CMU-PT/RNQ/0015/2009

Trustworthy and Resilient Operations in a Network Environment

Deliverable D7

First Specification of the Recovery Mechanisms

Information and Communication Technologies Institute
Carnegie Mellon | PORTUGAL
AN INTERNATIONAL PARTNERSHIP

Executed by: **FCTUC**

Direction/Department: **DI**

Date: **19-09-2012**

Version Number	Owner	Change Control	Date
0.1	FCTUC	Initial draft	19-04-2012
0.6	FCUL	Extensive revision	5-07-2012
0.7	FCTUC	Complete Document	18-09-2012

Additional information:

Author/s: Bruno Sousa, Marilia Curado ...	
Contributor/s: António Casimiro, Diego Kreutz ...	
Beneficiaries contributing on the deliverable: FCTUC, FCUL	
WP contributing to the deliverable:	WP2
Estimation of pm spent on the deliverable:	X
Nature of the deliverable:	Report
Total number of pages:	34

— Document generated on 19 September 2012 at 10:13 —

Contents

List of Figures

List of Tables

Executive Summary

This document is the first deliverable of Task 2.2: Automated Reconfiguration and Adaptation. The main objective of this task is to investigate proactive network management and error-recovery techniques to support fault-tolerant network operation. This is achieved through monitoring of the control and data planes, for fast detection of errors and intrusions and for seamless recovery. In this context, this deliverable describes the initial work that was done in the task, providing the first specification of the recovery mechanisms proposed in the TRONE project.

The components of the TRONE project conceived for automated reconfiguration and adaptation resort to, and are tightly related to the monitoring and diagnosis mechanisms (mainly developed in Task 2.1) and to the intrusion-tolerant components defined in WP3. In this document we first introduce to the problems considered in the work and then we provide an overview of related work regarding approaches for Byzantine fault and intrusion tolerance, as well as solutions for automated recovery. The document then describes the work performed in three main directions. First, we focus on multi-homing-based reconfiguration and recovery techniques, which have been specified as key tools for an improved fault-tolerant system. Then we describe the recovery mechanisms that are considered within the Fault and Intrusion Tolerant (FIT) event broker, a central component of the TRONE architecture for the trustworthy dissemination of events. Finally, we also describe our work on performance improvements through the use of adaptive protocols, namely timeout-based adaptive protocols. We provide results that show the positive impact of dynamically estimating and adjusting timeout values (e.g., used in failure detectors, or in consensus protocols such as the one needed in the FIT event broker), in contrast with using statically configured timeout values.

Chapter 1

Introduction

During the last years, recovery mechanisms have been the subject of intensive research activity. However, in spite of the wide variety of mechanisms proposed, there remains a lack of evidence regarding the behaviour of various recovery mechanisms and their influence on the Information Technology infrastructure. This issue is of particular relevant in cloud-based systems, as addressed in the TRONE project. The first specification of the TRONE recovery mechanisms is presented in this document. This chapter presents the motivation for the work performed and the problem statement driving the automated recovery components developed in TRONE.

1.1 Motivation

The Telecom industry has been evolving from simple voice operators towards multiple service providers, supporting on-demand provisioning, in a seamless and technology independent manner. This heterogeneous scenario creates new demands on the infrastructure, increasing the likelihood of failures, either accidental or malicious. At the same time, users are becoming more demanding in terms of the quality of the service they get, creating new challenges to the operators, which must meet the users expectations. The prevention and reaction of failures is a key issue that operators must address in an effective way.

Portugal Telecom (PT) has been implementing cloud-based solutions, including a complete virtualization infrastructure, which is being used to offer several major services to enterprise customers as well as internally. In this context, and given the growing dependence of PT on cloud-based technologies, robustness of this kind of infrastructures is paramount to the company. It is therefore of major importance to develop mechanisms that dynamically

reacts to failures, allowing a fast recovery and seamless support of the services provided. To achieve this goal there is the need to address recovery at different levels, including the fault and intrusion tolerant (FIT) event broker, responsible for the trustworthy dissemination of events and the communication, and the resilient data transfer, at the transport level.

1.2 Problem Statement

When offering Infrastructure as a Service cloud services, the network should perform well despite overloads resulting from failures or attacks affecting some links/entities. Fast reconfiguration when some problems are detected is hence an important asset. In this context, the main goal of the work described in this deliverable is the specification of the TRONE recovery mechanisms for the cloud infrastructure.

Automated reconfiguration and adaptation in TRONE is focused on the recovery mechanisms of the fault and intrusion tolerant event broker, on the reconfiguration and recovery mechanisms of the SCTP based multi-homing approach, and finally, on the dynamic adaptation of timeouts, reflecting the variable state of the system. The FIT event broker in TRONE encompasses recovery procedures and state management, which can be controlled by the replicas and by the applications, as for instance, a key-value storage application. The TRONE multi-homing approach aims to achieve improved resilience to faults affecting the communication links used by client applications, which can make use of security-related diagnosis information to trigger reconfigurations. The TRONE approach for dynamic adaptation of timeouts relies on the achievement of adaptive consensus, through the introduction of new functionalities to static timeout-based protocols.

This document is structured as follows. Chapter 2 identifies related work on reconfiguration and adaptation mechanisms, according to three main perspectives: fault and intrusion tolerant solutions, automated reconfiguration and recovery, and dynamic timeout adaptation. The way each of these issues is addressed in TRONE is detailed in the following chapters. Chapter 3 describes the multi-homing reconfiguration and recovery mechanisms, Chapter 4 specifies the recovery mechanisms of the fault intrusion tolerant event broker, and Chapter 5 details the approach for dynamic adaptation of timeouts and assesses its performance. The main conclusions and issues to be addressed in the next phase of the project are presented in Chapter 6.

Chapter 2

Related Work

This chapter identifies related work on mechanisms for automated reconfiguration and adaptation, comprising different recovery approaches. Section 2.1 presents fault and intrusion tolerant solutions, and section 2.2 addresses technological independent approaches at the transport layer for automated recovery. The orthogonal issue of dynamic timeout adaptation is presented in section 2.3.

2.1 Byzantine Fault and Intrusion Tolerance

Byzantine fault tolerant protocols are essential for any system which requires to be reliable even when an arbitrary behavior occurs, such as those caused by bugs, unpredicted environmental events and attacks. There are different Byzantine fault tolerant (BFT) algorithms and techniques [3, 8, 21, 30, 37]. In spite of that, there is a path to be fulfilled from Byzantine fault tolerance to intrusion tolerance [3]. Intrusion tolerant systems should be designed to assure security properties, such as confidentiality, integrity and availability, even if parts of the systems is under control of an adversary [14, 7]. Thus, fault and intrusion tolerant (FIT) solutions take the systems one step further when compared to pure BFT. A FIT system is able to tolerate faults and intrusion, making the system reliable and resistant despite problems such as furtive attacks and compromised replicas.

BFT itself is not enough to provide intrusion tolerance. Intrusion tolerant techniques, components and architectures have been proposed and evaluated through the past decades [34, 35]. Some of the most explored techniques include proactive recovery, proactive-reactive recovery, diversity and trusted components [4, 33, 28, 22, 16, 15, 20].

BFT protocols are one of the basic building blocks of FIT systems. Those protocols provide the means to design and deploy replicated services, where each replica has a well defined rule in the system. An attacker, for instance, will target the system replicas, intending to compromise the service. One way to improve the resilience of the system is through proactive and reactive recovery techniques. The idea is to reduce the window of vulnerability and, consequently, extend the FIT system life time. Well designed and implemented recovery techniques will make the attackers tasks harder. Even if one replica is compromised, as soon as the recovery procedure takes place, the attacker or bug is going to fade away since the replica will have its state recovered, starting a new and fresh execution.

Diversity is another important trick that can be applied in different levels of the replicated system. There are recent works showing that OS and database diversity are practical examples of how different software types and versions can help to improve the system resilience against common problems and attacks [15, 16].

In FIT systems, diversity and recovery techniques are important to strive the system life by avoiding or reducing the impact of intrusions. Along with other security mechanisms, such as secret sharing, a FIT system is capable of surviving under continuous attacks, for instance.

2.2 Automated Recovery

The Stream Control Transport Protocol (SCTP) [27] is a protocol that enables transport of data in a stream oriented way. It is opposed to the Transport Control Protocol (TCP), as it is byte-oriented. Furthermore, SCTP supports multipaths. In the association establishment phase SCTP nodes are able to exchange information regarding the configured IP addresses they have. Therefore, in the failure of a path (pair of addresses), there is another one that can be used. The operation of SCTP can also be configured by applications via a dedicated Application Programming Interface (API) [26]. Configurations, such as number of retransmissions, are possible via this API.

SCTP performs end-node and path failure detection. For each end-node SCTP keeps information of each retransmission performed. If the number of retransmission exceeds the *Association.Max.Retrans*, then the end-node is considered as unreachable and, consequently, all the data transmissions are stopped. Path failure detection mechanisms include heartbeat messages, timeouts and number of retransmissions. Heartbeat are signalling messages that are exchanged between SCTP nodes to monitor reachability of paths (to determine that a certain pair of addresses can be used). Heartbeat messages are expected to be acknowledged,

and if no acknowledgement is performed within a certain period with a count higher then the *Path.Max.Retrans* parameter, then the path is marked as inactive. Upon the inactivation of paths, SCTP automatically transmits packets on paths previously selected as backup. The interval on which heartbeat messages are exchanged and the maximum limit to mark a path as inactive (*Path.Max.Retrans*) are configurable.

SCTP includes multiple parameters that affect the recovery and protection mechanisms of SCTP. Eklund et al. [12] have determined the best configuration of SCTP recovery parameters to enable an efficient transport of signalling messages. Moreover, SCTP, due to the intrinsic recovery mechanisms, has been evaluated in the Reliable Server Pool architecture to allow server redundancy and session failover [11]. Despite the enhanced configuration of SCTP for recovery, achieved by Eklund el al. [12] and Dreibholz et al. [11], it has been obtained in a static approach. That is, parameters were configured and performance of SCTP was assessed. Dynamic configuration of SCTP recovery mechanisms should be pursued, as they enable SCTP to adapt to conditions of network. These dynamic configuration should go beyond reachability configuration and should include network conditions, such as an increase in the delay of a certain path, or an increase of packet loss.

2.3 Timeout adaptation

To reason about performance, a crucial issue is the characterization of the temporal behavior of the network on which protocols are executed. This is particularly important when considering uncertain communication environments, in which network delays are strongly exposed to the effects of contention [19]. If a correct timeout can be selected according to the observed network conditions, and if the protocol in which the timeout is used can be dynamically adapted, then performance can be improved.

The problem of dynamically selecting timeouts has been studied in several different contexts. For example, this is a critical problem for real-time multimedia systems and applications, because in those systems the delay in detecting errors caused by conservative large timeouts would compromise the quality of service perceived by end users. There are many works that propose solutions for timeout selection and adaptation in this context, which rely on the ability to monitor several operational parameters (e.g., [1]).

The research area of networks and distributed systems is also concerned with adaptation and timeout selection, especially with the growth of complex dynamic systems, based on wireless networks, largely distributed communication environments and mobile applications,

in which making static timing assumptions may compromise the system performance, or even the safety of protocols and algorithms.

In the networks field, a well-known approach for timeout selection is the retransmission timeout estimation used by the TCP protocol. It implements a very simple algorithm introduced in [18], in which retransmission timeouts are computed based on the observed round-trip times and their variations. However, some researchers have shown that the TCP congestion control mechanism is not appropriate for networks experiencing highly variable delays, like wireless networks or networks under attack, given that these delays are considered as indicators of congestion, causing a reduction in the transmission rate and compromising the network throughput. In general, the proposed solutions for this problem, in particular applied in the context of wireless networks, are based on extensions to the link layer protocols [25, 38].

Timeout-based adaptive solutions have also been proposed in the context of failure detection, with the objective of improving the tradeoff between detection time and accuracy of failure detectors. Typically, timeouts are derived from some statistics applied to a number of observed heartbeats delays (e.g. average delay), and on different mechanisms to compute safety margins [2, 24]. Other solutions focus on adapting parameters for failure detection according to specified QoS requirements. In particular, the failure detector proposed in [6] follows an approach based on the feedback control theory to compute the interrogation period, adapting both timeout and period in runtime. In [9] we proposed an adaptive failure detector, which uses the *Adaptare* framework [10] as a timeout provisioning service, and compare its performance with other adaptive failure detectors, including the QoS-driven approach presented in [5].

In general, the works mentioned above handle the problem of timeout selection in specific application contexts, without separating the monitoring and adaptation aspects from the application semantics. Such tailored approaches can deliver optimized results, but make it difficult to reuse the monitoring and adaptation mechanisms in other contexts. In contrast with the typical approaches, the approach we propose to adopt in TRONE has two fundamental advantages. First, we propose to use a framework for monitoring and adaptation support that may be used as a timeout provisioning service and, in that sense, is fully independent of the specific application context. This makes it easier to develop dynamically adaptive protocols, simply transformed from static ones. Second, the framework is driven by a dependability requirement expressed though a coverage value. This is the crucial parameter to be set, and defines the minimum expected probability that the computed timeout will be large enough for the application purposes. The coverage value does not depend on the ex-

ecution environment, thus no fine tuning process is required to reuse adaptive solutions that follow our approach in different environments: they automatically adapt to new conditions and networks, computing the necessary timeouts to secure the expected coverage.

Chapter 3

Multi-homing based Reconfiguration and Recovery

3.1 Goals

The goals with multi-homing based reconfiguration and recovery include the improvement of SCTP performance regarding detection of failures and optimized reconfiguration. SCTP is improved by considering multiple criteria in the path selection process.

3.2 Algorithms/Mechanisms

The choice of a transport protocol is normally based on the application requirements. For instance, if reliable delivery is required than TCP is a natural choice. Nonetheless, TCP has not advanced characteristics like multi-homing support, features that SCTP provides natively. SCTP includes recovery mechanisms and provides a primary-backup protection model. On the failure of a path (chosen as primary) another one is employed (as backup) in a transparent way, without the user or application intervention. In addition SCTP, through the Concurrent Multipath Transfer (CMT) extension [17] enables the concurrent protection model, on which paths can be used simultaneously, for instance, to increase throughput. In the cloud perspective, services are protected at the transport layer when employing SCTP. TCP has no such mechanism natively incorporated, despite the recent multipath extension [13].

TRONE employs standard mechanisms, avoiding new protocols or amendments to existent ones, to enhance SCTP recovery. In a first instance, TRONE uses the SCTP API [26] to

configure SCTP via standard primitives *SET_PRIMARY_ADDR*. The decision on which path to use, relies on multiple criteria instead of the single reachability metrics of SCTP. This way protection and recovery is improved as these criteria include availability, recovery impact metrics. The multiple criteria include two major groups, multi-homing related metrics and Traffic Performance (TP) metrics. Multi-homing related include multi-homing goals, namely, resilience, ubiquity and Load Balancing [31]. All the criteria are organized in benefits (items that must be maximized) and costs (items that must be minimized), as depicted in Figure 3.1 and Figure 3.2, respectively.

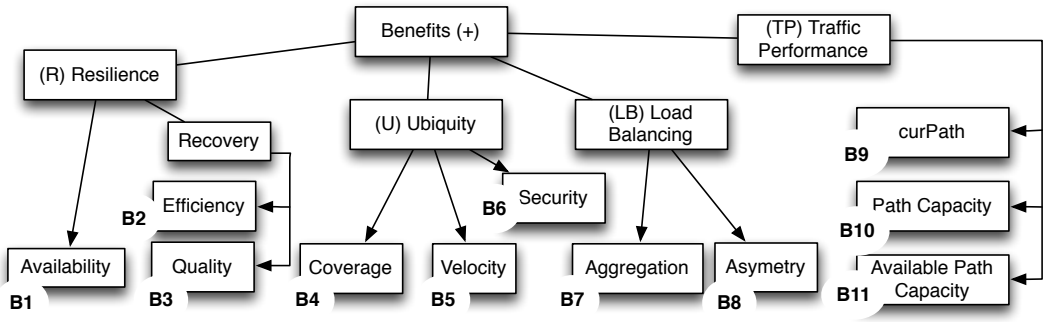


Figure 3.1: Benefits Criteria for Enhanced SCTP

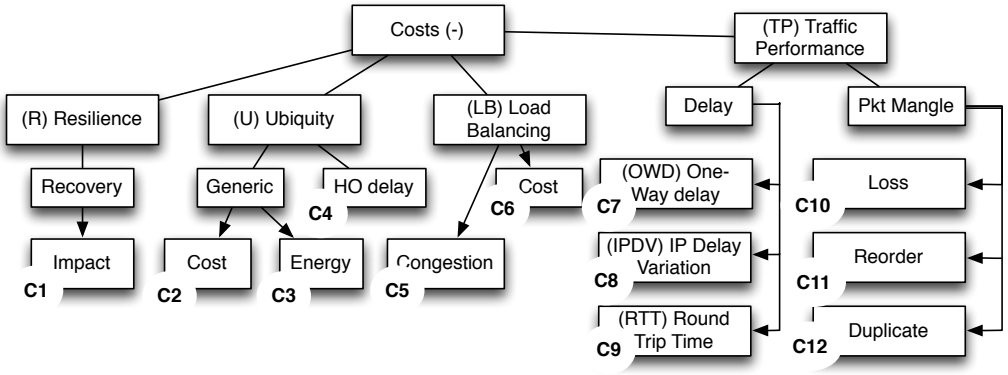


Figure 3.2: Costs Criteria for Enhanced SCTP

Benefits criteria are derived from [32], while Traffic Performance (TP) metrics include available path capacity and path capacity, as per *RFC 5136*. *curPath* may be 0 or 1 and is used to establish preference of a path that has been previously selected (*current path*) among the remaining. Costs criteria include the impact on applications that recovery mechanisms have, as well as characteristics that are associated with a technology, such as monetary costs, energy consumption and time to connect to the network (HODelay). Load Balancing (LB)

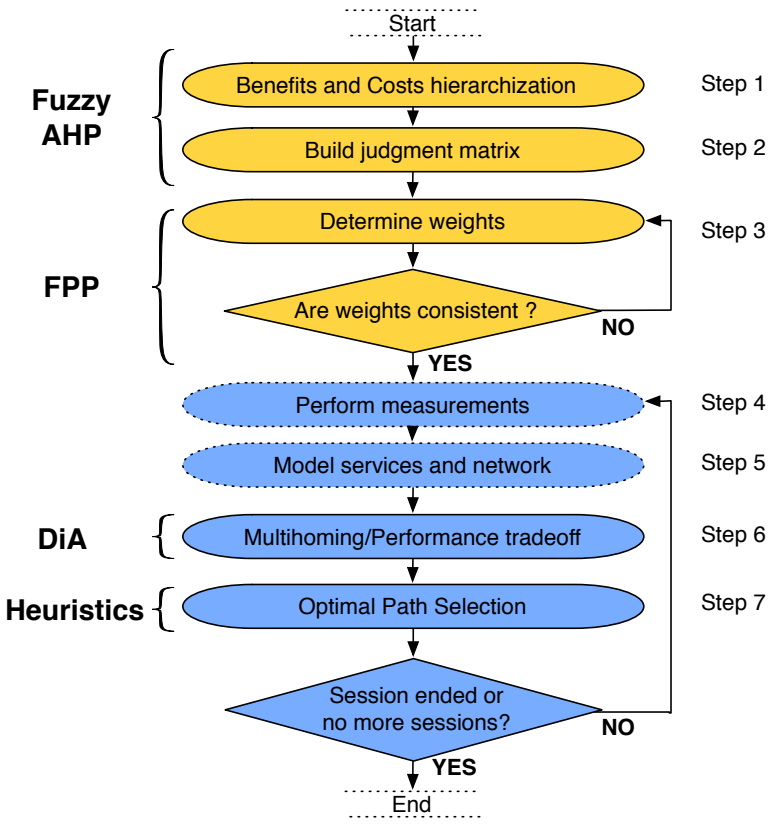


Figure 3.3: Multi-homing based reconfiguration algorithm

costs include overhead in terms of the cost of diverting packets over different paths (e.g. processing time) and the congestion that can occur by injecting more packets. TP criteria are split into two categories: Delay that includes criteria to assess the cost in terms of time and the Packet Mangle that includes criteria assessing the impact on applications in terms of packet loss, reordering or duplication. The delay criteria includes One-way delay (OWD) – *RFC 2679*, IP delay variation (IPDV) – *RFC 3393* and Round-Trip Time (RTT) – *RFC 2681*, while the Packet Mangle criteria considers Packet Loss – *RFC 2680*, Packet Loss pattern (loss distance) – *RFC 3357*, packet reordering – *RFC 4737* and packet duplication – *RFC 5560*. All these metrics are based on well-knows specifications done by the Internet Engineering Task Force (IETF) IP Performance Metrics (IPPM) working group.

Employing a Multiple Attribute Decision Mechanism (MADM) each path (pair of addresses) is scored. By applying heuristics aiming different goals (e.g., resilience, traffic performance) the optimal path is selected to be the primary path based on the MADM score. The configuration of the primary path is performed via the the SCTP API using the *SCTP_PRIMARY_ADDR* and *SCTP_SET_PEER_PRIMARY_ADDR* options.

As illustrated in Fig. 3.3, multi-homing reconfiguration is performed in several steps using diverse optimization techniques. The first one, is based on Analytic Hierarchy Process (AHP) to determine the value for the weights of the several multi-homing criteria. The Fuzzy Preference Programming method is employed to assess the consistency of weights. That is, in order to guarantee that preferences of a given criteria over another is guaranteed.

The determination of weights is only performed once, on a pre-loading of applications, for instance, data applications setting weights for the different criteria (e.g., available path capacity might be the most important criteria).

After setting all the weights, values for the diverse criteria can be collected during session lifetime. Such data collecting can be based on time-basis or event-basis (e.g., when an interface goes down). When the values of the different are collected, the optimal path can be determined by employing Distance to Ideal Algorithm (DiA) technique. By employing this MADM technique a matrix with scores of the different paths is obtained. By employing different heuristics, paths can simply be selected based on this score, or using threshold information, for instance, if the delay has exceed certain levels (e.g., $\geq 150ms$) or packet loss is higher than a certain limit (e.g.,$\geq 10\%$).

Chapter 4

Recovery in the FIT event broker

In this section we describe the recovery mechanisms provided for the FIT event broker, which was introduced in deliverable D10, through the BFT-SMaRt library. This library provides state machine replication protocols in order to keep a consistent view of all critical information inside the FIT event broker replicas, such as available channels and lists of clients registered or subscribed to each channel.

Figure 4.1 illustrates the data structure that needs to be consistent among all replicas of the FIT event broker. The channels (TAGs) can be created at any moment during the system life time. Old and new publishers and subscribers may be interested in one or more channels. Publishers execute a register procedure in order to request permission to publish events in the corresponding channels of interest. Similarly, subscribers need to call a subscribe procedure to get access to the events published in the respective channels of interest. This information (identified in the figure by the orange part) about the channels and clients has to be consistent among all replicas. Consequently, each replica recovery process must guarantee that the critical data will be correctly available on all recovered replicas.

BFT protocols work under the assumption that there are at least $n - f$ correct replicas in the system. The value of n will vary depending on the system model and the properties that need to be assured. In a typical BFT system, $3f + 1$ replicas are used to support up to f faults. However, $2f + 1$ replicas can provide the same safety and liveness when using hybrid models, in which there are trustworthy components that can be used (for instance, to authenticate messages) to support the protocol execution. In the former case we require $2f + 1$ **correct** replicas, while with hybrid models only $f + 1$ correct replicas are sufficient to guarantee the system safety and liveness properties.

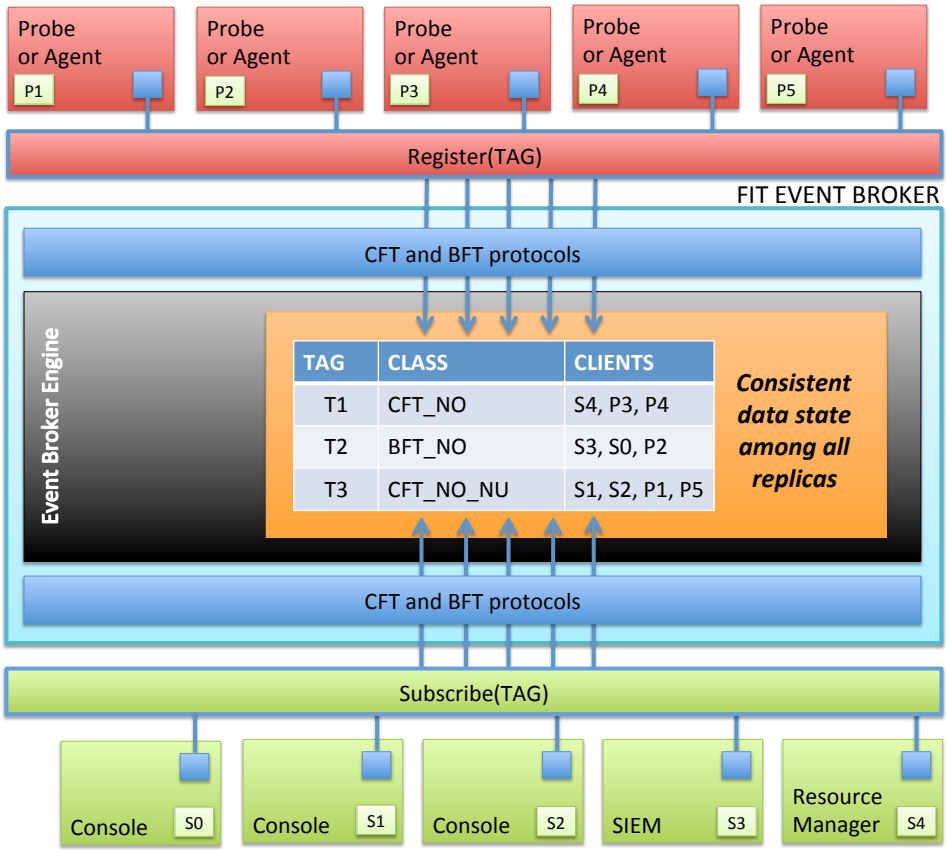


Figure 4.1: FIT event broker channel and client data consistency

In what follows we describe the fundamental steps for replica recovery within the BFT-SMaRt library, through replica state transfer executions. As mentioned, for generic active replication, BFT-SMaRt requires $3f + 1$ replicas to assure the safety and liveness properties with up to f faults or malicious replicas. The system state, used in the recovery procedures, is divided into application and replica states. Once a recovery procedure is started, both application and replica state may be required to update a replica.

There are two major cases where the replica and application data state will be required: (a) a replica detects that it is outdated; and (b) a new replica enters the systems, either to scale the number of replicas or to replace a faulty replica.

In the first case, the replica detects and triggers itself a state transfer execution. This might be the case when a replica is slower than others or has been unreachable for some time. The second scenario will happen when the system has to be scaled (when the number of replicas needs to be increased) or when an outside controller detects and replaces a faulty replica. In both cases, a replica recovery procedure is required to start a state transfer execution among the replicas. All $n - f$ (or more) correct replicas can be used to execute a state transfer in

order to synchronize the state of an outdated replicas.

State transfer executions are started by outdated or new replicas that just joined the system view. The most common cases that will trigger a state transfer are:

- a new replica is added to the system;
- a replica enters late in the system;
- a replica is slower than others and needs to update its own state;
- a replica was isolated or without network for some time;
- a replica is outdated for some unknown reason.

In all these cases, a replica identifies that it is late with respect to the other replicas by comparing its execution IDs (system views) with those (future) received from other replicas. As soon as a replica detects that it is outdates, it will request a state transfer execution, sending to all other replicas the last received execution ID, and requesting a confirmation. Once confirmed, it requests a state transfer until this execution ID.

The state transfer may include the last checkpoint and all operations executed after it or, alternatively, just a hash for verification purposes. In the latter case, only one replica will send the last checkpoint data and the last executed operations. The hash received from the remaining replicas is used to verify if the transferred state is correct. If not, the replica restarts the state transfer and requests a copy of the state from each replica. After confirming $f + 1$ matching states, the replica applies the checkpoint and all pending operations to its local state, finishing the state transfer execution.

Finally, BFT-SMaRt allows the state management to be delegated to the application. This is interesting in the sense that it provides flexibility and extensibility. Different applications will be able to manage state recovery in different ways by defining specific checkpointing and logging profiles. As an example, a simple key-value storage application may define less periodic checkpoints because the operations can be executed with low latency and resource consumption. However, heavy load database applications may require more frequent checkpoints in order to reduce the number of operations and amount of data that may be required in a state transfer executions. Those operations include everything that has been executed after the last checkpoint.

Chapter 5

Timeout-based adaptive consensus

In this section we describe an approach to support the dynamic adaptation of timeouts and achieve increased protocol performance. The approach is based on a probabilistic framework called *Adaptare*, and is applicable to any timeout-based protocol. However, we illustrate the specific benefits of the approach by considering the case of consensus, which is a fundamental problem in distributed systems. Dynamic adaptation of timeouts can be particularly useful in TRONE to achieve improved performance of the FIT event broker. In fact, as mentioned in the previous section, the FIT event broker uses the BFT-SMaRt library, which implements a consensus protocol and relies on timeouts for detecting faulty replicas, among other problems. The ability to dynamically select appropriate timeouts is therefore relevant for optimizing the time to react and recover from faults, while avoiding unnecessary reconfigurations by reducing the probability that false positives.

We first provide a brief introduction of Adaptare, since this is relevant to understand how the approach works and how it is applied to achieve adaptive consensus. Then we describe the solution for transforming a static consensus protocol into a dynamic one. The section is concluded with the presentation of experimental results, which compare the performance of the static and adaptive versions of this protocol, illustrating the potential improvements that may be achieved.

5.1 Adaptare: timeout provisioning service

We propose the use of *Adaptare* to compute timeouts during the protocol execution. *Adaptare* is a framework developed to support adaptive systems in stochastic environments, driving

the adaptation process according to dependability requirements specified by the client application. For self-containment, and since it is important to understand the interface provided by *Adaptare* that is used by the adaptive consensus protocol, we provide a short description of *Adaptare* in what follows. The interested reader is referred to [10], which provides a complete description of *Adaptare*, including algorithms, mathematical equations, implementation details, complexity analysis and a comprehensive performance evaluation.

Figure 5.1 illustrates *Adaptare*'s operation. Three parameters must be provided at the framework interface: (i) recent samples of the monitored temporal variable; (ii) the history size h, which defines the number of samples that will be analyzed by the framework; and (iii) the required coverage for the computed bound, which is the probability that this bound will hold given the network conditions. Then, *Adaptare* performs a probabilistic analysis of the samples in the history, in order to determine an upper bound on the observed variable, according to the required coverage.

Probabilistic characterization. The framework is based on the assumption that the system (represented by the monitored variable) behaves stochastically, alternating periods during which the environment conditions remain fixed (*stable phases*), with periods during which those conditions change (*transient phases*). During stable phases, the statistical process that generates the data flow may be characterized, and hence we can compute the corresponding distribution using an appropriate number of samples. On the other hand, if the environment conditions are changing, then the associated statistical process is actually varying and no fixed distribution can describe its real behavior.

Adaptare employs two goodness-of-fit (GoF) tests for the analysis of input samples in order to determine whether the system is in a stable period and, if this is true, which probability distribution better describes the current state: the Kolmogorov-Smirnov (KS) test and the Anderson-Darling (AD) test [21]. These are formal statistical procedures used to assess the underlying distribution of a data set. Five distributions are verified: exponential, shifted exponential, Pareto, Weibull, and uniform. The parameters for those distributions are estimated from the samples, using the maximum likelihood estimation (MLE) and the linear regression methods. A stable period with distribution F and parameters $p_1, p_2, ..., p_n$ is detected when some GoF test establishes the goodness of fit between the postulated distribution and the evidence contained in the experimental observations. Otherwise, it is assumed that the environment is changing, and a transient phase is detected.

Bound computation. Once the environment characterization is completed, *Adaptare* is able to compute a new dependable upper bound for the observed variable. For stable phases, this bound is derived from the cumulative distribution function (CDF) of the identified

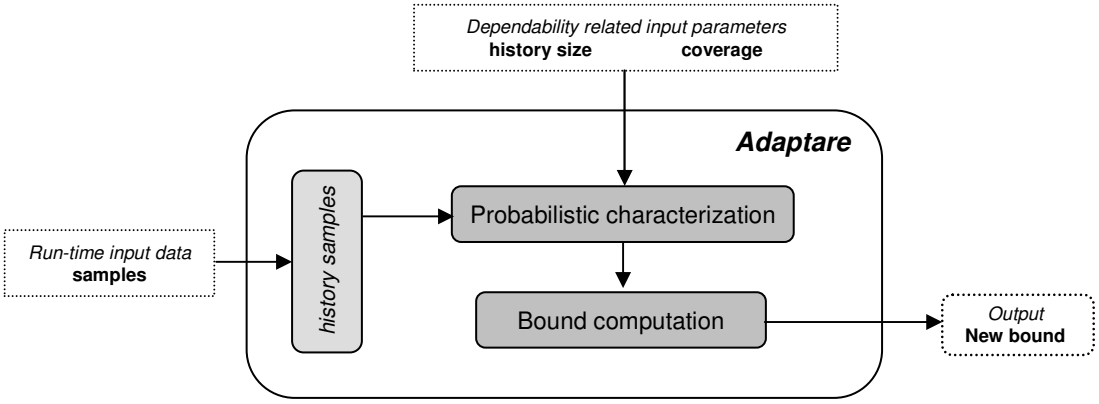


Figure 5.1: *Adaptare* framework

distribution. Formally, the CDF represents the probability that the random variable X takes on a value less than or equal to t ($F_X(t) = P(X \leq t)$). This definition allows to directly use the specified coverage parameter, which is, in fact, the probability that the variable value will be lower than the computed timeout. Thus, t is the new timeout (output bound), obtained by directly applying the required coverage and the estimated parameters into the distribution's CDF.

Whenever it is not possible to detect a distribution from the input samples (transient periods), *Adaptare* computes a conservative bound, based on the one-sided inequality of probability theory, which holds for all distributions.

In summary, if the environment does not change arbitrarily fast (in which case a probabilistic characterization would become invalid too quickly), then *Adaptare* is able to either determine a probability distribution that describes the observed pattern and select a new timeout accordingly (during stable phases), or compute a conservative timeout that holds for any probabilistic distribution (during transient phases).

One important practical property of the framework is the separation between monitoring and characterization. The client application may feed *Adaptare* with new observations of the variable of interest at any time during the system execution. The h (history size) most recent values constitute *Adaptare*'s internal history, with h provided at the framework interface. The characterization process is triggered only when a new timeout is requested: at this point, *Adaptare* runs the probabilistic mechanisms over the history of samples and returns the new timeout to the application.

5.2 Achieving adaptive consensus

We start by describing the consensus protocol that we considered in this study, which is a consensus protocol introduced in [23] to solve the k-consensus problem in wireless ad hoc networks. The reason why we considered this protocol was twofold. First, we had access to an implementation, based on static timeouts, which we could use in our work. Second, this protocol was developed for wireless environments, being able to natively deal with uncertainties, variable network latencies and network failures. Therefore, this allows to better infer the potential improvements that might be achieved only by the use of dynamic timeouts in the presence of environment uncertainties, in contrast with the baseline performance of a protocol supposedly designed for such environments.

5.2.1 Static consensus protocol

In the k-consensus problem, each process p_i proposes an initial binary value $v_i \in \{0, 1\}$, and at least $k > \frac{n}{2}$ of them have to agree on a common proposed value, where n is the number of participant processes. The remaining processes are not required to decide, but if they decide, it must be on the same value decided by the k processes.

The work assumes a communication failure model [29], which is more appropriate to represent wireless ad hoc networks, but an impossibility result states that under this model there is no deterministic algorithm that allows n processes to reach k-agreement, if more than $n-2$ transmission failures occur in a communication step [29]. The protocol introduced in [23] circumvents this impossibility by employing randomization to tolerate omission transmission faults.

The protocol executes in rounds. In a round, each process p_i broadcasts a message containing its identifier, proposal value and other variables comprising its internal state. Then it waits for messages broadcast by the other processes. When messages from the majority of the processes are received, process p_i will make progress by analyzing them, updating its state and possibly deciding on some value or initiating a new round. However, as assumed in the communication failure model, some messages that a process is supposed to receive may be lost. This may delay p_i (since it will need to wait for slower messages) or may even prevent progress to be done (if too many messages are lost). Therefore, in each round, process p_i waits for messages only during a pre-defined timeout. If not enough messages are received within this timeout, its state does not change and p_i starts a new round by retransmitting the original message. This protocol ensures safety regardless of the number of omission faults

in each round, while liveness is guaranteed in rounds where the number of omission faults is $f \leq \lceil \frac{n}{2} \rceil (n-k) + k - 2$.

Given that this protocol relies on a timeout to decide whether messages should be retransmitted and when that should be done, we now explain how to transform it into an adaptive protocol that dynamically adjusts the timeouts in response to variations on the network delays.

The architecture of our solution is very simple: each consensus process has its own instance of the timeout provisioning service (*Adaptare*), so that timeout selection decisions are fully decentralized. Furthermore, there is a logical separation between the timeout-based consensus protocol and timeout calculation functions, which makes it easier to apply the approach without the need of significant changes on protocol code, as detailed ahead. We believe that this also makes the approach more generic and suitable to be applied when build adaptive versions of other timeout-based protocols.

5.2.2 Protocol instrumentation

Typically, timeout-based protocols rely on a fixed timeout value that defines an upper bound on the time that a process should wait for some event. In order to make timeouts adaptive using *Adaptare*, it is necessary to: (i) identify the system variable that should be bounded by the timeout; (ii) instrument the protocol in order to collect samples of this variable that will be fed into *Adaptare*, and (iii) search for places in the protocol where the timeout is used, adding a call to *Adaptare* in order to update the timeout.

In the considered consensus protocol, the timeout defines how long a process will wait for the reception of messages from the majority of the processes, after initiating a round. Therefore, the variable to be monitored is the time elapsed between the beginning of a round and the reception of each message for that round. Even those messages that arrive after the round terminates should be considered, otherwise we would have information only about timely messages, compromising *Adaptare*'s analysis.

Given that, we instrumented the consensus protocol to keep track of the time at which each round is initiated. When a new message is received, the algorithm determines the round to which the message belongs, calculates the amount of time elapsed since the beginning of that round, and sends this value to *Adaptare*. Besides that, we only had to modify the consensus code so that, before a message is broadcast, a request is made to *Adaptare* to obtain the timeout value that should be used in that round.

5.2.3 Configuring Adaptare

One fundamental issue in the proposed approach is the correct configuration of *Adaptare*, so that dynamically obtained timeout values are appropriate to achieve the desired performance. In essence, when developing the adaptive solution it is necessary to understand which are the performance objectives of the application or protocol, and translate them into a coverage requirement, provided to *Adaptare*.

Defining the optimal coverage value depends on the specific case in consideration. For some protocols it might be desirable to select timeout values that will hold most of the time, with very high probability, while in other cases it might be better to be more aggressive, using timeouts that may hold only with a smaller probability, in favor of increased reactivity.

In this case, the reasoning was the following. During the consensus execution, all processes will be sending and receiving messages, and will be able to make progress as long as they receive enough messages during a defined time interval after they initiate a round (by sending a message). They do not need to receive all the possible messages. In fact, they only need to receive messages from the majority of the processes, that is, more than $\frac{n}{2}$. Given that the total number of messages they could receive in a round is n (one from each process, including its own), the fraction of required timely messages is thus $\frac{\lfloor \frac{n}{2} \rfloor + 1}{n}$. In other words, we can say that the selected timeout must be such that it allows messages to be timely with a minimum coverage given by $C = \frac{\lfloor \frac{n}{2} \rfloor + 1}{n}$. This defines the coverage value that must be used in the configuration of *Adaptare*, which will yield a timeout value that will be sufficiently large to allow progress to be made for the observed environment conditions.

Note that the timeout might not be always sufficiently large, leading to unnecessary retransmissions. This is natural when considering stochastic processes, with continuously changing environments. For instance, if at a certain moment there is an increase in the observed delays, this may lead, in a first moment, to retransmissions caused premature timeouts. As a reaction, these increased delays will be fed into *Adaptare*, which will also provide increased timeout values to the protocol. This automatic adjustment will bring the protocol back to the expected good behavior, in which it will wait just the necessary amount of time for making progress.

5.3 Performance evaluation

We executed a set of experiments in order to quantify the improvements that are achieved by our adaptive consensus protocol. We compared the performance of the static version

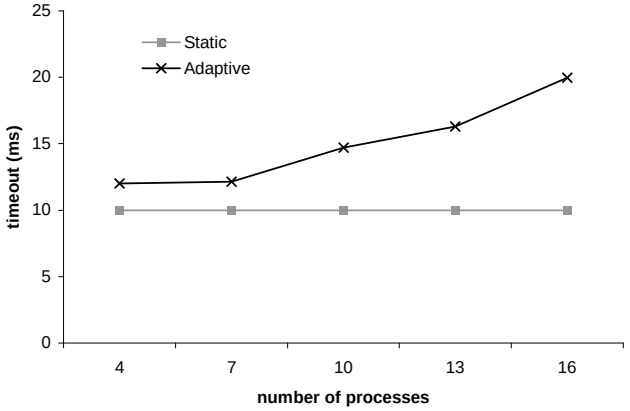
presented in [23] and of the adaptive version. The experiments were carried out on the Emulab testbed [36]. A total of 16 nodes were used, each one with the following characteristics: Pentium III processor, 600 MHz of clock speed, 256 MB of RAM, and 802.11 a/b/g D-Link DWL-AG530 WLAN interface card. The operating system was the Fedora Core 4 Linux with kernel version 2.6.18.6. The nodes were located on the same physical cluster and were, at most, a few meters distant from each other. Since the Emulab environment is not isolated, and our experiments could suffer from the interference of other nodes outside our control, we executed the experiments in six different days, to mitigate possible occasional interferences on the global results.

The number of processes participating in the consensus execution was set to 4, 7, 10, 13, and 16. Processes with odd identifiers initially propose the value 1, while processes with even identifiers propose 0, guaranteeing a divergent initial proposal set. An experiment comprises 20 consecutive executions of the static and the adaptive algorithms for a given n. We executed five experiments per day (one for each value of n), during the six different days, leading to a total of 1200 consensus executions.

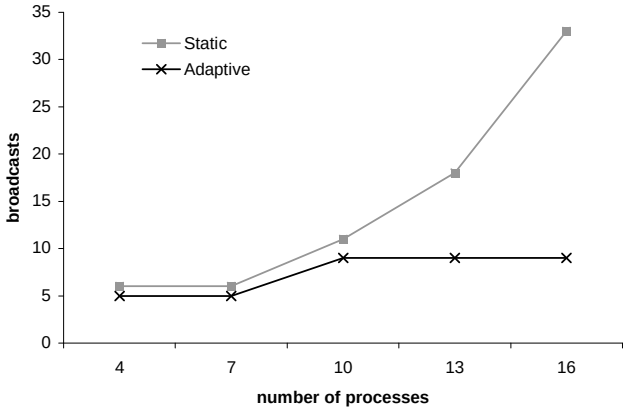
The static version of the protocol was configured to use a timeout of 10ms. This value was obtained by running a set of empirical tests with different static values, being the value that provided the better results on average. The initial timeout for the adaptive version is also 10ms, but it is dynamically adjusted according to *Adaptare* outputs. *Adaptare* was configured to use a history size of $h = 30$, as recommended in [10]. The required coverage was set to $C = \frac{\lfloor \frac{n}{2} \rfloor + 1}{n}$, as explained in Section 5.2.3.

Figure 5.2(a) shows the average timeouts. The lower timeout is the fixed value of the static algorithm, 10ms. The adaptive algorithm presents average timeouts from 12ms to 20ms, depending on the number of processes participating in the consensus execution. Since every process in the adaptive consensus computes its own timeouts independently, the local average differs from the global one. We note that this difference was, at most, 3.5ms.

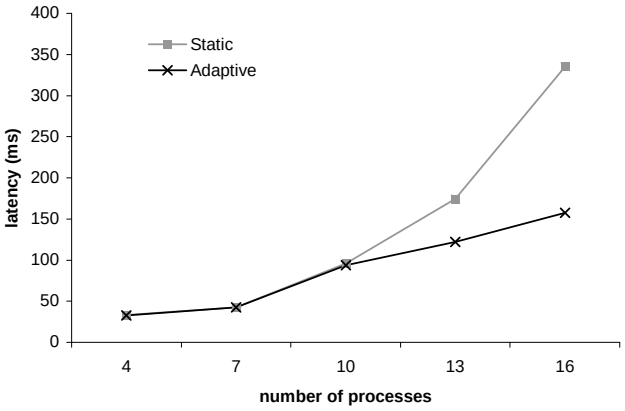
Analyzing the number of broadcast messages sent by each process (Figure 5.2(b)), the relation between timeouts and retransmissions is evident. In the adaptive consensus, which used higher timeouts, the number of broadcasts per process was significantly lower. This is a clear indication of the benefits that may be achieved by adapting the timeout. When consensus is executed by a higher number of processes, creating more contention and increased transmission delays, the static version uses an inadequate timeout value, whereas the adaptive version automatically adjusts the timeout to fit the environment conditions. By increasing the timeout, the adaptive version avoids premature retransmissions and hence prevents the network load to increase even more and affect negatively the observed network delays, as



(a) Average timeout



(b) Average number of broadcasts per process



(c) Average latency

Figure 5.2: Adaptive vs. static consensus

well as the consensus latency.

Increasing the timeout also increases the delay of necessary retransmissions. Ultimately, this delay could impact the consensus execution time, which is the fundamental performance

indicator perceived by end users. However, our results show that the overall execution time is nevertheless better for the adaptive version, which indicates a positive trade-off in favor of the increased timeouts. In fact, Figure 5.2(c) shows the latency improvements for the dynamic version, which were particularly visible for the scenarios with 13 and 16 processes. Therefore, this also confirms the experiments in the wireless environment presented in Section 5.3, which suggested that a timeout of 10ms would be sufficient for consensus among up to 10 processes, but too small for more processes. Both static and adaptive versions of the protocol achieved similar latencies in the scenarios with 4, 7 and 10 processes. However, in executions with 13 and 16 processes, with the timeout increasing to 16ms and 20ms (in average), latency improvements were about 30% and 53%, respectively.

In summary, this set of experiments revealed the importance of dynamically adjusting time-related variables of distributed algorithms according to observed changes in the operating conditions. The timeout adjustments that took place in the adaptive algorithm lead to an improvement of the network load up to 80%, and of the latency up to 50%.

Chapter 6

Conclusion and Next Steps

This deliverable describes the approach and mechanisms specified in the TRONE project to achieve automated reconfiguration and adaptation. TRONE uses a threefold strategy for improved recovery in cloud-based systems. The first solution comprises the replica and application supported recovery of the FIT event broker, which is responsible for the trustworthy dissemination of events, through Byzantine fault tolerant consensus. The second solution is based on multi-homing at the transport layer, which enhances resilience by applying mechanisms that allow fast and reliable reconfiguration in the presence of failures or security alerts. The third solution addresses the dynamic adaptation of timeouts, configuring timeout-based protocols and taking into consideration de state of the system. With such a combined approach, TRONE has the potential to improve the fault-tolerance and performance of the cloud services and infrastructure as well as the user satisfaction.

The next steps concerning the automated recovery mechanisms include the experimental evaluation in the PT cloud infrastructure, complemented by the analytical assessment of specific components (e.g. optimization problems). From the lessons learned in this phase, the final specification and evaluation of the TRONE recovery mechanisms will complete the activities of task 2.2.

Bibliography

[1] Ali C. Begen and Yucel Altunbasak. An adaptive media-aware retransmission timeout estimation method for low-delay packet video. *IEEE Transactions on Multimedia*, 9(2):332–347, 2007.

[2] Marin Bertier, Olivier Marin, and Pierre Sens. Implementation and performance evaluation of an adaptable failure detector. In *32nd IEEE Int. Conf. on Dependable Systems and Networks*, pages 354–363, 2002.

[3] Alysson Bessani. From byzantine fault tolerance to intrusion tolerance. In *5th Workshop on Recent Advances in Intrusion-Tolerant Systems (WRAITS)*, 2011.

[4] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, November 2002.

[5] W. Chen, S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. *IEEE Transactions on Computers*, 51(1):13–32, 2002.

[6] Alirio Santos de S and Raimundo Jos de Arajo Macdo. QoS self-configuring failure detectors for distributed systems. In *10th Int. Conf. on Distributed Applications and Interoperable Systems*, pages 126–140, 2010.

[7] Y. Deswarte, L. Blain, and J.-C. Fabre. Intrusion tolerance in distributed computing systems. In *Research in Security and Privacy, 1991. Proceedings., 1991 IEEE Computer Society Symposium on*, pages 110 –121, may 1991.

[8] Tobias Distler, Ivan Popov, Wolfgang Schrder-Preikschat, Hans P. Reiser, and Rdiger Kapitza. Spare: Replicas on hold. In *NDSS*. The Internet Society, 2011.

[9] M. Dixit and A. Casimiro. Adaptare-fd: A dependability-oriented adaptive failure detector. In *Proceedings of the 29th IEEE Symposium on Reliable Distributed Systems (SRDS'10)*, pages 141–147, New Delhi, India, November 2010. IEEE Computer Society Press.

[10] M. Dixit, A. Casimiro, P. Lollini, A. Bondavalli, and P. Verissimo. Adaptare: Supporting automatic and dependable adaptation in dynamic environments. *ACM Transactions on Autonomous and Adaptive Systems*, (to appear). Also as Dep. of Informatics, Univ. of Lisbon, Technical report TR-09-19, 2012.

[11] Dreibholz, Thomas and Rathgeb, Erwin P. Overview and Evaluation of the Server Redundancy and Session Failover Mechanisms in the Reliable Server Pooling Framework. *International Journal*, 2(1):1–14, 2009.

[12] Johan Eklund, Karl-Johan Grinnemo, Stephan Baucke, and Anna Brunstrom. Tuning SCTP failover for carrier grade telephony signaling. *Computer Networks*, 54(1):133–149, January 2010.

[13] Alan Ford, Costin Raiciu, Mark Handley, Sebastien Barre, and Janardhan Iyengar. Architectural Guidelines for Multipath TCP Development. IETF Request For Comments: 6182, March 2011.

[14] Joni Da Silva Fraga and David Powell. A fault and intrusion-tolerant file system. In J B Grimson and H JEditors Kugler, editors, *Proceedings of the 3rd International Conference on Computer Security*, volume 203, pages 203–218. Elsevier Science Publishers B.V., 1985.

[15] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro. Os diversity for intrusion tolerance: Myth or reality? In *Dependable Systems Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, pages 383 –394, june 2011.

[16] I. Gashi, P. Popov, and L. Strigini. Fault diversity among off-the-shelf sql database servers. In *Dependable Systems and Networks, 2004 International Conference on*, pages 389 – 398, june-1 july 2004.

[17] J.R. Iyengar, P.D. Amer, and R. Stewart. Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths. *IEEE/ACM Trans. Netw.*, 14(5):951–964, 2006.

[18] V. Jacobson. Congestion avoidance and control. *SIGCOMM Computers Communication Review*, 18:314–329, August 1988.

[19] Amit P. Jardosh, Krishna N. Ramachandran, Kevin C. Almeroth, and Elizabeth M. Belding-Royer. Understanding congestion in IEEE 802.11b wireless networks. In *5th ACM SIGCOMM Conf. on Internet Measurement*, pages 25–25, 2005.

[20] Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. Cheapbft: resource-efficient byzantine fault tolerance. In *Proceedings of the 7th ACM european conference on Computer Systems*, EuroSys '12, pages 295–308, New York, NY, USA, 2012. ACM.

[21] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative byzantine fault tolerance. *ACM Trans. Comput. Syst.*, 27(4):7:1–7:39, January 2010.

[22] Bev Littlewood and Lorenzo Strigini. Redundancy and diversity in security. In Pierangela Samarati, Peter Ryan, Dieter Gollmann, and Refik Molva, editors, *Computer Security ESORICS 2004*, volume 3193 of *Lecture Notes in Computer Science*, pages 423–438. Springer Berlin / Heidelberg, 2004.

[23] Henrique Moniz, Nuno Ferreira Neves, Miguel Correia, and Paulo Verissimo. Randomization can be a healer: consensus with dynamic omission failures. In *23rd Int. Conf. on Distributed Computing*, pages 63–77, 2009.

[24] Raul Ceretta Nunes and Ingrid Jansch-Porto. QoS of timeout-based self-tuned failure detectors: The effects of the communication delay predictor and the safety margin. In *34th IEEE Int. Conf. on Dependable Systems and Networks*, pages 753–761, June 2004.

[25] Ioannis Psaras, Vassilis Tsaoussidis, and Lefteris Mamatas. CA-RTO: A contention-adaptive retransmission timeout. In *14th Int. Conf. on Computer Communications and Networks*, pages 179–184, 2005.

[26] R. Stewart and K. Poon and M. Tuexen and V. Yasevich and P. Lei. Sockets API Extensions for Stream Control Transmission Protocol (SCTP) . IETF RFC: RFC6458, December 2011.

[27] R. Stewart (ed.). Stream Control Transmission Protocol. IETF Request for Comments: 4960, September 2007.

[28] H.P. Reiser and R. Kapitza. Hypervisor-based efficient proactive recovery. In *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*, pages 83 –92, oct. 2007.

[29] Nicola Santoro and Peter Widmayer. Time is not a healer. In *6th Symposium on Theoretical Aspects of Computer Science*, pages 304–313, 1989.

[30] Atul Singh, Pedro Fonseca, Petr Kuznetsov, Rodrigo Rodrigues, and Petros Maniatis. Zeno: eventually consistent byzantine-fault tolerance. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, NSDI'09, pages 169–184, Berkeley, CA, USA, 2009. USENIX Association.

[31] Bruno Sousa, Kostas Pentikousis, and Marilia Curado. Multihoming Management for Future Networks, 2011.

[32] Bruno Sousa, Kostas Pentikousis, and Marilia Curado. UEF: Ubiquity Evaluation Framework. In *WWIC*, pages 92–103, 2011.

[33] P. Sousa, A.N. Bessani, M. Correia, N.F. Neves, and P. Verissimo. Resilient intrusion tolerance through proactive and reactive recovery. In *Dependable Computing, 2007. PRDC 2007. 13th Pacific Rim International Symposium on*, pages 373 –380, dec. 2007.

[34] V. Stavridou, B. Dutertre, R.A. Riemenschneider, and H. Saidi. Intrusion tolerant software architectures. In *DARPA Information Survivability Conference Exposition II, 2001. DISCEX '01. Proceedings*, volume 2, pages 230 –241 vol.2, 2001.

[35] Paulo Veríssimo, Nuno Neves, and Miguel Correia. Intrusion-tolerant architectures: Concepts and design. In Rogério de Lemos, Cristina Gacek, and Alexander Romanovsky, editors, *Architecting Dependable Systems*, volume 2677 of *Lecture Notes in Computer Science*, pages 3–36. Springer Berlin / Heidelberg, 2003.

[36] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *5th Symposium on Operating Systems Design and Implementation*, pages 255–270, 2002.

[37] Timothy Wood, Rahul Singh, Arun Venkataramani, Prashant Shenoy, and Emmanuel Cecchet. Zz and the art of practical bft execution. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 123–138, New York, NY, USA, 2011. ACM.

[38] George Xylomenos and Christos Tsilopoulos. Adaptive timeout policies for wireless links. In *20th Int. Conf. on Advanced Information Networking and Applications*, pages 497–502, 2006.