Using Dependability Benchmarks to Support ISO/IEC SQuaRE

Jesús Friginal, David de Andrés, Juan-Carlos Ruiz  
STF-ITACA, Universitat Politècnica de València,  
Campus de Vera s/n, 46022, Spain  
Email: {jefrilo, ddandres, jcruiizg}@disca.upv.es

Regina Moraes  
School of Technology, University of Campinas - UNICAMP  
Campus I Limeira, Brazil  
Email: regina@ft.unicamp.br

Abstract—The integration of Commercial-Off-The-Shelf (COTS) components in software has reduced time-to-market and production costs, but selecting the most suitable component, among those available, remains still a challenging task. This selection process, typically named benchmarking, requires evaluating the behaviour of eligible components in operation, and ranking them attending to quality characteristics. Most existing benchmarks only provide measures characterising the behaviour of software systems in absence of faults ignoring the hard impact that both accidental and malicious faults have on software quality. However, since using COTS to build a system may motivate the emergence of dependability issues due to the interaction between components, benchmarking the system in presence of faults is essential. The recent ISO/IEC 25045 standard copes with this lack by considering accidental faults when assessing the recoverability capabilities of software systems. This paper proposes a dependability benchmarking approach to determine the impact that faults (noted as disturbances in the standard) either accidental or malicious may have on the quality features exhibited by software components. As will be shown, the usefulness of the approach embraces all evaluator profiles (developers, acquirers and third-party evaluators) identified in the ISO/IEC 25000 “SQuaRE” standard. The feasibility of the proposal is finally illustrated through the benchmarking of three distinct software components, which implement the OLSR protocol specification, competing for integration in a wireless mesh network.

Keywords—Dependability Benchmarking; Quality Evaluation; ISO/IEC SQuaRE;

I. INTRODUCTION

In a globalised context where software vendors from all over the world, compete for offering the most innovative, reliable, and cheapest products, reducing the time-to-market and development costs of software is an essential issue. This situation has promoted the use and integration of reusable third-party components, known as Commercial-Off-the-Shelf (COTS) components, in software products [1].

The use of COTS components imposes a set of quality considerations related to risks and uncertainties [2] that should be taken into account when integrating them in software solutions. Such risks derive from the lack of knowledge and documentation that one may have about the COTS components structure, development process and available APIs. Given such considerations and the huge variety of potential COTS components found today in the market for each specific need, the selection of the right alternative, among those available, remains a complex and critical task, since a wrong choice may result in an unaffordable set of economical, reputation and/or human losses [3].

Avoiding such wrong decisions is not easy and requires the consideration of performability requirements for all software components. This means that, in addition to conform with their functional requirements, software components must exhibit minimum levels of performance and dependability features to justify their quality. For any component meeting its functional specification, the greater its performability, the better its quality. This motivates the importance of evaluating the ability of each software component to meet its functional and performance specifications in absense, but also in presence of disturbances (faults and attacks) that may affect its regular behaviour during execution. It is worth noting that this issue goes beyond the mere test of the recoverability properties exhibited by a component. It encompasses the idea of including the notion of robustness in the concept of software quality as suggested in [4].

Dependability benchmarking is a well-known technique to evaluate the impact of faults over the various aspects considered for establishing the quality of software products, ranging from operating systems, to databases and web servers [5]. The basic idea is to define an evaluation process that (i) proposes a set of performance and dependability measures to characterise the quality of a particular software component, named the benchmark target, (ii) identifies the execution profile and experimentation setup to deploy, (iii) establishes the experimental procedure to follow, the time available for experimentation, and the procedure to retrieve, from the system under benchmarking, the measurements required to deduce the performance and dependability measures initially established, and finally (iv) provides guidelines to exploit such measures in order to establish a ranking among evaluated targets to select the most appropriate one for the considered solution. Fault and attack injection are basic techniques underpinning this type of benchmarking [6]. They are used to emulate during experimentation the occurrence of disturbances and observe the resulting system reaction, while keeping the reproducibility of experiments and minimising the level of spatial and temporal intrusive-ness induced in the target system.

The evaluation framework established by dependability
benchmarks is today partially integrated in the ISO/IEC Systems and software Quality Requirements and Evaluation (SQuaRE) standard, which defines an evaluation module, the ISO/IEC 25045, dealing with the assessment of the recoverability of software systems in the presence of accidental faults. Although, the ISO/IEC 25010 module underlines the importance of checking the security of software products, no attackloads are considered today in the current version of the ISO/IEC 25045 standard module. On the other hand, the assessment process described in the module only makes sense when the evaluated system provides fault tolerance, i.e. implements mechanisms to recover from the occurrence of faults. The interest of dependability benchmarks in other evaluation contexts, like those identified in the ISO/IEC 25040 module (development, acquisition and external evaluation), are not currently explored so far by the standard.

The present paper defends the interest of enriching the evaluation processes described in the SQuaRE 25040 standard module with dependability benchmarks establishing how to assess the quality of software components at runtime attending to, among other criteria, performability considerations. To cope with such objective, Section II performs a critical analysis of the ISO/IEC 25045, identifies its limitations and establishes the interest of dependability benchmarking for the different evaluation processes specified in the ISO/IEC 25040 standard module. Then, section III presents the various conceptual and technical modifications that one should introduce in such evaluation processes to exploit the potentials of dependability benchmarks for software quality evaluation, specially when a COTS must be selected. The feasibility of the approach is illustrated through the case study provided in section IV. Finally, section V reports on lessons learnt and provides conclusions.

II. CRITICAL ANALYSIS OF ISO/IEC 25045

ISO/IEC 25000 (SQuaRE) joins together the ISO/IEC 9126 and 14598 standards, conserving the core of the proposed quality models and the guidelines that are relevant for the software product quality evaluation. It defines a software quality model that has been used as a reference for evaluating the software (i) internal quality (static attributes of a software product satisfy stated needs), (ii) external quality (a software product enables the behaviour of a system to satisfy stated needs), and (iii) quality in use (a product can be used to meet users needs with effectiveness, efficiency, freedom from risk and satisfaction). In addition, SQuaRE complements the previous standard with some amendments, in particular, recognising the importance of security and dependability aspects [7].

Following this trend, the recently released ISO/IEC 25045 standard [8] has been incorporated into the SQuaRE series to provide the basis for the quality evaluation under the viewpoint of recoverability (a product can recover affected data and re-establish the state of the system in the event of a failure). This characteristic is of prime importance for evaluating the fault-tolerance capability of components and systems, and thus the ISO/IEC 25045 proposes a disturbance injection methodology for its characterisation. This methodology is structured in three phases: (i) the execution of the workload, addressed to exercise the system in absence of faults (baseline phase), (ii) the execution of the very same workload but now in the presence of the faultload (test phase), oriented to detect and exploit system vulnerabilities, and (iii) the comparison of the set of measures obtained in both baseline and test phases to assess the recoverability properties of the system.

Although these standards represent a step forward towards the inclusion of dependability characteristics in the quality of software products, there still exists a number of considerations that limits the completeness of the proposed methodologies. Dependability benchmarking approaches may complement these standards to widen their scope, beyond the mere testing of the recoverability properties of fault-tolerant software, to support the performability evaluation of even non-critical software products. The limitations that could be tackled by adapting the exiting concepts from the dependability benchmarking domain include:

1) Recoverability measures: The ISO/IEC 25045 standard only relies on two different measures (resiliency – ratio between the throughput obtained in absence and presence of faults, and autonomic recovery index – degree of automation in the system response against a threat) in order to evaluate the recoverability capabilities of components and systems. However, research in the field of recoverability performed from the dependability benchmarking viewpoint [9], highlights the importance of also characterising the recoverability of a system through its ability to detect, diagnose, repair, and restore the normal behaviour of the system. For instance, in case of suffering malicious attacks, the software product with the lowest detection and diagnosis times could be able to react faster and, thus, limit the effect of these disturbances in the system. Likewise, low repair and restoration times are interesting to reduce the system downtime. Hence, recoverability attributes should comprise all those measures that may support the software product selection for fault-tolerant systems.

2) Fault injection for quality evaluation: Despite fault injection being a well-known technique that has proven its usefulness in a wide range of software engineering domains [10], neither old ISO/IECs 9126 and 14598 nor the recent ISO/IEC SQuaRE considered its use until the appearance of ISO/IEC 25045. However, its application has been restricted to just the evaluation of the recoverability characteristics of the system. As dependability benchmarking states, the potential of fault injection goes beyond this particular dimension and is an indispensable tool to assess the quantitative degradation of the system behaviour in presence of disturbances. For instance, although different
products may achieve similar recoverability levels, the one obtaining better capacity, resource consumption, or timing behaviour in presence of disturbances is probably the best suited component for integration. Accordingly, the quality evaluation process will benefit from applying fault injection techniques for the assessment of other reliability subcharacteristics defined in ISO/IEC SQuaRE beyond recoverability, and other quality (sub)characteristics related to performance efficiency and security.

3) **Categories of faults:** The ISO/IEC 25045 standard just focuses on common high-level operational faults and events such as unexpected shutdown, resource contention, loss of data, load resolution and restart failures (as depicted in Figure 1 with a black wide dashed line). However, as shown in Figure 1, the disturbance scope in the context of dependability benchmarking comprises not only accidental faults, but also malicious ones (attacks). Indeed, the consideration of attacks in the dependability evaluation process is currently of prime importance, as components and systems are nowadays designed to ease their interconnection and facilitate the exchange of data. Hence, the proposed fault injection process must include also an attackload to evaluate the impact of attacks on product quality and enable the detection of existing vulnerabilities.

4) **Statistical significance of evaluation:** The statistical significance of quality evaluation processes is an important factor when validating the conformity of the results. For this reason, ISO/IEC 25045 proposes executing the baseline three times and considers it acceptable if the statistical significance among these three runs does not differ more than 5%. Although this procedure may be sound for the baseline phase, as the workload execution in absence of disturbance could probably be repeatable, this is not the case for the test phase. Even when executing the very same workload, results may greatly differ when distinct types of faults and attacks are injected in different locations (injection points) and moments (injection times). Existing research on dependability benchmarking could be useful to determine the required number of experiments to be performed, according to the eligible targets and the selected fault- and attackload, to ensure a certain level of consistency and stability on results that may vary depending on the system characteristics.

5) **Interpretation of measures:** The results of the application of the ISO/IEC 25045 standard should be presented as a report summarising the impact of faults in the system, giving a special attention to those weaknesses that should be improved. However, lessons learnt from dependability benchmarking experience show that analysing the huge amount of information resulting from the number of considered measures, faults, attacks, and scenarios, is quiet complex. That is why the systematic use of existing methodologies for guiding the interpretation of results when considering multidimensional measures, as presented in different dependability benchmarking studies, could be quite convenient for the comparison and selection of COTS software components.

As can be seen, dependability benchmarking concepts, methodologies and approaches, may be adapted to address the limitations of the ISO/IEC SQuaRE standard, beyond the mere assessment of the system recoverability, to support the evaluation, comparison and selection of COTS software components and systems in presence of disturbances.

### III. DEPENDABILITY BENCHMARKING APPROACH

Basically, a dependability benchmark is characterised through different dimensions [5], which define (i) the expected benchmark user and target system, (ii) the purpose of each considered measure and how to interpret them, (iii) how to configure the system under benchmarking and (iv) how to exercise the target component to experimentally obtain the measurements required to deduce the proposed measures. Our proposal focuses on adapting all these notions to introduce them into the general evaluation process defined in ISO/IEC 25040. The different stages of this dependability benchmarking approach and the activities deployed on each of them, depicted in Figure 2, are next detailed.

#### A. Establishment of the evaluation requirements

The proposed benchmarking methodology does not intend to automate the task of system operators when selecting a proper component at integration time, it rather tries to support and guide the comparison and selection of COTS components meeting the defined system requirements.

It must be noted that the software product quality requirements must also take into account the occurrence of faults and attacks in the system. Accordingly, acceptable limits (quality thresholds) for performance degradation or increased resource consumption, for instance, should also
be defined for systems that may continue working in a degraded mode.

Following this reasoning, and as COTS components are usually provided as black-boxes, this dependability benchmarking approach will determine the impact of faults and attacks on the functionality provided by COTS components (coarse-grained viewpoint), rather than focusing on internal aspects.

B. Evaluation specification

The evaluation specification defines the most appropriate measures to evaluate the component’s requirement and the experimental conditions (experimental profile) under which the experimentation will be performed.

1) Measures: Benchmark representativeness depends, to a large extent, on the set of measures selected to characterise the system behaviour and are, therefore, dependent on the evaluator needs. In this sense, our methodology relies on a subset from the 8 characteristics and 31 subcharacteristics of the software quality model defined by ISO/IEC 25010. As focusing on the impact of faults and attacks on the system behaviour, those quality characteristics related to the system operation, like performance efficiency, and those related to its dependability, like reliability and security, have been considered (see Table I). In practice, these abstract (sub)characteristics are quantified via their association with (one or more) measurable attributes, whose value is obtained from measurement functions, as Figure 3 depicts.

2) Decision criteria definition and application: Once measures have been computed, it is necessary to determine their degree of goodness within the applicative context by means of decision criteria defined in terms of numerical thresholds. As a wide amount of measures could hinder the results analysis, and so the components comparison and selection, our proposal considers the aggregation of measures as a valuable approach to complement the results obtained from evaluation and represent them in a friendly and easy-to-interpret way.

The Logic Score of Preference (LSP) [12], which has been successfully used for the quantitative quality evaluation of a wide variety of software engineering products (ranging from databases to web browsers), is a fuzzy-logic-based technique that computes the global score (S) of a component through measures aggregation via Equation 1.

\[
S = \left( \sum_{i=1}^{k} w_i s_i^r \right)^{1/r}
\]  

In Equation 1, each \( s_i \) represents an elementary score (referred to as elementary preference) obtained by each (sub)characteristic. Different criterion functions detail how to quantitatively evaluate each (sub)characteristic to establish an equivalence between its value and the system requirements within a 0-to-100 scale. Additionally, each elementary preference is associated to a weight \( w_i \) which determines its importance in the evaluation context and \( k \) is the amount of elementary scores considered in the aggregation, where \( \sum_{i=1}^{k} w_i = 1 \). The power \( r \) is a mathematical artifice, described in detail in [12], aiming at satisfying the logical relationship required by the evaluator for the different elementary scores within the same aggregation block.

Figure 2. Stages defined in our dependability benchmarking approach

![Figure 2](image1)

Figure 3. Relationship between quality characteristics and measures (extracted from ISO/IEC 25020)

![Figure 3](image2)
Finally, in order to complement the analytical results of LSP, we propose the use of graphical approaches, like Kiviat or radar diagrams [13], to represent the different measures in an easy-to-interpret footprint.

3) Work-, fault- and attack-load: The representativeness of results obtained throughout the proposed benchmarking process will depend, among other things, on the selection of a workload that matches as close as possible the real operation conditions of the final system. As real workloads (applications used in real environments) are really hard to obtain, and the representativeness of synthetic workloads is doubtful from a dependability viewpoint, we propound the use of carefully designed realistic workloads (based on real applications) to obtain a suitable degree of representativeness.

Likewise, the different disturbances a system may be subjected to during benchmarking should reflect as much as possible those that may affect the system during its normal operation in the applicative context it is deployed on (e.g., web servers, databases, communication networks, etc.). Typically, only those accidental faults and stressful conditions experienced by systems in the field, which constitute the faultload, were considered in the standard. However, the great impact that misuse and malicious attacks may have on the expected behaviour of the system, has led as to consider the necessity of incorporating an attackload into the proposed benchmarking approach.

C. Evaluation design, execution and provision of results

The evaluation design must consider practical aspects, like the experimental representativeness, observability and intrusiveness, to successfully deploy the evaluation process. Among these aspects, determining the proper amount of experiments that should be performed to increase the confidence on the obtained results is of paramount importance but has been obviated by the standard.

Our proposal relies on Equation 2, which is typically used in statistical testing, to determine the number of single disturbances \(N\) to be injected for each considered fault/attack model. \(P\) is the lowest probability for a single disturbance to affect a given element. \(Q\) reflects the probability of statistically targeting that particular element at least once during experimentation.

\[
N(Q, P) = \frac{\ln(1 - Q)}{\ln(1 - P)}
\]  

(2)

In case that it is not feasible to perform the required number of experiments, due to its huge number or its long duration, we recommend that the number of experiments performed should be bounded by the manpower involved and the time available for experimentation.

Once designed, the experimental evaluation can be performed following the proposed four-fold approach depicted in Figure 4.

![Figure 4. Experimental procedure](image)

During the initialisation phase, the component under benchmarking is configured and deployed within the system, which is also properly configured.

After this, the baseline phase, known as Golden Run in the dependability domain [5], starts. It consists of a number of successive experiments in which the system executes the selected workload, in absence of disturbances, and the component activity is monitored. Each experiment presents an initial set-up time, required to restore the original state of the system, followed by a warm-up time, devoted to lead the system to a stable state.

The test phase corresponds to the execution of the workload in the presence of the disturbance-load to evaluate the impact of these disturbances on the system behaviour. Its execution is identical to the baseline phase, but introducing the fault/attack into the system at a particular location at a given time. To ease the analysis of results, our approach considers that a test phase is related to one particular type of fault/attack, thus requiring additional test phases in case of considering more disturbances.

At the end of the experimentation, the check phase analyses all the log files generated with information collected while monitoring the system activity. In order to reduce the degree of intrusiveness on measurements, all this information will be offline processed, filtered and correlated, to extract the measures required to quantify the system behaviour, their average values and variability indicators such as the standard deviation or confidence intervals.

To ease the comparison process once the different eligible candidates have been evaluated, these results will be accompanied by their LSP global score and their corresponding Kiviat graphs.

IV. Case Study

Wireless multi-hop (also known as ad hoc) networks are spontaneous, dynamic and self-managed networks comprising all sort of (mobile) devices which collaborate to maintain network connectivity without any centralised infrastructure, thus saving the time and money required to deploy it. Routing protocols are essential software components in charge of establishing communication routes among nodes. Due to the nature of these kind of networks, it is useless to evaluate the quality of available routing protocols (more than 50 specifications) during the network operation in absence of disturbances, since the open wireless communication
medium and the absence of fixed infrastructure (certification authorities) make them really sensitive to accidental faults (e.g. ambient noise or low battery) and malicious attacks (e.g. jellyfish or tampering). Hence, the feasibility and usefulness of the proposed dependableability benchmarking approach will be shown through a case study involving the selection of the most suitable routing protocol implementation to be deployed for a given ad hoc network.

### A. Evaluation requirements

The purpose of this case study is the pilot deployment of a simple ad hoc network within the Department of Systems Data Processing and Computers of our university (UPV) as a previous step to its deployment on scenarios where the quick, easy, and low-cost Internet connectivity is today a necessity. Prior experiences developed by members of the same department in African schools can be consulted in [14].

As previously stated, system integrators must carefully select the most suitable routing protocol to provide quality communications without delays nor interruptions. olsrd, developed by the most active and wider community devoted to the deployment of open-source routing protocols (www.olsr.org), is the most extended implementation of the popular Optimized Link State Routing (OLSR) protocol. Accordingly, three different versions of olsrd have been selected as benchmarks targets: v.0.4.10 (released in 2006), v.0.5.6 (released in 2008) and current v.0.6.0 (released in 2010). Their only dependencies are the standard C libraries and they all used the default configuration.

The system under benchmarking comprises 16 wireless nodes: six (6) HP 530 laptops (1.46 GHz Intel Celeron M410 processor, 512 MB of RAM, internal IEEE 802.11b/g Broadcom WM184G wireless card, 4 Li-Ion cells battery (2000 mAh)) running an Ubuntu 7.10 distribution, and ten (10) Linksys routers (200 MHz MIPS processor, 16 MB of RAM, IEEE 802.11b/g Broadcom BCM5352 antenna) running a WRT distribution (White Russian). A similar deployment can be found in [15] to provide low-cost Internet access to an actual residential area.

Finally, taking into account the selected applicative context, the ad hoc routing protocol quality requirements will be defined in terms of *time behaviour* (fast communications), *resource utilisation* (low resource consumption, battery in our case), and *capacity* (transferred information) for *performance efficiency*, *availability* (communication should be possible most of the time) and *recoverability* (communication must recover from unexpected events) for *reliability*, and *integrity* (transferred information must be right) for *security*. To determine the requirements for each selected subcharacteristic, we would follow those defined in [16] for a voting application addressed to enable lecturers to get feedback from audience in auditoriums or convention centres. According to this information, an availability over a 90% will be excellent whereas below a 55% would be in the bounds for an acceptable communication. The common limits of a medium quality communication with respect to its time behaviour range from 40ms to 400ms [17]. As far as evaluating the rest of selected subcharacteristics was not a requirement in [17], we will set their quality requirements as follows: acceptable transfer rate between 110Kbps and 180Kbps, packet integrity among 95% and 99%, route integrity between 20% and 50%, a resource utilisation ranging from 11.7 to 15.7, a recoverability among 20s and 40s and route resiliency between 55% and 90% according to the behaviour experimented by network nodes in a previous fault-free experimentation.

### B. Measures selection

The different subcharacteristics that have been previously selected should be quantified by means of a number of different measures. According to the applicative context of this case study, recoverability will be characterised through *packet resilience* and *route restoration*, whereas integrity will be defined in terms of *packet integrity* and *route integrity*. However, the time behaviour, resource utilisation, capacity, and availability, will be quantified by just one measure: *route delay*, *energy consumption*, *route throughput*, and *route availability*, respectively. The purpose, definition, | Quality measure | Route throughput | Route delay | Energy consumption | Route availability | Packet resilience | Route restoration | Packet integrity | Route integrity |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>v.0.4.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Definition</td>
<td>Avg. time required by destination in a given time period</td>
<td>Avg. energy consumed by node to traverse a communication route from the source to the destination node</td>
<td>Avg. % of time that the communication route published between sender and receiver is ready to be used</td>
<td>Avg. % of packets whose content has not been unexpectedly modified</td>
<td>Avg. % of the throughput protocol to reestablish a route fallen</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Measurement form</td>
<td>Rate of packet generation</td>
<td>Spectrate lost</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Focus</td>
<td>External</td>
<td>External</td>
<td>External</td>
<td>External</td>
<td>External</td>
<td>External</td>
<td>External</td>
<td>External</td>
</tr>
<tr>
<td>Unit</td>
<td>Kbps</td>
<td>Milliseconds</td>
<td>Jiffies</td>
<td>Seconds</td>
<td>Seconds</td>
<td>Seconds</td>
<td>Seconds</td>
<td>Seconds</td>
</tr>
<tr>
<td>Interpretation</td>
<td>The higher the better</td>
<td>The lower the better</td>
<td>The lower the better</td>
<td>The lower the better</td>
<td>The lower the better</td>
<td>The lower the better</td>
<td>The lower the better</td>
<td>The lower the better</td>
</tr>
<tr>
<td>Scale</td>
<td>Absolute</td>
<td>Absolute</td>
<td>Percentage</td>
<td>Absolute</td>
<td>Percentage</td>
<td>Absolute</td>
<td>Percentage</td>
<td>Percentage</td>
</tr>
<tr>
<td>Used for</td>
<td>Capacity</td>
<td>Time behaviour</td>
<td>Resource utilisation</td>
<td>Availability</td>
<td>Recoverability</td>
<td>Recoverability</td>
<td>Integrity</td>
<td>Integrity</td>
</tr>
</tbody>
</table>
measurement and interpretation of these measures is detailed in Table II.

C. Definition of criterion functions

In order to simplify the analysis and interpretation of the defined measures, the LSP approach promotes the definition of different criterion functions ($C_i$) that translate the resulting measures into 0-to-100 values (elementary scores). In our case, we have considered two generic continuous criterion functions, one increasing, to compute the elementary score of the higher-the-better measures such as route throughput, route availability, packet integrity and route integrity, and one decreasing, to compute the lower-the-better measures such as route delay and energy consumption (notes as $a_i$ in Figure 5).

These functions are parameterised according to the lower and higher bounds ($X_{min}$ and $X_{max}$) for increasing functions, and $X_{max}$ and $X_{min}$ for decreasing functions, respectively) that delimit the quality threshold for a given measure. These thresholds have been determined according to the protocol quality requirements previously specified. Table III summarises these thresholds and relates each measure with its criterion function.

The criterion of evaluating the global protocol quality is defined by means of the global score computed by the LSP approach (the higher the better). We have determined that, for this case study, all the measures must meet the thresholds defined and, thus, the weak quasi-conjunction (e.g., $r = 0.261$ when aggregating two elements and $r = 0.192$ when aggregating three elements [12]) is the logical function that best meets this requirement. Finally, all the measures have been considered equally significant and they have been assigned the same weight within the aggregation function.

![Figure 5. Elementary criterion functions used in the case study](image)

However, as far as a low energy consumption is not usually a mandatory requirement in fixed (mesh) networks, we have reduced its relative weight (0.1 out of the total performance efficiency) with respect to route throughput (0.45) and route delay (0.45), within the performance efficiency category. Figure 6 illustrates the proposed recursive aggregation until the single global score $S$ is computed thanks to different aggregation blocks.

D. Workload

The applicable traffic selected to exercise the network was defined in terms of synthetic UDP Constant Bit Rate (CBR) data flows of 200 Kbps, similar to the rates observed in daily scenarios [15].

E. Fault- and attack-load

Communication interruption is one of the most important problems in the domain of wireless mesh networks [18], and it can result from a number of different disturbances. The proposed fault- and attack-load, extracted from our prior investigation [19], consist in a number of accidental faults and malicious attacks whose occurrence may impact the behaviour of the considered routing protocols during the network operation: (i) wireless communications are susceptible to suffer interferences from ambient noise, (ii) being an open medium enables attackers to saturate it with broadcasting storms of packets (flooding) that consume nodes resources and prevent communication, and (iii) intrusion-based attacks may be deployed to either delay the communications (jellyfish), alter the transferred information (tampering), or disable the communication (selective forwarding). Table IV summarises the considered disturbances.

![Figure 6. Complete LSP model applied in our case study](image)

### Table III

<table>
<thead>
<tr>
<th>Measure</th>
<th>Criterion function</th>
<th>$X_{min}$</th>
<th>$X_{max}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Route throughput</td>
<td>Decreasing</td>
<td>110Kbps</td>
<td>180Kbps</td>
</tr>
<tr>
<td>Route delay</td>
<td>Decreasing</td>
<td>40ms</td>
<td>400ms</td>
</tr>
<tr>
<td>Energy consumption</td>
<td>Decreasing</td>
<td>11J</td>
<td>15J</td>
</tr>
<tr>
<td>Route availability</td>
<td>Increasing</td>
<td>55%</td>
<td>90%</td>
</tr>
<tr>
<td>Packet resiliency</td>
<td>Increasing</td>
<td>55%</td>
<td>90%</td>
</tr>
<tr>
<td>Route restoration</td>
<td>Decreasing</td>
<td>20s</td>
<td>40s</td>
</tr>
<tr>
<td>Packet integrity</td>
<td>Increasing</td>
<td>95%</td>
<td>99%</td>
</tr>
<tr>
<td>Route integrity</td>
<td>Increasing</td>
<td>20%</td>
<td>90%</td>
</tr>
</tbody>
</table>

### Table IV

<table>
<thead>
<tr>
<th>Disturbance</th>
<th>Type</th>
<th>Origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ambient noise (A)</td>
<td>Accidental</td>
<td>Natural/Human-made</td>
</tr>
<tr>
<td>Tampering attack (T)</td>
<td>Malicious</td>
<td>Human-made</td>
</tr>
<tr>
<td>Selective forwarding attack (S)</td>
<td>Malicious</td>
<td>Human-made</td>
</tr>
<tr>
<td>Jellyfish attack (J)</td>
<td>Malicious</td>
<td>Human-made</td>
</tr>
<tr>
<td>Flooding attack (F)</td>
<td>Malicious</td>
<td>Human-made</td>
</tr>
</tbody>
</table>
F. Evaluation design

Nodes were deployed in the department according to the topology shown in Figure 7. Different probes were set up in order to monitor the communication establish between nodes A and F and, from this information, quantify the quality of the considered routing protocol. A detailed description of the testbed used to setup all the nodes, execute the workload, inject the selected disturbances, monitor the system, and analyse the information from log files to deduce the measures of Table II, may be found at [20].

This route consisted of three different hops (A–C, C–D, D–F) with exactly the same probability of being targeted by one of the selected faults and attacks. So, in order to attain a minimum of a 0.99 in the quality of the results, and according to Equation 2, the number of experiments to be performed was of 15 per configuration. As we were considering 5 disturbances tests and 1 baseline × 3 routing protocols, we got 18 possible configurations, leading to a total number of 270 experiments. Taking into account that experiments lasted 9 minutes each, the total execution time was 2430 minutes.

G. Evaluation results

Table V shows the average values and the standard deviation obtained from experimentation for the proposed measures depending on the considered disturbance and the target protocol version. Although the analysis of all these numbers could be of interest to perform some detailed study about very particular aspects of the software components, it is quite difficult to estimate their quality thus enabling a fair comparison. Additionally, it is worth noting that measures cannot be independently analysed, since this could lead the evaluator to misleading conclusions, e.g., some disturbances may reduce the energy consumption of the nodes as they hinder their communication capabilities, but this can hardly be considered a benefit for the network as it impacts the final service provided to the user. Accordingly, the more measures we consider, the more difficult to obtain an accurate global vision of the impact of disturbances on the protocol behaviour.

This is then the role that must be played by the proposed LSP approach, aggregating the different measures in order to obtain a global score that could help evaluators in the decision process. Table VI lists the quality scores of the considered protocol versions in presence of representative disturbances observed from both the viewpoint of each characteristic and the global perspective. Such coarse-grained results are delivered from applying the aggregation process already shown in Figure 6 to the fine-grained results obtained in V.

According to these results, the best candidate to be integrated into the final deployment is olsrd v.0.4.10, as it maximises the global score (33.18), whereas the other two versions present similar (lower) results. If we focused on the behaviour of each protocol in presence of a particular disturbance (which is also depicted in Figure 8), olsrd v.0.4.10 is still the best option when dealing with ambient noise, selective forward attacks and flooding attacks, but versions 0.5.6 and 0.6.0 (in no particular order) take the lead when subjected to tampering attacks. Jellyfish attacks seem to impact all the considered protocols in the same way and no one could be taken as the best option to face that particular disturbance. However, evaluators may be interested in just one particular characteristic to evaluate the quality of the protocols and, in that case, olsrd v.0.4.10 obtains the best scores (76 vs. 18 and 40 vs. 28) for performance efficiency and resilience with respect to the other two versions of the protocol, which are best suited for security (16 vs. 8).

Following this analysis, the quality of olsrd v.0.5.6 and v.0.6.0 do not really differ in presence of the considered disturbances, and they could be used indistinctly when tampering attacks or security are the main concerns of the evaluator. Otherwise, olsrd v.0.4.10 is the best candidate to meet the requirements established for this case study.

V. DISCUSSION AND CONCLUSIONS

The development of procedures and guidelines for the certification of the dependability of software components has always been a need for the industry related to safety-critical systems, as unexpected failures may endanger the environment and human lives. A number of standards, like [21], has adopted a Safety Integrity Level (SIL) ranging from 1 (minor property and production protection) to 4 (catastrophic community impact), as a statistical representation of the dependability of safety instrumented system. Accordingly, in order to meet the overall safety requirements, the most
common safety standards such as IEC 61508 for electronic systems [21], DO-178B for airborne systems [22], and EN 50128 for railway systems [23], present very strict requirements on software development and testing. However, once the integration of untrusted third party COTS components is indispensable to meet time-to-market and development costs, the dependability assessment of even non-critical systems is a must [24].

Following this trend, the ISO/IEC SQuaRE [7], which defines procedures for evaluating the quality of software components, has extended its scope to enable the evaluation of the system recoverability [8]. Although being a remarkable effort, this approach strictly focuses on recoverability, thus neglecting the rest of dependability-related attributes, and may only be used for fault-tolerant systems, thus reducing its applicability and usefulness. This paper proposes the adoption of dependability benchmarking notions to enrich the quality evaluation of software products, beyond the mere robustness assessment, within the framework of the ISO/IEC SQuaRE standard. In this way, the proposed methodology could greatly benefit (i) developers, who will be able to detect vulnerabilities in their code and improve it to achieve stated dependability requirements, (ii) acquirers, who may select the most suitable component to be integrated into a system to meet specified performability requirements, and (iii) owners/operators, who could fine tune the configuration parameters of the system to adapt it to changing requirements [25].

The proposed approach integrates dependability benchmarking into the ISO/IEC SQuaRE standard to tackle existing limitations in the quality evaluation of software products in presence of disturbances. First of all, and considering the increasing importance of possible interactions among existing components, not only accidental faults (as proposed by the ISO/IEC 25045 module for recoverability), but also malicious ones (attacks) should be injected on the system to evaluate their impact on the quality characteristics of the target component. Moreover, as different disturbances may affect the component behaviour in a different way, this work emphasises the pertinence of selecting a set of faults and attacks that is representative of the applicative context the component will be deployed on. Likewise, apart from considering recoverability-related measures, it is of prime importance to obtain, on the one hand, reliability- and security-related measures, and on the other hand, performance efficiency-related measures in absence but also in presence of disturbances to determine their effect on the behaviour of the system. Our proposal also provides means to determine the required number of experiments to ensure that results will attain a given quality level. Finally, as far as the number of considered measures increases, the analysis of results following traditional techniques becomes more complex. Analytical and graphical benchmarking-based techniques for the aggregation and interpretation of results, such as the Logic Score of Preferences and Kiviav diagrams, constitute a very useful support to overcome the problem of scalability as they provide a more concise vision of the system, as shown in the case study.

The resulting benchmarking approach enriches the ISO/IEC SQuaRE standard and enlarges its application domain for evaluating the quality of software COTS components from a performance and a dependability point of view even for non-critical applications. Nevertheless, as seen in the presented case study, there exist several aspects that

<table>
<thead>
<tr>
<th>Target</th>
<th>Disturbance</th>
<th>Performance efficiency</th>
<th>Reliability</th>
<th>Security</th>
<th>Global quality</th>
<th>Global quality per disturbance</th>
<th>Global quality per disturbance</th>
</tr>
</thead>
<tbody>
<tr>
<td>v.0.10</td>
<td>A</td>
<td>73.81</td>
<td>61.78</td>
<td>7.18</td>
<td>35.39</td>
<td>33.18</td>
<td>33.18</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>100.00</td>
<td>98.63</td>
<td>0.36</td>
<td>27.27</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>40.05</td>
<td>67.99</td>
<td>34.20</td>
<td>48.86</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>J</td>
<td>4.81</td>
<td>95.65</td>
<td>72.64</td>
<td>28.44</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>42.63</td>
<td>65.30</td>
<td>7.18</td>
<td>29.56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>v.0.6</td>
<td>A</td>
<td>33.03</td>
<td>7.94</td>
<td>7.15</td>
<td>18.02</td>
<td>18.02</td>
<td>18.02</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>100.00</td>
<td>66.64</td>
<td>2.47</td>
<td>32.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>4.65</td>
<td>5.73</td>
<td>65.95</td>
<td>13.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>J</td>
<td>4.81</td>
<td>59.83</td>
<td>67.80</td>
<td>30.78</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>54.38</td>
<td>7.67</td>
<td>7.15</td>
<td>15.72</td>
<td></td>
<td></td>
</tr>
<tr>
<td>v.0.60</td>
<td>A</td>
<td>73.05</td>
<td>7.82</td>
<td>7.15</td>
<td>17.81</td>
<td>17.81</td>
<td>17.81</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>97.92</td>
<td>64.68</td>
<td>2.92</td>
<td>32.76</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>4.66</td>
<td>60.07</td>
<td>66.41</td>
<td>14.24</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>J</td>
<td>4.81</td>
<td>62.32</td>
<td>62.33</td>
<td>30.89</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>60.42</td>
<td>7.41</td>
<td>7.15</td>
<td>16.28</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
are dependent on the applicative context where the COTS component is to be deployed, or the quality thresholds ($X_{\min}$ and $X_{\max}$), weights ($w_i$), and operator types ($o_i$) in charge of measures aggregation. In this sense, our future work ambitions to characterise different applicative contexts, providing templates with pre-computed parameters that benchmark users may customise for their particular needs, thus easing the application of this methodology for the quantitative evaluation, comparison, and selection of software COTS components.

ACKNOWLEDGEMENTS

Work sponsored by the Spanish project TIN2009-13825 and Brazilian FAEPEX and CAPES proc. n. BEX3587-10-0.

REFERENCES


