasks rather than as a single action goal. Furthermore, not only was the transfer timing prolonged but the accuracy of the tracking task was degraded, when the action goal was viewed as two separate tasks. These results support the concept that motor action goals might be represented as a single motor gestalt and indicate that task boundaries defining a motor chunk or single motor gestalt are not inherent to the action goal, but are ultimately determined by participants’ subjective representations of the task, shaped by the instructions given.
6. CONCLUSION

Video games have been shown to encourage stroke rehabilitation due to the simple fact that they require concentration of thought and some form of physical exertion (e.g., [10] [11]). However, there has not been a clinical view to deriving the required early markers to inform intervention, as described here, directly from the gaming hardware itself (in our case the joystick). Our most recent work [12] demonstrates an ability to determine change using video game devices benchmarked against therapist intervention. However, this paper goes further and investigates the validity of using off-the-shelf hardware coupled with bespoke video games to gain early response markers in the case of stroke rehabilitation.

This study provides proof of concept that video games and automated analysis systems can provide the capability to rapidly evaluate even subtle aspects of rehabilitation strategies, such as the content of instructions. We propose to develop a library of action phases, which can be easily combined and incorporated into bespoke video games together with automated data extraction and analysis to provide a flexible system for early response markers for evaluation of rehabilitation efficacy.

7. REFERENCES

ABOUT THE AUTHORS:

Richard Davison is a researcher in video game technology at Newcastle University. Having worked in industry, Richard has developed key skills in many technical aspects of video game production. His most recent work has focussed on rehabilitative techniques through the medium of video games.

Sara’s interest in Neuroscience arose during her undergraduate studies in physics at the University of Rome “La Sapienza” (Italy). She then moved to Neuroscience research and attended the Joseph Fourier University in Grenoble (France) as part of an Erasmus program and studied in the Neuroimaging field. Janet Eyre offered Sara the opportunity to undertake a PhD, supported by the Wellcome Trust, in her multidisciplinary team. After the PhD Sara took up a postdoctoral position at Newcastle University. Sara is currently working in the Institute of Neuroscience and in the Institute of Ageing and Health, exploring the use of video games for rehabilitative purposes.

Kholood Shalabi is a PhD candidate working between the Institute of Neuroscience and the School of Computing at Newcastle University. Kholood’s subject of study is the use of video games as a therapy in upper limb rehabilitative therapies.

Gary Ushaw received the B.Sc degree in Electronics from the University of Manchester Institute of Science and Technology (UMIST) in 1987, and the PhD in Signal Processing from the University of Edinburgh in 1995. From 1987 to 1991 he worked as an electrical engineer on large-scale public projects for communications and control, working with CEGB, British Rail, and the combined electricity generating boards of India. From 1995 to 2011, he worked in the games industry as software engineer and later as engineering manager, focusing on high-end console gaming with publishers including Ubisoft, Sony, Rockstar, BBC and Atari. In 2011 he joined the teaching staff at Newcastle University School of Computing Science, concentrating on video game engineering, rehabilitative gaming, and multicore systems.
Graham Morgan is a computing scientist specialising in systems research, in particular high-performance parallel computing over networked environments and multi-processor architectures; over the past ten years he has specialised in the application of these technologies to the field of video games. Graham works closely with the video game industry. This has resulted in the development of many mathematical and engineered technological solutions currently exploited by the video games industry. Recently attention of the research has turned to applying video game engine technology in the context of healthcare. This is unique globally, as all existing healthcare video game research and application has to rely on existing hardware and commercial off the self entertainment video games.

Janet Eyre was a Rhodes Scholar and previous Wellcome Senior Fellow in Clinical Science, she currently holds the Chair in Paediatric Neuroscience at Newcastle University and is a Consultant in Paediatric Neurology. Professor Eyre is an internationally recognised expert in brain plasticity following brain injury across the life span from birth to old age and its implications for rehabilitation. She has been awarded the following for her work in therapeutic video games: The NHS Innovations North Bright Ideas in Health Award 2009; CELS Business for Life Awards – Partnership with the NHS 2010; The UnLtd and Higher Education Funding Council for England Entrepreneur Award 2010; Medical Futures Best Innovation to Improve Patient Care – Cardiovascular Innovation Award 2011; Medical Futures National Health Service Innovation of the Year Award 2011.
Deployment and Activation of Faulty Components at Runtime for Testing Self-Recovery Mechanisms

Kiev Gama
Centro de Informática
Universidade Federal de Pernambuco (CIn/UFPE)
Recife, Brazil
kiev@cin.ufpe.br

Didier Donsez
LIG Laboratory
University of Grenoble
Grenoble, France
didier.donsez@imag.fr

ABSTRACT
The hypotheses on potential sources of error in an application can be specified in a fault model, which helps testing scenarios that are likely to be buggy. Based on a fault model, we developed custom fault detection mechanisms for providing self-recovery behavior in a component platform when third-party components behave inappropriately. In order to perform the tests for validating such mechanisms, it would be necessary to use a technique for fault injection so we could simulate faulty behavior. However such a technique may not be appropriate for a component-based approach. The behavior of systems tested with faults injected in the interface level (e.g., passing invalid parameters) would not represent actual application usage, thus significantly differing from cases where faults are injected in the component level (e.g., emulation of internal component errors). This paper presents our approach for testing self-adaptive mechanism, involving a general model for fault deployment and fault activation. Faulty components deployed at runtime represent faulty behaviors specified in the fault model. These faults are remotely activated through test probes that help testing the effectiveness of the platform’s self-adaptive mechanisms that are fired upon the detection of the specified faulty behavior.1

Categories and Subject Descriptors
D.2.5 [Software Engineering]: Testing and debugging – testing tools, error handling and recovery

General Terms
Reliability

Keywords
Fault model, self-recovery, testing.

1. INTRODUCTION
With systems becoming more and more complex, different research communities in computer science have concentrated on approaches that can minimize human intervention for system maintenance. One of the motivations to attain is a reduction of costs concerning installation, configuration, tuning up and maintenance of software applications. Self-adaptation or self-adaptive systems [26] is a more general term to denote systems employing autonomous mechanisms that generally do not involve direct human decision. Many related techniques are under the self-* (“self star”) flag for grouping them together (e.g., self-adaptation, self-configuration).

There are three major types of conditions enumerated in [8] to identify when systems would need to employ self-adaptation mechanisms: system errors, changes in the target system environment and changes in user preferences. Targeting the first scenario we can find self-healing systems, which are those that are able to detect when they are not operating correctly and automatically perform the necessary adjustments to restore themselves [15]. As stated before, the objective is to have no human intervention but if this is not the case, we can say that the system has assisted self-healing. In [15], the authors observe that while some scholars consider self-healing systems as an independent research branch, others include it as a subclass of fault-tolerant systems.

The implementation of a self-healing system may follow different architectural schemes, having several possibilities to be implemented [15]. But in general, such systems must be able to recover from a failed component by detecting and isolating it, taking it off line, fixing it (or keeping it isolated if not fixable), and reintroducing the fixed or replacement component into service without any apparent application disruption [14].

Despite the many advances in strategies for implementing self-adaptive software, testing and assurance are considered as the least focused phases in engineering such systems [30]. Techniques for verification and validation – which include testing – are included as important topics on the research roadmaps for self-adaptive systems [9][24]. According to Kephart and Chess [22], the construction of test systems that capture the size and complexity of realistic systems and workloads would be virtually impossible. What they suggest to be done instead is testing newly deployed autonomic elements in situ. They can be executed with established and trusted elements with similar functionality. We support the idea that conventional types of testing (e.g., unit testing, integration testing) may not be sufficient for verifying if self-adaptive mechanisms are working appropriately, especially in platforms that support the dynamic installation of components.

In this paper we describe the general-purpose mechanism created to verify if the autonomic behavior of a component sandbox with self-healing characteristics [12] was correctly implemented. Based on a fault model, we developed custom mechanisms for providing self-recovery behavior in the OSGi platform when third-party components behave inappropriately. In
order to perform the tests for validating such mechanisms, it would be necessary to use some fault injection approach so we could simulate the faulty behavior mapped in the fault model. As the technique for introducing faults into the system we chose not to use fault injection. Instead, faulty components were deployed at runtime, simulating scenarios where runtime software evolution (i.e., adding or removing components) would take place. We used the same fault model for creating test cases with the types of faults the platform was supposed to resist. It is important to note that, although we deployed components at runtime, we implemented an approach that focuses on testing the recovery mechanisms of the platform instead of testing components deployed on that platform.

The remainder of this article is organized as follows: section 2 provides more background on the system recovery subject; section 3 and briefly introduces our self-healing component sandbox; section 4 details the fault modeling approach; section 5 discusses the implementation of that approach; section 6 describes the validation; section 7 briefly discusses related approaches and finally, section 8 presents conclusions on the subject.

2. BACKGROUND ON SYSTEM RECOVERY

Recovery mechanisms of typical fault-tolerant techniques employ redundancy. By using such approach, when a component fails, a backup component or procedure can replace the failed component providing the correct computations without losing service. However, in the case of a failure due to external factors (e.g., hardware) that are not covered by the employed fault-tolerant mechanisms, the system may enter an inconsistent or unstable state. There is also a need for mechanisms that can restore system to its normal state. Recovery-oriented approaches try to tackle such issues by providing mechanisms that deal with a post-fault (or even post-failure) scenario. The next subsections provide an overview of two major techniques for handling those issues: self-healing systems and recovery-oriented computing.

2.1 Self-healing Systems

With systems becoming more and more complex, different research communities in computer science have concentrated on approaches that can minimize human intervention for system maintenance. One of the motivations to attain is a reduction of costs concerning installation, configuration, tuning up and maintenance of software applications. Self-adaptation or self-adaptive systems [26] is perhaps a more general term systems to denote systems employing autonomous mechanisms that generally do not involve direct human decision. We can find many related techniques usually under the self-* (“self star”) flag for grouping them together (e.g., self-adaptation, self-configuration, etc). There are three major types of conditions enumerated in [8] to identify when systems would need to employ self-adaptation mechanisms: system errors, changes in the target system environment and changes in user preferences. Targeting the first scenario we can find self-healing systems, which are those that are able to detect when they are not operating correctly and automatically perform the necessary adjustments to restore themselves [15]. As stated before, the objective is to have no human intervention but if this is not the case, we can say that the system has assisted self-healing. In [15], the authors observe that while some scholars consider self-healing systems as an independent research branch, others include it as a subclass of fault-tolerant systems.

The implementation of a self-healing system may follow different architectural schemes, having several possibilities to be implemented [15]. Generally, such systems must be able to recover from a failed component by detecting and isolating it, taking it off line, fixing or isolating it, and reintroducing the fixed or replacement component into service without any apparent application disruption [14].

2.1.1 Autonomic Computing

Following the self-* trend targeting adaptive mechanisms and less maintenance costs, a new research initiative called autonomic computing was started by IBM in the 2000’s. The term was inspired by the autonomic nervous system, for describing systems that are self-manageable. According to IBM’s vision [22], self-healing is one of the four main aspects of autonomic-computing, which also include self-configuration, self-optimization and self-protection. Self-healing consists of automatic detection, diagnosis and repair of software and hardware problems. Self-configuration is based on high-level policies, the system transparently reacts to internal or external events and adjusts its own configuration automatically. Self-optimization. concerns a system able to improve continuously its performance. Self-protection refers to the automatic anticipation and reaction of system wide failures due to malicious attacks or cascading failures which were not self-healed.

2.1.2 Autonomic Managers

Under the autonomic computing design proposed by IBM, these characteristics can be realized with the help of one or more autonomic managers. An autonomic manager (AM) is implemented using an intelligent control loop, based on feedback control theory. A managed element or managed resource can be hardware (e.g., a processor, an electronic device) or software (e.g., a component, a subsystem, a remote service). A managed element exposes manageability endpoints (touchpoints) that provide sensors and effectors [22]. Sensors provide data (e.g., memory consumption, current load) from the element while effectors allow performing operations such as reconfiguring. An autonomic element consists of one or more managed elements controlled by an AM that accesses the managed elements via their touchpoints.

Control loops, taken from control theory and control engineering, are important elements for building self-adaptive systems. They allow automated reasoning which involves a feedback loop with four key activities: collect, analyze, decide, and act [9]. IBM proposes a MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) control loop model (Figure 1) for constructing AMs. Their model is used as a one of the main references for autonomic control loops. Basically, the control loop monitors data (e.g., the inspection of system performance or current state) from a managed element; interprets them verifying if any changes need to be made; if so, the action needed is planned and executed by accessing the managed element’s effectors. Knowledge is a representation of live system information (e.g., an architectural model) that may be used and updated by any of the MAPE components, thus influencing decision taking.

An AM can also have just portions of its control loop to be automated [20]. Functionalities that are potentially automated could also be under manual supervision (e.g., decision taking upon certain events) of IT professionals. The administrators are also responsible for configuration, which can ideally [18] be done by means of high-level goals, usually expressed by means of event-condition-action (ECA) policies, goal policies or utility function policies.
Effectors increasing the mean time to fail (MTTF), ROC focuses on try to avoid applications from failing, that is, they concentrate on failure instead of trying not to fail. While typical research efforts by maintainability [11]. The purpose of ROC is dealing with applications dependability by reducing application recovery time (maintainability) thus increasing availability (directly influenced by human operators make mistakes, the ROC effort aims to enhance error’s cause; an application must recover from such errors. By acknowledging that hardware fails, that software has bugs and that these are considered as something that will eventually happen during application execution and no matter what was the error’s cause; an application must recover from such errors. By acknowledging that hardware fails, that software has bugs and that human operators make mistakes, the ROC effort aims to enhance applications dependability by reducing application recovery time (maintainability) thus increasing availability (directly influenced by maintainability) [11]. The purpose of ROC is dealing with failure instead of trying not to fail. While typical research efforts try to avoid applications from failing, that is, they concentrate on increasing the mean time to fail (MTTF), ROC focuses on reducing the mean time to repair (MTTR) with automated recovery mechanisms, avoiding the delays when human intervention is necessary.

2.2 Recovery-oriented Computing

The Recovery-Oriented Computing (ROC) research project, conducted by Stanford University and the University of Berkeley, employs principles that are similar to those of self-healing for improving system dependability of Internet services. Under the perspective of ROC [29], errors originated from people, hardware or software are considered as something that will eventually happen during application execution and no matter what was the error’s cause; an application must recover from such errors. By acknowledging that hardware fails, that software has bugs and that human operators make mistakes, the ROC effort aims to enhance applications dependability by reducing application recovery time (maintainability) thus increasing availability (directly influenced by maintainability) [11]. The purpose of ROC is dealing with failure instead of trying not to fail. While typical research efforts try to avoid applications from failing, that is, they concentrate on increasing the mean time to fail (MTTF), ROC focuses on reducing the mean time to repair (MTTR) with automated recovery mechanisms, avoiding the delays when human intervention is necessary.

2.2.1 Process Aging and Rejuvenation

The term software aging has been used by Parnas [28] to describe software that becomes obsolete due to lack of modifications or software that becomes complex and with a compromised performance because of a bad management on changes. In a sense more appropriate the context of ROC, software aging is also referred by [17] as process aging, which is the result of performance degradation or complete failure after software systems executing for a long time (e.g., hours, days).

ROC employs techniques that are related to Software Rejuvenation [17], which is a cost effective solution to avoid unanticipated software failures related with process aging. In order to prevent application failures from happening due to process aging, software rejuvenation works as a sort of preemptive rollback mechanism. It introduces proactive repairs that can be carried at the discretion of the user (e.g., when few or no users are connected to the application). The mechanism consists of gracefully terminating an application when it is idle, and immediately restarting it at a clean internal state. However, it is important to keep the application’s permanent state before terminating it. The goal is to clean up only inconsistent state resulted from non-deterministic faults without losing the correct application state, a principle also followed in ROC.

2.2.2 General Design Principles

According to the principles of ROC, software has to be developed taking into account that it will eventually fail, and it should facilitate its recovery. Some design principles are proposed in ROC:

- Recovery experiments to test repair mechanisms
- Diagnosing the causes of errors in live systems;
- Partitioning to fault containment and fast recovery from faults;
- Reversible systems to handle undo and provide a safety margin;
- Defense in depth in case the first line of defense does not contain an error;
- Redundancy to survive faults and failing fast to reduce MTTR.

ROC introduces the concept of crash-only software [3], which advocates that crash-only programs should be able to crash safely so they can recover quickly. It suggests the usage of fine grained components (crash-only components), state segregation, decoupling between components, retryable requests and leases. An important idea to retain is that this design admits that most failures are originated from non-deterministic faults and can be recovered by reboots. Therefore every suspicious component is “microrebooted” (further detailed in the next subsection). By employing such technique, components will be rebooted before the system fails. Also, by developing crash-only components the recovery process becomes very cheap.

2.2.3 Microreboots

Systems that run continuously for a long time tend to present performance degradation as well as an increase in failure occurrence rate [16]. Normally, hard to identify faults could be caused by diverse sources that are difficult to track such as race conditions, resource leaks or intermittent hardware errors. In such cases reboots are the only solution for reestablishing correct application execution and bring the system back to an acceptable state [6]. Several studies suggest that many failures can be recovered by rebooting, even when their cause is not known [4]. In [17], the authors show evidence that a significant amount of software errors are a consequence of peak conditions in workload, exception handling and timing. Such errors typically disappear upon software re-execution after clean-up and re-initialization. These are typical examples of non-deterministic faults, which we often face in our day-to-day experience as users of desktop and server applications as well as embedded systems. If we take the example of embedded systems of ordinary devices (e.g., portable phones, ADSL modems), in the presence of unattended behavior (e.g. unresponsiveness, freezing) the common user reaction to that is rebooting the device. After the restart is complete the device’s behavior comes back to normality.

Techniques such as Software Rejuvenation may be employed to avoid such scenarios in continuously running software that starts to degrade. However, while the software rejuvenation approach is of preventive nature, ROC proposes a mechanism that can act in a corrective way (after failing) as well as in a preventive way (before failing) like that other strategy. A practical recovery technique called Microreboot [4] [6] for the individual reboot of fine-grained components, achieves benefits similar to full application restarts but at much lower costs. Such approach increases application availability, because only one part of the application is restarted while the rest of the application is still executing. By employing this approach on individual
components, one introduces a significant delay avoiding a full application reboot, which can be employed as a last resort for recovering from non-deterministic faults when microreboots are no longer being effective.

In order to achieve safe microreboots, the crash-only principles must be taken into account. Applications should be designed with fine-grained components that are well-isolated and stateless. The microreboot design suggests the usage of a state store for keeping the state of components outside of them. By doing so, the process of state recovery is completely independent of application (i.e. component) recovery thus avoiding any state corruption when microrebooting components.

3. SELF-HEALING SANDBOX

Self-healing is one of the four characteristics desired in autonomic computing. Although that property may have overlapping objectives with self-protection, systems can employ autonomic computing principles that not necessarily provide all four desired characteristics. In previous work [12] we presented an architecture and implementation of a self-healing component sandbox for the execution of third party code. The approach extends the OSGi service platform [27] by focusing on dynamically loaded components that may potentially put in risk application stability. Different origins of such risks may relate to incompatibilities as well as undesired behavior of intentional (e.g., malicious components) or unintentional nature (e.g., programming errors). The platform was divided in two parts, as depicted in Figure 2: the main or trusted part (trusted components), and a sandbox (untrustworthy/unknown components). By executing code in a fault-contained sandbox, no faults are propagated to the trusted part of the application. The sandbox is monitored by a control loop that predicts and avoids known types of faults. If the sandbox crashes or hangs, it can be automatically recovered to normal activity without stopping the main application.

The autonomic manager is responsible for monitoring the sandbox and taking action to fix faulty scenarios when anomalies are detected according to our fault model. Although we presented a logical division of its components, the autonomic manager is implemented as a monolithic Java application. The monitoring, analysis and adaptations are performed by a MAPE-K control loop, which is the most common approach for self-adaptive systems [9]. A minor part of the monitoring role is also present in a watchdog component, while a significant part of the analysis and adaptation code was externalized from the control loop, and maintained as separate script files that could be changed during execution.

4. FAULT MODELING

4.1 Fault Model

The hypotheses on potential sources of error in an application can be specified in a fault model [1], which is a good predictor of faults, being very useful for effective testing and for fault detection mechanisms. According to Binder [1], two general types of fault model exist: conformance-directed testing and fault-directed testing. In our approach we focused on fault-directed testing, where the inconsistencies we addressed for employing a self-healing sandbox in OSGi applications were divided into three categories: resource consumption, library crashes and dangling objects. These bug hazards are the basis of our fault model. Although the design of a fault model does not imply in a visual representation, we illustrate our model as a UML class diagram, (Figure 3). The root class on the model generalizes the category of the covered problems as a faulty behavior (i.e., behavior that is not expected), which, for the sake of simplification, is a concept grouping faults, errors and failures. The higher level class and its two direct subclasses are abstract classes (in italics) used for generalization purposes, therefore they do not represent concrete cases of faulty behavior that we have modeled. The next two paragraphs briefly detail each one of those two abstract subclasses.

Unresponsiveness. Crashes and Application hangs were considered as possible cases of application unresponsiveness. According to the definitions found in [17], a system is considered as hung if it is unresponsive. This is different from a crashed system which characterizes a system whose process is abruptly stopped and is no longer in memory.

Resource usage. It was divided into CPU and memory categories. Both of them correspond to the excessive usage of the respective resource, but they can have specialized categories. Excessive thread allocation demands much more CPU usage for thread scheduling and context switching. Denial-of-Service (DoS) constitutes the excessive usage of a given resource in such a way that it is not able to serve other requesters, making the resource unavailable. We introduce the category of stale services in the diagram because it is the type of dangling object that we are able to detect and deal with in our mechanisms. Since it concerns a very specific type of memory inconsistency, we have considered it as a subcategory of memory issues, which is also true for other types of dangling objects that are not part of our fault model.
4.2 Fault Deployment and Activation Model

Fault injection is one of the methods to evaluate system dependability. Although tools can have different strategies for inserting faulty code into systems [19], they may not reflect the abnormal behavior that we want to avoid in self-healing systems (e.g., memory being overallocated, CPU hogs). After choosing a fault model, it must be determined how to inject faults into the computer system that will be tested [7]. However, such a technique may not be appropriate for a component-based approach. The behavior of systems tested with faults injected in the interface level (e.g., passing invalid parameters) significantly differs from faults injected in the component level (e.g., emulation of internal component errors), not representing actual application usage [25].

Because we deal with a dynamic platform where components can be installed during application execution, we rather focused on deploying faulty components instead of injecting faults into the system. Therefore, for testing the self-healing behavior of the recovery mechanism we should rather focus on test cases resembling component fault injection that can reflect possible faults happening in a realistic scenario. In our case, the term fault deployment would be more appropriate, since the dynamic platform allows components to be deployed and started at runtime. When the faulty components used in our approach are deployed, their faults are dormant. The faulty behavior can be activated through a remotely accessible probe.

A high level view of our proposed fault deployment and testing mechanism is illustrated in Figure 4. The elements in that general-purpose model are partly based on the basic components of a fault injection system [19]. The dashed line empty arrow means a dependency (rather in a conceptual way) while the solid lines means an association that refers to direct usage (e.g., the fault activator references the target system).

The faulty component represents the components that are coded to intentionally present the faults described in the fault model. The component must implement the fault activation interfaces, exposing them as probes to be activated remotely. The faulty behavior must remain dormant unless explicitly activated by the fault activator, which should be implemented as an external tool or agent that connects to the target system. The fault deployer can also be an external tool or agent that is responsible for deploying the faulty components in the target system. The feedback loop is responsible for monitoring the target system and in case the faulty behavior from the fault model being detected, it has to activate the appropriate recovery mechanism that acts upon the system.

![Figure 4. Conceptual model illustrating the fault deployment.](image-url)

5. IMPLEMENTATION

5.1 Fault Detection and Recovery

Self-healing systems should be able to recover from failed components, by detecting the faults and fixing the faulty behavior. Based on that fault model that was just presented, we are able to provide mechanisms for detecting such anomalies, but with current technology it is not possible to detect all of them at the component level. Component-based platforms do not provide features for individually measuring resource consumption of components. Therefore, we lack precision on some of the employed detection techniques. Because of these limitations we were able to identify the actual sources of faulty behavior only for the case of stale services, and denial of service because we could insert monitoring mechanisms in the OSGi service layer, which allowed us to track those problems. The rest of the faulty behaviors described in the fault model could be only identified in a general basis, without being able to exactly point out the objects or classes where the problem comes from. This limitation represents a major drawback for a sandbox that is shared with other components since the reboot penalty is for all sandbox, and therefore all of the components running there.

The detection of stale services has a particular mechanism that allows the identification of the problem at the component level, because it can be triggered by notifications when an unregistered service is invoked. Therefore, the detection of stale services is performed in two ways: by verifying it during each cycle of the control loop, or by receiving such notifications asynchronously. The former works as a fault forecast mechanism, but the current implementation involves imprecise heuristics for guessing the potential retainer of the service reference among the importers of the service interface package. The latter case would consist of fault detection and allows a more precise identification. The strategy found for fixing this behavior is more precise when the asynchronous notifications on stale service calls are sent. The exception thrown when invoking an invalidated proxy provides a stack trace that can be parsed, so the class name of the invoker can be identified. Finding the bundle that contains that class consists of a linear search of each bundle’s set of bundle entries (the list of contents) that match the name of the class. After identifying it, we perform a call to the update method in the bundle, performing a sort of microreboot. The bundle would be started and stopped, thus releasing the references to other services and recreating its bindings to a consistent state. However, this would not guarantee that the problem will not happen again, since the code of the bundle was not changed. There are attempts to avoid such problem in the OSGi platform, the proposition of the OSGi Micro Edition (ME) [2] describes the fact of not having stale references as one of the requirements in the specification.

The verification of denial of service behavior also relies on information captured in the OSGi service layer. The sandbox monitors that information by counting and logging service calls towards the main platform. In each control loop cycle of the autonomic manager, the corresponding verification in the policy will use the current value of total services invocation count and compare it with the last cycle. If it is greater than the maximum value configured, the policy script checks the sandbox log to verify if there is a particular service that is being overused or if this is just a overusage of the main platform through various services being called by the sandbox platform. If a single service is identified as the bottleneck for the excessive invocation, the policy verifies in the knowledge base for ActionEvents that have a
DoS behavior diagnosis containing that same service as a target. If it has already happened with that service, the strategy used is a script for invalidating that service. If it has never happened before, we perform a microreboot in the sandbox. Possible refinements on the diagnosis mechanism in the former case would be: (1) temporary invalidation of the overused service, which would be a temporary solution that could take place again; or (2) invalidate the proxy to that service, and provide a mechanism that would provide individual proxies per bundle, allowing to identify the misbehaving bundle next time the excessive usage of that service causes a DoS.

The other faulty behaviors mapped by the fault model cannot be precisely mapped to the “guilty” component. As already discussed, the information logged in the knowledge base can help making inferences of potential causers of a given anomaly, but is not enough for precisely identifying the origin of the problem. In case of detecting such behavior anomalies without knowing from which component it comes from, the technique Microreboots are still used as a resort for resetting the system state as an attempt to leave the faulty state. However, the granularity level of the microreboot is increased. Instead of an individual component, a subset of components (the ones that are active in the sandbox) is rebooted simultaneously. We can still consider the reboot as “micro” because part of the application keeps executing.

Concerning the unresponsiveness, its detection is performed by the watchdog component, outside the regular control loop as a separate surveillance mechanism. The other anomalies specified in the fault model are identified in the control loop during the analysis phase of the cycle. The script that contains the logic of the policy is executed and evaluates the read values against the defined thresholds, however a decision for performing a microreboot must not be based on the information of a single loop cycle. The knowledge base is used for verifying past loop cycles (e.g., during the last minute) and check if the threshold in question has been surpassed continuously.

5.2 Fault Deployment and Activation

The set of tests concerning the validation of the recovery mechanisms consisted in simulating scenarios using our fault model. Each one of the faulty behaviors mapped in the fault model must have at least one corresponding “fault implementation” that causes the respective fault. They can be implemented in individual components or freely grouped in a number of components. Our implementation of that approach relies on the Java Management Extension (JMX), which gives Java applications built-in support (e.g., standard API, component model, protocols) for management and monitoring. Through a remotely accessible interface, available as a JMX MBean component (manageable bean), we can access the faulty components and activate the faults, so the abnormal behavior can be presented and the diagnosis and recovery mechanisms can take action. The fault activation API we define was straightforward, consisting of an MBean with a name property and an activate method. Any faulty behavior had to implement that interface.

Figure 5 illustrates an example scenario with some of the components used in our test application. The OSGi components (called bundles in OSGi terminology) deployed with the faulty behavior publish the test probes as JMX MBeans in the MBeanServer, which allows external applications to access it through different connectors, in this case the default RMI connector. Through such management consoles it is possible to inspect the currently available MBeans and call their methods that trigger the faulty behavior. We accessed the MBeans through the VisualVM tool as depicted in Figure 7. The fault deployer is responsible for installing the faults in the running system, and can take different forms: an automated script, an administrator using a textual or graphical tool, etc. We used a custom plugin [13] for the Visual VM (an administration tool for the Java Virtual Machine) for deploying the faulty components. The faulty components contained code for registering the MBeans in the underlying MBeanServer, so they could be available to JMX tools.

6. VALIDATION

The target application scenario for validating our self-adaptive approach is illustrated in Figure 6, showing a simplified view of a network of machines constituting an RFID supply chain presented in previous work [21].

Figure 5. The test probes can be accessed by external tools such as monitoring consoles.

Figure 6. The scenario illustrates high availability requirements in the edge computers (circled) that collect data and also need to autonomously react to failures.

The figure illustrates the network infrastructure behind a supply chain where the products information comes from multiple
places. Elements with distinct roles comprise this network: edge servers, premises servers, EPCIS (Electronic Product Code Information Services) and ONS (Object Naming Service). In the context of our work, we focus on the edge servers (small computers connected to sensors) and RFID readers for capturing context data (e.g., temperature, vehicle weight, weather) and reading RFID tags, respectively. The middleware layer that is deployed in the edge captures that data, and sends it to other servers. It can be sent to an intermediary premise server (a warehouse), which filters data and possibly stores some information, or directly to the EPCIS which centralizes the information on all RFID tags previously scanned, allowing that data to be shared with other applications and different organizations, located anywhere with the help of an ONS.

6.1 Dependability Requirements

The edge servers may be located in relatively distant places (e.g., an entrance gate, a truck weigh station, a warehouse) where physical access by systems administrators is difficult, and where the people surrounding it (if anybody) may not be familiar with IT systems. Minimal human intervention is required, and in case of failure, the application must be able to recover from it autonomously. Applications could be remotely administered, however, manual surveillance of the systems may be time consuming, error prone and most of the times, unnecessary if the number of failures is relatively low (e.g., once a week). The integration of sensor devices and RFID readers into applications developed with Java (the technology used for that middleware) involves importing or wrapping native libraries (e.g., a device driver). Risks of failure are increased, but the system must be available and ready to scan tags and capture sensor data. Since third-party and native code runs in the application, risks increase and a recovery mechanism is necessary, so administrators have minimal intervention in such systems. Such human involvement should be kept to a minimal level, like distant software updates. A remote update is more appropriate than sending technical personnel to perform that task. However, such runtime updates can introduce undesired consequences to the application (e.g., incompatibilities involving drivers, device, and system components). Therefore, by taking into account these dependability requirements, where applications need a high level of availability, our platform had to provide mechanisms for reliability (reducing mean time to failure) and maintainability (reducing mean time to repair). These were the three most important requirements for the platform. In order to test if such mechanisms were successfully implemented in the platform, we simulated faulty scenarios that are presented in the next subsections.

6.2 Tests

Because we are focused in abnormal behavior, the tests had to be done in a controlled environment where we could manipulate the variables in order to reproduce the expected faulty behavior in accordance to our fault model. The scenario consisted of an OSGi application where the core components (e.g., reporting components, data filtering and gathering) of the edge need to provide high availability and are in the main (trustworthy) part of the platform. The untrustworthy components are hosted in the sandbox part of the platform that was running in the edge computer. Sensors and RFID reader simulator components were hosted in the sandbox. One the motivating scenarios concerns applications that collect RFID and sensor data. The application illustrates a scenario where we typically use native drivers wrapped in Java components to access physical devices. Devices may be plugged and detected at runtime, as are their respective drivers. The interaction between the application components that consume data provided by the untrustworthy code is done through OSGi’s service layer. In case of an illegal operation or a severe fault in the native code, the whole application is compromised. In this use case the application must also run non-stop and be able to recover in case of such severe faults and for doing so we employ, as a single solution, the different dependability aspects woven in the OSGi framework. The assumptions previously mentioned have to be true in this environment, so our approach can work correctly. Considering the recovery-oriented design principle, components and services used in the application are stateless. External devices contain the data (e.g., temperature, RFID tags), which is read by the components installed in the application. How they store data or how they guarantee that it will not be corrupted is out of the scope of our discussion.

Although profiling and monitoring suites are fundamental for tasks like tuning-up application performance, finding application bottlenecks and memory leaks, most of these tools do not take automatic administration decisions (e.g., performance adjustment, the detection of problems) during execution. Indeed, such tooling sets are powerful and some of them provide good levels of flexibility, allowing to easily use their infrastructure. In our experiments we have developed plugins for the VisualVM, in order to help us manage and monitor sandboxed OSGi applications. The MBeans that are illustrated in Figure 7 provide the test cases concerning most of the problems that were specified in the fault model that we propose. The fault model previously described was also used as a reference for implementing the test cases. Each test case was an OSGi component (bundle) providing code that triggers a faulty behavior. The following faulty behaviors were provided as OSGi bundles, as illustrated in Figure 6: Overutilization of CPU; application crash; excessive invocation of services (Denial of Service); excessive memory location; application hang; and excessive thread instantiation.

The monitoring functionality that allowed identifying basic resource usage such as memory allocation, CPU and thread instantiation were based on the ones provided by the Java platform. However, it does not provide resource usage perspective in the component granularity level neither provides much precision about the data monitoring, which significantly differs from the process. Because there is not much control about how much data a component is using, the policy implemented in the control loop is to simply reboot the whole sandbox when a threshold is exceeded.

6.3 Discussion

6.3.1 Experiment

Although the components were manually coded, the testing process can be automated since activating the faults can be done remotely. Since the tests can be fired through an interface that can be invoked (JMX-based probes), our approach can be more automated in the sense that it could be integrated to a test harness. Currently there is no specific test output. We need to perform an analysis of the control loop logs in order to establish a correlation of events, which is partially automated in one of the plugins we developed for our tool suit based on the Visual VM. Stale services retention is a fault possibility described in our fault model, but could not be activated through the same general purpose activation mechanism. Because that faulty behavior depends on lifecycle operations (install, update, stop, etc), its activation had to
be performed outside of our fault deployment and custom activation environment. That specific test was performed with the help of the Beanshell scripting engine. However, it would be possible to adapt it to that model by wrapping the lifecycle code inside a faulty component implementing the fault activation interface.

The implemented recovery mechanisms presented limitations concerning the precise identification of the components, this has no implications in the testing approach presented here. Actually, in the first test cycles minor implementation bugs could be diagnosed in some of the recovery mechanisms. Besides, using that approach makes it easier to simulate abnormal behavior that would be difficult to reach by using the system. An alternative is generating workloads with the help of external tools, in an attempt to reach the abnormal behavior that would fire the adaptation mechanisms. It is important to observe that our approach targets testing the recovery mechanisms of the platform, not the components deployed on the platform. Another point that is important to mention is that in case of faults not mapped by our fault model take place, most likely the system will not recover from them.

Figure 7. Probes for test activation seen in the Visual VM tool.

6.3.2 Evaluation of trust.

The evaluation of component trust is a difficult task especially if done during application execution. The purpose of this platform is not just hosting components in the sandbox ad vitam æternam. This mechanism is necessary for protecting the main application from the potential malfunctioning of other components. It is desired in some cases that the isolated component be promoted to a trustworthy status so they can execute as part of the main application.

We have not found a specific model or approach for automatically doing it at runtime. Injection of faults into interfaces between components, as used in [32], allows simulating the propagation of errors across components and how they behave on such scenarios. Approaches like that are appropriate to testing environments, though. Our target scenario consists of a production environment where components can be dynamically deployed at anytime, either by a system administrator or through an automatic mechanism.

As an appropriate strategy to the sandboxed OSGi approach, we support the idea of runtime observation of the component. During a sort of quarantine period, the interaction between the component and the system (i.e., other components) can be analyzed. After having enough historical information (e.g., Knowledge base) to be verified, the level of trust could be evaluated based on that analysis. The verification of historical data from the knowledge base can help to tell if the evaluated component introduced anomalies or undesired behavior in the system.

An obstacle to a precise evaluation, as discussed in the end of previous section, concerns the lack of fine-grained resource monitoring at the component level. Without that information, one cannot be sure about the resource consumption of a component. A second problem would concern the code coverage of these interactions. Part of our hypotheses considers the potential incompatibilities in compositions. Therefore, the historical data should also provide information that makes possible to take into account if, (1) all the methods of the services provided by the evaluated bundle have been invoked, and (2) if the services consumed by the evaluated bundle were already invoked. However, there are no guarantees that all of these interactions will take place during execution. Therefore, the minimal degree of coverage could be a criterion to be specified in this case.

If we take into account the possibility of hosting as a group in the sandbox all components that interact together, the decision of promotion (i.e., moving the component to the trusted container) would have to involve the whole set of cohesive components that are involved in related computations, otherwise by promoting one component only we would be generating performance penalties since the component would now have to interact through interprocess communication mechanisms.

Once the criteria are reasonably defined as well as the verification mechanisms, it would be possible to automatically promote components (if the fine-grained monitoring was available, of course). For now, the component promotion performed in our approach lies in human observation and decision before changing the policy for dynamically.

7. RELATED WORK

Voas [31] mentions that fault injection does not necessarily need automated tools, and discusses the important contributions that faults manually inserted in code can do for improving software quality. In our case, instead of injecting faults by changing the systems’ code, components manually coded with faults are deployed in the system.

Specifically focusing in self-adaptive systems, not many efforts are found around testing. The ones found in literature deal with self-testing. Denaro et al. [10] propose a testing infrastructure for tracking runtime information for components. That information is used in automatic testing of new versions of components and new configurations of component-based systems. Traon et al. [31] thinks of tests ahead in the design. Their self-testable components already contain test sequences and test oracles as part of their infrastructure.

A promising work found is that of King et al. [23], which employs a self-testing approach whose objective is to test the system after an adaptation has been perform. Their mechanism assumes that autonomic managers will perform the necessary system adjustments when they detect the need to adapt. However, this mechanism does not guarantee that the autonomic manager
correctly detected the system deviations that cause adaptation. Instead of testing/validating the adaptation, our approach tests if the autonomic manager is correctly detecting the deviations. Therefore we rather attempt to answer the first iteration of the recursive question “who verifies the verifier?”.

8. CONCLUSIONS
Self-adaptive systems employ autonomous mechanisms that generally do not involve direct human decision. Despite the advances in the field, the testing process of such systems is a topic that is not very explored. Even though fault injection is a mechanism typically used for testing systems, the behavior of systems that are tested using faults injected in the interface level would not represent an actual scenario of application usage. Therefore it would significantly differ from cases where faults are injected in the component level. We found that this is not the best mechanism for adaptive systems, especially if the target platform allows software evolution at runtime.

Based on a fault model, we developed custom mechanisms for providing self-recovery behavior in a dynamic component platform when third-party components behave inappropriately. In order to perform the tests for validating such mechanisms, it would be necessary to use a technique for fault injection so we could simulate faulty behavior. We defined a fault deployment and activation model that explicitly delimits the roles of the elements involved in the testing process. The tests could be activated remotely through test probes. The implementation of that approach allowed executing live tests in the running systems, so we could verify that the adaptive behavior was correctly functioning.

9. ACKNOWLEDGMENTS
The work presented here was initially carried out as part of the ASPIRE project, co-funded by the European Commission under the FP7 programme, contract 215417. This work was also supported by INES - Instituto Nacional de Ciência e Tecnologia para Engenharia de Software (http://www.ines.org.br/) financially supported by CNPq.

10. REFERENCES


ABOUT THE AUTHORS:

Didier Donsez is a professor in Computer Sciences at Université Grenoble 1 (France). He earned a doctorate (Ph.D) in Computer Sciences (1994) at University of Paris VI: Pierre et Marie Curie and a HDR (Habilitation à Diriger des Recherches) in Computer Sciences (2006) at Université Joseph Fourier. He was assistant professor in Computer Sciences at University of Valenciennes, France (1996-2001) at Université Joseph Fourier, Grenoble France (2001-2007). His does research at the ERODS team (Laboratoire d’Informatique de Grenoble), which is focused on efficient and robust distributed systems.

Kiev Gama is an assistant professor at the Federal University of Pernambuco (Brazil) since 2012, and a member of the Instituto Nacional de Engenharia de Software (INES). He received his Ph.D from the University of Grenoble (France) in Computer Science (2011). His research interests involve the usage of Software Engineering for constructing Distributed Systems in the context of the Internet of Things and Smart Cities.
Error Control Based Energy Minimization for Cooperative Communication in WSN

Syed Hassan Ahmed*, Dongkyun Kim†, Safdar Hussain Bouk‡
School of Computer Science and Engineering, Kyungpook National University
Daegu, Korea.
{hassan, bouk}@monet.knu.ac.kr, dongkyun@knu.ac.kr

ABSTRACT
As small-sized nodes are deployed in Wireless Sensor Networks (WSNs), efficient energy utilization becomes a fundamental challenge. Poor channel conditions lead to retransmissions resulting in unnecessary energy usage. To optimize energy consumption and improve network lifetime, error-control strategies are commonly required to manage channel impairments such as noise and fading. Cooperative communication is also considered a candidate solution to address the effects of channel fading. Therefore, the energy efficiency of cooperative schemes when applied with Automatic Repeat Request (ARQ), Hybrid-ARQ (HARQ), and Forward Error Correction (FEC) is investigated in this work. Moreover, expressions for energy efficiency of Direct Transmission, Single Relay Cooperation, and Multi Relay Cooperation are derived. Our work focuses on achieving energy-optimal communication in WSNs. Through simulations, it is observed that error-control strategies with a cooperative scheme can significantly enhance system performance in terms of energy optimization.

Categories and Subject Descriptors
C.2.1 [Computer-Communication Networks]: Network Architecture and Design-Wireless communication

General Terms
Algorithms, Design, Reliability, Performance

Keywords
ARQ, Cooperative Communication, ECC, Energy Efficiency, FEC, HARQ, WSNs.

1. INTRODUCTION
The advancement of technology has opened many new platforms for researchers to explore miniaturization. Wireless Sensor Networks (WSNs) have gained widespread attention in the recent past owing to their importance in many applications such as monitoring, signal processing, and health. The wireless sensor nodes deployed in a WSN field are small; however, they are capable of sensing different environmental and physical parameters and vital signs. They also include minimal processing and communication capabilities. These nodes are battery operated and generally, power constrained because of their small size. The lifetime of a battery is directly related to the size of a device. Therefore, a primary objective of WSNs is to prolong network lifetime by the efficient utilization of available energy resources[3, 11].

There are many strategies devised to optimize energy consumption in WSNs including efficient transceiver switching, routing, and self-organizing nodes in a WSN deployment area. The efficient switching of a transceiver requires that it be turned ON and OFF at the time of transmission and reception of data because a significant portion of the energy is consumed on transmission, reception, and continuous channel sensing. However, the switching of a transceiver does not guarantee energy conservation in the event that a transceiver is active and there is no data to be received [13].

Efficient routing and the division of a WSN deployment area into different regions to share data aggregation and communication responsibilities are also methods to minimize energy consumption. Periodic data aggregation, the transmission of the aggregated data over efficient routes, and switching of these responsibilities among different nodes can minimize energy consumption in the WSN. However, these techniques require efficient mechanisms to select optimal routes and divide the network into balanced groups to equally share the load among all the WSN nodes. This can be difficult in a WSN scenario [4]. Moreover, the above-mentioned solutions are prone to channel impairments resulting in the degradation of system performance. In the event of an error-prone channel, a WSN node frequently receives and discards corrupted packets and requests retransmission through Automatic Repeat Request (ARQ). This activity results in unnecessary energy usage and inefficient channel utilization [6, 12].

A well-investigated approach that effectively improves the link quality of an error-prone channel by reducing retransmission and packet error rate is the Forward Error Correction (FEC) technique. FEC can be implemented by two categorized coding schemes, namely convolutional codes and block code. Because convolutional codes have been extensively investigated, particularly in the WSN scenario [7], they are considerably simple and can be more easily implemented.

For stronger codes that provide better performance, the
complexity of the circuitry at the decoder increases, leading to the unnecessary usage of battery resources because of the additional processing power [5]. An energy-efficient decoder must consume less power compared to the power consumed in the reception of information and may consume slightly more power than a decoder with no error correction. Based on application requirements, the performance of FEC codes and ARQ is addressed in [19]. Hybrid-ARQ (HARQ), which is a combination of conventional ARQ and FEC, is well investigated in [10, 2, 9, 20]. A soft decision relaying function, Estimate-Forward (EF), is applied to relaying networks. Based on mutual information (MI), the throughput efficiency of HARQ is investigated in this work. Moreover, in [17], a detailed study is performed to investigate the packet transfer delay and energy efficiency of two basic retransmissions, hop-by-hop and end-to-end retransmissions.

In this paper, we investigate the energy efficiency of ARQ, HARQ, and FEC techniques for WSNs. The principal objective of this work is to analyze the impact of relay communications when implemented with these techniques. We consider different transmission scenarios including Direct Transmission (DT), Single Relay Cooperation (SRC), and Multi-Relay Cooperation (MRC). It is observed that MRC outperforms the other methods in terms of reduced transmission energy for a successful data transmission. Successful data transmission in SRC involves three transmitting nodes, namely, the source and two relay nodes. However, channel conditions between the source and destination and those between the source and relays are also important factors to be considered with distance separation between them. We evaluate performance based on different parameters including bit error rate (BER), symbol error rate (SER), Delay, and Throughput.

The remainder of this paper is organized as follows. Section 2 presents an overview of previous work on error correction and detection along with importance of cooperative communication when applied to WSNs. In section 3 Error Correction Codes (ECCs) are discussed and their energy consumption is highlighted. A System model with expressions of energy efficiency for DT, SRC and MRC is introduced in Section 4. Section 5 describes the simulation results in detail. Finally, concluding remarks are presented in Section 6.

2. RELATED WORK AND MOTIVATION

In wireless networks such as cellular or ad hoc networks, wireless agents may increase their effective quality of service (measured at the physical layer by bit error rates, block error rates, or outage probability) via cooperation. Cooperation leads to interesting compromises in code rates and transmit power. Regarding power, it may be argued that more power is required because each user in the cooperative mode is transmitting for both users. Conversely, the baseline-transmit power for both users is reduced because of diversity. Ideally, there is a net reduction of transmit power, assuming all other factor remain constant. Similar questions arise for the transmission rate of the system. In cooperative communications, each user transmits both their own bits and information for their partner. It would seem that this could cause a loss of transmission rate in the system. However, the spectral efficiency of each user improves because the channel code rates can be increased owing to cooperation diversity.

Significant work has been undertaken focusing on the energy efficiency optimization problem in WSNs. In [6], different types of ECCs were investigated and the power consumption along with the BER for various ECC in a WSN scenario were analyzed. The authors concluded that Reed Solomon code $RS(31,21)$ was an appropriate choice on the basis of BER and power consumption criteria. Similarly, a trade-off between transmission and processing energy consumption was investigated in [12]. The authors proposed and evaluated the hybrid scheme based on ARQ and FEC. The link error rates between different pair of network nodes were included in an evaluation metric. FEC schemes employing convolutional codes with code rate 1/2 were considered for evaluation. The authors optimized code complexity along each transmission hop and found that the optimal complexity along each hop minimizes the overall energy consumption and results in an increased network life-time.

Energy expenditure per data bit for both coded and uncoded system was discussed in [5]. This energy expenditure was taken as a function of critical distance $d_{CB}$, the distance where the particular decoder gets optimal energy consumption. Hence, at the distances greater than $d_{CB}$ for coded system results in net energy savings for WSNs. It was also found that analog decoders were typically better than digital decoders for the low-power requirements of WSNs.

A cross-layer analysis of error control for the intrinsic nature of WSNs, when broadcast and multi-hop routing were applied, was addressed [19]. A comprehensive comparison of FEC and ARQ was studied and cross layer effect of routing, medium access control and physical layer was considered. It was shown that the channel-aware cross-layer routing protocols techniques improve performance of WSN.

An incremental redundancy IR-HARQ protocol was discussed in detail by [9]. Possible use of Low Density Parity Check (LDPC) and Raptor codes for HARQ were studied in detail in this work.

In order to avoid retransmission for efficient usage of resources through memoryless relaying was defined by [20]. In this study, non-decoding relays were introduced for resource management. Similarly, number of transmissions resulting from frame error were identified in [15]. The work suggested the applicability of advanced FEC schemes in WSNs.

Figure 1 illustrates the working principle of ARQ, HARQ and FEC. Data is transmitted in the form of frames by a transmitter. Depending upon an error probability (EP), a receiver decides to reply with acknowledgement (ACK) for an error-free frame transmission or with negative-ACK (NACK) for a retransmission. As ACK and NACK use the same channel, they must also be protected by cyclic redundancy check (CRC). In the event of a lost or delayed frame, the sender must timeout and resend the previous frame. A sequence number (SeqNum) is used to avoid the reception of duplicate frames. The SeqNum and next frame expected (NFE) values are added in the frame header.

In type 1 (T1) HARQ, the error detection (ED) information and FEC is added before the transmission. On the reception of a coded data block, a receiver decodes the error correction code. With acceptable channel quality, transmission errors are correctable, and the correct data block can be received. Using error correction codes, channel condition is examined. In the event of inferior quality, a retransmission is requested, following the ARQ. Conversely, in type 2 (T2) HARQ, either ED bits or FEC information bits are transmitted on successive transmissions.
To avoid retransmissions, FEC is used in data transmission. By adding redundant data to the message, a receiver is able to detect and correct errors within certain bounds. FEC is categorized into Block Coding and Convolutional Coding. Block codes work on fixed-sized data blocks. Convolutional codes function with data streams of arbitrary lengths. In block codes, information bits are followed by parity bits, whereas convolutional codes require information bits to be spread along the sequence.

Figure 2 demonstrates an example of error correction and detection schemes. It indicates that ARQ, HARQ, and FEC can be used as ECC for better performance of communication systems. Depending on the application, any of these schemes can be implemented for efficient and reliable communications. SRC and MRC can also be implemented to utilize the energy of the sensor nodes more efficiently. Original data is encoded with any of the coding schemes. Then, it is transmitted following the principle of cooperative communication and is decoded on reception accordingly.

Motivated by previous work regarding the energy efficiency of WSNs, the impact of cooperative communication when applied with ARQ, HARQ, and FEC is investigated in this paper. Expressions for the energy efficiency of SRC and MRC are also derived. Moreover, the performance of the above-mentioned schemes is studied in detail. Based on our research, several results can be observed. The use of MRC can significantly enhance system output. In MRC, encoded data to be transmitted is divided into sub-frames and then transmitted through multiple paths. This multi-path transmission of data not only reduces the cost of retransmissions but also adds to the energy efficiency of the system.

3. ERROR DETECTION & CORRECTION

To decode received noisy bits correctly, one of the proposed solution is the addition of parity bits to the information sequence. This ability to correct errors in the received
information sequence is defined as Error Correction Coding (ECC). Through the usage of ECC, we can achieve better performance of BER for the same signal-to-noise (SNR) ratio. Furthermore, we can provide the same BER at a lower SNR than with an un-coded transmission. The SNR difference to achieve a certain BER for a particular coding and decoding algorithm, compared to an un-coded method, is defined as the coding gain for that code and decoding algorithm.

Minimum transmit power required for an un-coded system to achieve a desired BER at a given SNR can be found by [5] as:

\[ P_{TX,U}[W] = \eta_u 10^{(SNR_u/10 + RNF/10)} (KTB) \frac{E_b}{N_0} N(\frac{4\pi}{\lambda})^2 d^n \]  

where RNF defines receiver noise in dB, \( \eta_u \) is un-coded system’s spectral efficiency and \( SNR_u \) defines required SNR to achieve the target BER with an un-coded system. Required minimum transmit power when using ECC is given by:

\[ P_{TX,ECC}[W] = \eta_u B_c \frac{P_{TX,U}}{10^{ECC_{gain}/10}} = \frac{P_{TX,U}}{10^{ECC_{gain}/10}} \]  

where \( Eb_X = P_{TX}/R \) in J/bit is the required transmit energy per transmitted information bit, and is obtained by dividing required transmit power by \( P_{TX} \) with transmission R in bps. Difference between the minimum required transmit energy per information bit for coded and un-coded system defines transmit energy saving per information bit of the coded system and is given as:

\[ Eb_{TX,ECC}[j/\text{bit}] = \frac{P_{TX,U}E_{C}}{R} = \frac{Eb_{TX,U}10^{-ECC_{gain}/10}}{R} \]  

\[ Eb_{TX,U} - Eb_{TX,ECC} = Eb_{TX,U}(1 - 10^{-ECC_{gain}/10}) \]

Therefore, use of ECC optimizes the required minimum transmit power and energy per decoded bit because of the coding gain \( ECC_{gain} \).

4. SYSTEM MODEL

In this section, we discuss energy efficiency of DT, SRC and MRC. Moreover, expressions for energy consumption are also derived for all of these mentioned strategies.

4.1. ECONOMIC EFFICIENCY

As the power consumption of all nodes is constant and is denoted by \( P_t \), \( \alpha \) is the path loss exponent and noise components being modeled as additional AWGN with variance \( N_0 \). Received SNR (\( \gamma \)) for a link can be obtained by probability distribution function (PDF) given in [16] as:

\[ f_{\gamma_{ij}}(\gamma) = \frac{1}{\sigma_{ij}} \exp\left(-\frac{\gamma}{\sigma_{ij}}\right) \tag{8} \]

where average SNR (\( \sigma_{ij} \)) between node \( i \) and node \( j \) can be expressed as:

\[ \sigma_{ij} = \frac{P_\gamma (\gamma_{ij})^{\alpha}}{N_0} \]  

where \( \gamma_{ij} \) denotes distance of a link with node \( i \) and \( j \) being transmitter and receiver, respectively.

Let us assume that \( M-QAM \) is adapted with the modulation level \( b = \log_2 M \text{bits/symbol} \), the close-form expression for the average symbol error-rate (SER) of a link is given by [16] as:

\[ y_{RD} = y_{SD} + y_{SRof f}^1 + y_{Rof f}^2 \]  

Assuming that the transmitting power for all nodes is constant and is denoted by \( P_t \), \( \alpha \) is the path loss exponent and noise components being modeled as additional AWGN with variance \( N_0 \). Received SNR (\( \gamma \)) for a link can be obtained by probability distribution function (PDF) given in [16] as:

\[ f_{\gamma_{ij}}(\gamma) = \frac{1}{\sigma_{ij}} \exp\left(-\frac{\gamma}{\sigma_{ij}}\right) \tag{8} \]

where average SNR (\( \sigma_{ij} \)) between node \( i \) and node \( j \) can be expressed as:

\[ \sigma_{ij} = \frac{P_\gamma (\gamma_{ij})^{\alpha}}{N_0} \]  

where \( \gamma_{ij} \) denotes distance of a link with node \( i \) and \( j \) being transmitter and receiver, respectively.

Let us assume that \( M-QAM \) is adapted with the modulation level \( b = \log_2 M \text{bits/symbol} \), the closed-form expression for the average symbol error-rate (SER) of a link is given by [16] as:

\[ y_{RD} = y_{SD} + y_{SRof f}^1 + y_{Rof f}^2 \]  

Assuming that the transmitting power for all nodes is constant and is denoted by \( P_t \), \( \alpha \) is the path loss exponent and noise components being modeled as additional AWGN with variance \( N_0 \). Received SNR (\( \gamma \)) for a link can be obtained by probability distribution function (PDF) given in [16] as:

\[ f_{\gamma_{ij}}(\gamma) = \frac{1}{\sigma_{ij}} \exp\left(-\frac{\gamma}{\sigma_{ij}}\right) \tag{8} \]

where average SNR (\( \sigma_{ij} \)) between node \( i \) and node \( j \) can be expressed as:

\[ \sigma_{ij} = \frac{P_\gamma (\gamma_{ij})^{\alpha}}{N_0} \]  

where \( \gamma_{ij} \) denotes distance of a link with node \( i \) and \( j \) being transmitter and receiver, respectively.

Let us assume that \( M-QAM \) is adapted with the modulation level \( b = \log_2 M \text{bits/symbol} \), the closed-form expression for the average symbol error-rate (SER) of a link is given by [16] as:
\[ \text{SER}_{ij} \approx 2(1 - 2^{-b/2})(1 - \sqrt{\frac{3\sigma_{ij}}{2(2^b - 1) + 3\sigma_{ij}}}) \]  

Thus, PER of a link can be obtained as:

\[ \text{PER}_{ij} = 1 - (1 - \text{SER}_{ij})^{L/b} \]  

where \( L \) is the length of the data packet. Considering, PER of DT equal to PER of S-D link it can be written as:

\[ \text{PER}^D = \text{PER}_{sd} = 1 - (1 - \text{SER}_{sd})^{L/b} \]  

PER for SRC given by [16] can be evaluated as:

\[ \text{PER}_{SRC} = \text{PER}_{sd}\text{PER}_{sr} + \text{PER}_{sd}(1 - \text{PER}_{sr})\text{PER}_{rd} \]  

Based upon Eq. (13), we calculate PER for MRC as follows:

\[ \text{PER}_{MRC} = \text{PER}_{sd}\text{PER}_{sr1}\text{PER}_{sr2} + \text{PER}_{sd}(1 - \text{PER}_{sr1})\text{PER}_{rd1}\text{PER}_{sr2} \]  

From the above equation it is clear that transmission from S to D can be carried through multiple ways i.e., multiple paths like S-D and S-R-D are used for efficient communication and corresponding PER can be reduced. Moreover, energy efficiency of the overall system is expressed as:

\[ \eta = \frac{L_p(1 - \text{PER})}{E} \]  

where \( L_p \) is the data packet length, \( \eta \) defines transmit power of S, \( \text{PER} \) denotes packet error rate of DT, SRC and MRC, and \( E \) is the energy consumption for transporting a data packet with either of the above mentioned schemes.

Finally, the total energy consumed for transmitting one data packet with DT is expressed as:

\[ E^D = (P_t(1 + \beta) + P_{ct} + P_{cr})(\frac{L}{R_b}) \]  

Energy efficiency of DT is given by:

\[ \eta^D = \frac{L_p(1 - \text{PER}^D)}{E^D} \]  

Total consumed power for SRC to transmit a single packet is statistically described by [16] as:

\[ p^\text{SRC}_{total} = \begin{cases} p_t(1 + \beta) + P_{ct} + 2P_{cr} & \text{PER}_{sd}, \\ P_t(1 + \beta) + P_{ct} + 2P_{cr} & \text{PER}_{sd}\text{PER}_{sr}, \\ 2P_t(1 + \beta) + 2P_{ct} + 3P_{cr} & \text{PER}_{sd}(1 - \text{PER}_{sr}). \end{cases} \]  

The first term in above expression \((1 - \text{PER}_{sd})\) defines successful transmission over S-D link and receiving power of D and R is denoted by \( 2P_{cr} \). In the same way \((\text{PER}_{sd}\text{PER}_{sr})\) defines failure probability of both transmissions over the S-D and S-R links. At the end \((\text{PER}_{sd}(1 - \text{PER}_{sr}))\) defines failure of transmission over an S-D link. Therefore, the total consumed energy for transmitting one data packet with SRC is given as:

\[ E^c = \frac{1 - \text{PER}_{sd}}{R_b}(P_t(1 + \beta) + P_{ct} + 2P_{cr})L \]
\[ + \frac{\text{PER}_{sd}\text{PER}_{sr}}{R_b}(P_t(1 + \beta) + P_{ct} + 2P_{cr})L \]
\[ + \frac{\text{PER}_{sd}(1 - \text{PER}_{sr})}{R_b}(2P_t(1 + \beta) + 2P_{ct} + 3P_{cr})L \]

Thus, energy efficiency of SRC is given by:

\[ \eta^{SRC} = \frac{L_p(1 - \text{PER}^{SRC})}{E^{SRC}} \]  

By using Eq.(18), therefore, we calculate total consumed power for MRC in order to transmit one data packet as follows:

\[ p^{MRC}_{total} = \begin{cases} p_t(1 + \beta) + P_{ct} + 3P_{cr} & (1 - \text{PER}_{sd}), \\ P_t(1 + \beta) + P_{ct} + 3P_{cr} & (\text{PER}_{sd}\text{PER}_{sr1}\text{PER}_{sr2}), \\ 3P_t(1 + \beta) + 2P_{ct} + 3P_{cr} & (\text{PER}_{sd}(1 - \text{PER}_{sr1}\text{PER}_{sr2})). \end{cases} \]  

In above expression the term \( 3P_{cr} \) defines receiving power of D, \( R_{off} \) and \( R_{off}^2 \) and \( 3P_t \) defines transmit power of S, \( R_{off}^2 \) and \( R_{off}^2 \), respectively. Moreover, through (19) total consumed energy for transmitting one data packet with MRC is calculated as:

\[ E^c = \frac{1 - \text{PER}_{sd}}{R_b}(P_t(1 + \beta) + P_{ct} + 3P_{cr})L \]
\[ + \frac{\text{PER}_{sd}\text{PER}_{sr1}\text{PER}_{sr2}}{R_b}(P_t(1 + \beta) + P_{ct} + 3P_{cr})L \]
\[ + \frac{\text{PER}_{sd}(1 - \text{PER}_{sr1}\text{PER}_{sr2})}{R_b}(2P_t(1 + \beta) + 2P_{ct} + 3P_{cr})L \]

Thus, energy efficiency of MRC is calculated as:

\[ \eta^{MRC} = \frac{L_p(1 - \text{PER}^{MRC})}{E^{MRC}} \]  

To understand the importance of relay communication in WSNs for enhancement of network lifetime, we applied these schemes along with ARQ, HARQ and FEC. We considered the above system model and investigated the effect of relay communication for three different techniques named as ARQ, HARQ and FEC.

5. PERFORMANCE EVALUATION

We conducted performance comparison of the ARQ, HARQ and FEC methods. Moreover, we investigated the effect of off-path relays in the form of SRC and MRC for resource optimization in WSNs. In our simulations, a specialized form of Reed Solomon (RS) code was used for FEC. Our coding scheme assumed a code of RS (N, K) resulting in a code word of length N symbols, each having K bits of data. Stop and Wait ARQ with CRC-4 was also analyzed. Further, we evaluated the impact of SRC and MRC on HARQ. A (7,4) Hamming code was used in HARQ. We used MATLAB for the simulations.

In Figure 7, the Hamming code used to implement HARQ is evaluated. Figure 7 also shows the effect of HARQ encoding along with cooperative communication. A hard decision hamming encoder was used for HARQ. The BER analysis was carried in the presence of AWGN channel. As indicated in Figure 7, it is clear that data transmitted without any sort of coding has higher BER. Data encoded with hamming codes produced a lower BER that un-coded data. The simulation results illustrate that SRC and MRC significantly enhance system performance and achieve considerably improved BERs compared with the un-coded data and the data sent without cooperation.

Figure 5 provides a detailed description of RS codes being used as FEC. To evaluate FEC, a hard decision Reed Solomon encoder was used. The BER analysis was performed in the presence of an AWGN channel. From Figure 5, it can be observed that data transmitted without any
coding had a higher BER. Data, when transmitted with RS code encryption, achieved a better BER. The results demonstrate the impact of SRC and MRC with FEC encoding. It can be seen that the use of SRC and MRC significantly enhanced the system output and achieved considerably improved BERs when compared with un-coded data and data sent without cooperation. When applying SRC and MRC to a communication system, an optimized performance can be obtained.

In Figure 8, throughput analysis is shown as a function of BER for ARQ scheme. Throughput for Stop and wait ARQ along with DT, SRC and MRC is calculated. From the Figure 8, it is clear that at lower bit error rates throughput of direct path and SRC degrades whereas, MRC follows certain trends. MRC seems to be a good choice in both lower and higher BERs. Furthermore, comparison of these approaches with theoretical throughput also signifies the advantage of MRC more explicitly.

Figure 9 shows a delay analysis of ARQ scheme for both SRC and MRC. Stop and wait ARQ with CRC-4 detection was used in the simulations. The graph depicts the system BER versus transmission Delay. The delay of DT, SRC, and MRC is evaluated and compared against the theoretical results. As shown in Figure 9, for off-path relay, transmission delay may vary substantially depending upon relay position. For direct communication, however, this variation can be avoided, we determined that the use of MRC significantly increased the performance.

In Figure 10, symbol error rate of HARQ and HARQ along with SRC are analyzed for varying SNR. From the Figure 10, it is clear that for a given value of SNR, a limited improvement is observed in SER. This happens due to errors caused by fading over a single channel.

Figure 11 shows an analysis of HARQ scheme implemented using MRC. We assumed that the source and relay have an equal power level for transmission. T2 HARQ is investigated in the simulations. Figure 11 also shows that better symbol error rates are achieved when visualized in a given set of operating SNR values.
Figure 9. Analysis of Delay in ARQ using SRC and MRC

Figure 10. Symbol Error Rate versus SNR analysis of HARQ with HARQ with SRC

Figure 11. Symbol Error Rate versus SNR analysis of HARQ with HARQ with MRC

5.1 Fairness in Delay

The overall network delay of the ARQ scheme is analyzed in the graphs. Transmission delay is investigated for different BER values with a constant transmission timeout. The delays of DT, SRC, and MRC are also evaluated and compared against theoretical results. It is shown in the graphs that...
different timeouts produce different transmission delays. A lower number of retransmissions produces less transmission delay as they have less total round trip times for the transmissions. Moreover, the performance of MRC in all cases is better than DT and SRC. Figures 12, 13, and 14 present the performance of ARQ scheme in terms of transmission delay.

5.2 Fairness in Throughput

Figures 15, 16 and 17 present the performance of DT, SRC and MRC schemes in terms of throughput. The network throughput of DT, SRC and MRC only depends on BER. Different timeouts do no effect the overall throughput. Packets may be transmitted many times; however, only one copy of the packet is retained at the destination side. Therefore, the network throughput for all three cases only depends only on the SNR and BER.

6. CONCLUSION AND FUTURE WORK

In this work, the energy efficiency of three different error correction and detection strategies was studied when implemented with cooperative diversity. We investigated the applicability of ARQ, HARQ, and FEC for error correction and detection for WSNs. Moreover, we observed the significant impact of SRC and MRC when applied with these schemes. Through simulation results, it was revealed that ARQ, HARQ, and FEC with SRC and MRC can significantly improve system performance. We evaluated the performance in terms of throughput, delay, BER, and SER. Because the energy optimization of WSNs depends on a network topology, an optimal selection of relays for efficient communication must also be considered. For this purpose, we plan to extend our work to support dense networks more efficiently. Other works including [1], [8], [14], and [18] have proposed energy-efficient protocols without considering dense networks. Hence, the achievement of energy efficiency in dense network is a new challenge.

7. ACKNOWLEDGMENTS

This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the CITRC (Convergence Information Technology Research Center) support program (NIPA-2013-H0401-13-1005) supervised by the NIPA (National IT Industry Promotion Agency). The corresponding author of this manuscript is Dongkyun Kim.

8. REFERENCES


ABOUT THE AUTHORS:

Syed Hassan Ahmed completed his Bachelors from Kohat University of Science and Technology, Kohat, Pakistan in 2012. Afterwards, he joined CAST Research Centre as a distinguished Visiting Researcher in COMSATS Islamabad, Pakistan. Currently he is a PhD Student (integrated with Masters) at School of Computer Science and Engineering, Kyungpook National University, Daegu, Korea. During his Bachelors and Masters studies, he published several referred journal and conference papers in multiple topics. He is also IEEE/ACM member. His research interests include Wireless Sensor Networks, Cyber Physical Systems and Vehicular Communications, etc.

Dongkyun Kim is a professor with the Department of Computer Engineering, Kyungpook National University, Daegu, Korea. He received the B.S. degree at Kyungpook National University. He pursued his M.S. and Ph.D. degrees at Seoul National University, Korea. He was a visiting researcher at Georgia Institute of Technology, Atlanta, GA, USA. He also performed a post-doctorate program at University of California, Santa Cruz. He has been a TPC member of several IEEE conferences. He received the best paper award from the Korean Federation of Science and Technology Societies, 2002. His research interests are Ad-Hoc network, sensor network, and wireless LAN, etc.

Safdar Hussain Bouk was born in Larkana, Pakistan in 1977. He received the Bachelor’s degree in Computer Systems from Mehran University of Engineering and Technology, Jamshoro, Pakistan, in 2001 and Masters and PhD in Engineering from the Department of Information and Computer Science, Keio University, Yokohama, Japan, in 2007 and 2010, respectively. Currently he is a working as a Post Doc Fellow at Kyungpook National University, Daegu, Korea. His research interests include Wireless Ad-hoc, Sensor Networks and Underwater Sensor Networks.

Nadeem Javaid completed his Ph.D. from the University of Paris-Est, France in 2010. Previously, he has completed Masters in Electronics from Quid-I-Azam University, Islamabad in 1999. Before going to France, he has been teaching for more than eight years in different Universities of Pakistan; like, International Islamic University, COMSTAS Institute of IT, Hamdard University, etc. His research interests include Wireless Ad-hoc, Sensor, Vehicular and Body Area Networks.