The Fast and the Fair: A Fault-Injection-Driven Comparison of Restart Oracles for Reliable Web Services

Philipp Reinecke†, Aad P. A. van Moorsel‡, Katinka Wolter†
†Humboldt-Universität zu Berlin
Institut für Informatik
Berlin, Germany
{preineck,wolter}@informatik.hu-berlin.de
‡University of Newcastle upon Tyne
School of Computing Science
Newcastle upon Tyne, United Kingdom
aad.vanmoorsel@ncl.ac.uk

***Abstract*— Web Services are typically deployed in Internet or Intranet environments, making message transfers susceptible to a wide variety of network, protocol and system failures. To mitigate these problems, reliable messaging solutions for web services have been proposed that retry messages suspected to be lost. It is of interest to evaluate the performance of such reliable messaging solutions, and in this paper we therefore utilise a newly developed fault-injection environment for the analysis of time-out strategies for the Web Services Reliable Messaging standard. We compare oracles that determine retransmission times with respect to the tradeoff between two metrics: the effective transfer time and the overhead in terms of additional message transmissions. Our fault-injection environment allows faults to be invoked in the IP layer, representing a variety of failure situations in the underlying system. The study presented in this paper includes two adaptive oracles, which set the length of the retransmission interval based on round trip time measurements, and two static oracles. The study considers both HTTP and Mail as SOAP transports. We conclude that adaptive oracles may significantly outperform static oracles when the underlying system exhibits more complex behaviour.**

I. Introduction

With the continuing acceptance of Web Services technologies as a means of integrating applications, the dependability of the Web Services stack becomes increasingly important. Several attempts of defining an appropriate reliability standard have converged in Web Services Reliable Messaging (WSRM), which provides a framework to deliver messages 'reliably between distributed applications in the presence of software component, system, or network failures' [BIMT05]. Of the four delivery assurances specified in [BIMT05], both 'at least once' and 'exactly once' necessitate the retransmission of lost messages. While the standard describes positive and negative acknowledgements to determine the message transmission status, it does not provide details on the preferred waiting time (for a positive acknowledgement) until re-sending a message. Although exponential backoff is mentioned as one way to adjust the retransmission interval, the issue is effectively left open.

In this paper we experimentally investigate the influence of time-out strategies on the performance of, and overhead introduced by, WSRM. In particular, we analyse representative algorithms for four classes of restart[1] oracles (as explained in Section III). We will see that the more complex the behaviour of the underlying network and system, the more it pays off to utilise strategies that adapt the time-out value based on observed data.

There are two important aspects to the evaluation of WSRM time-out strategies we will address throughout the paper: the optimal strategy as decidable at the level of WSRM and

[1]Throughout the paper we use restart, retry, resend and retransmit interchangeably.

the interaction with reliability mechanisms at lower layers, in particular TCP. Especially the latter is extremely difficult to track, and we believe that experimental analysis such as conducted in this paper is needed to gain a better understanding of cross-layer issues. Therefore, we have created a fault injection environment for the analysis of WSRM, in which the transmission of IP packets can be interrupted. This allows us to mimic scenarios that from the perspective of the WSRM layer are representative for a variety of network, protocol and system failure behaviours. Since we inject faults at the IP level, we can experimentally evaluate the complex interaction between TCP and WSRM reliability mechanisms.

This paper combines the idea of restart to reduce completion times [vMW04] with an explicit consideration for the ensuing overhead. (We use the term overhead and fairness as referring to the same idea.) With restart, there always exists a tradeoff between fairness and timeliness. Whereas shorter intervals help to overcome faults faster, the resulting more frequent restarts consume more resources. Longer timeouts are more conservative, but also slow down recovery and thus message end-to-end delay. For all four time-out oracles we study in this paper we evaluate their application with respect to the tradeoff between fairness and timeliness.

We are not aware of other analysis of WSRM time-out mechanisms, although related and complementary work in web service fault injection as well as WSRM performance exists. In particular, Looker and Xiu [LX03] present a framework to inject faults using a modified SOAP layer. This allows one to explore the impact of specific web service failures, but this approach does not yield an understanding of the effects fault handling in the lower layers has. Regarding the impact WSRM reliability mechanisms have on performance, Pallickara *et. al.* [PFY+] study the overhead of WSRM implementations in comparison to SOAP (Simple Object Access Protocol), focusing on node-specific costs such as processing time, but not considering the specifics of the time-out mechanism. At lower levels of the stack, dependability has been studied more extensively. For instance, Allman and Paxson [AP99] explore how different parameters of RTO algorithms affect fault recovery in TCP. Our work complements such research in that we study several oracles in regard to the general tradeoff between fairness and timeliness on a higher level. We inject faults in the IP layer and thus elicit fault-handling in all layers beneath WSRM. Our study is among the first to not limit itself to the prevalent HTTP transport for web services, but also investigate SOAP over SMTP as an example for more intricate lower layer behaviour.

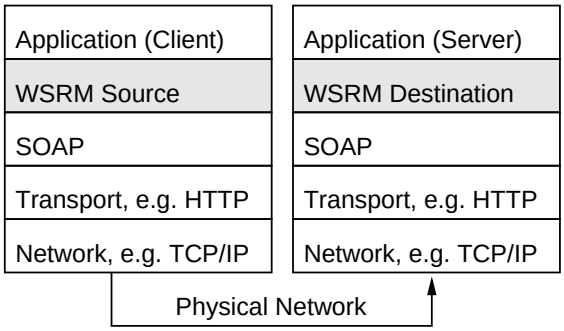

Fig. 1. A sample WSRM setting. (Adhering to SOAP terminology, we do not call HTTP an application, but rather a transport for the layers above it.)

The remainder of this paper is organised as follows. In the next section we elaborate on WSRM and the fault injection environment. Section III introduces the different restart oracles. Section IV discusses the results of our fault injection experiments.

II. WSRM and the Fault Injection Environment

Fig. 1 depicts a basic WSRM deployment. Resting beneath the application and atop a stack of further layers, the WSRM component provides reliable message transfer between the application's endpoints. The direction of message flows is reflected in the distinction between a 'WSRM Source' and a 'WSRM Destination': the sending party is the source, while the recipient is the destination. To ensure transmission, the source keeps resending each message until it receives an acknowledgement from the destination.

There are several complex layers beneath WSRM. The SOAP layer offers an abstract medium. SOAP implementations (e.g., Axis [Apaa]) then use lower-level protocols as SOAP transports to send and receive messages. As these transports are often application level protocols themselves (e.g., HTTP is an application level protocol in the TCP/IP stack), and thus they in turn utilise lower network protocol layers.

Each higher layer's reliability is influenced by faults in the layers beneath it. These faults may eventually lead to failing message transfers. On the other hand, some lower layers offer reliability as well. TCP, for instance, resends unacknowledged IP packets to provide reliable connections over unreliable IP networks. However, fault handling at any point in the stack

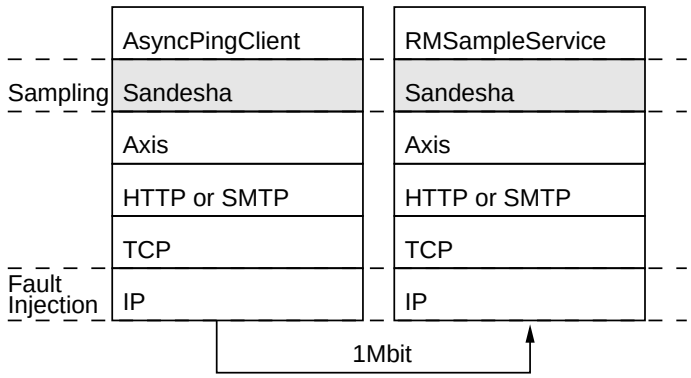

Fig. 2. Basic experiment setup.

does not necessarily guarantee reliability at the uppermost level. Furthermore, fault handling mechanisms may lead to undesirable timing behaviour in upper layers.

To elicit fault handling in the stack (including WSRM), we inject faults into the IP layer. By injecting faults at this low level we are able to investigate the inter working of TCP and WSRM reliability mechanisms. The injected faults correspond to IP packet loss, and may lead to delays and losses in message transfer. From the perspective of the WSRM layer, these effects may be representative for a wide variety of failure scenarios, including server hardware and software as well as network failures. On the other hand, faults in the higher layers (e.g., at SOAP level) will typically show distinctly different patterns than those studied here because they affect complete messages, rather than individual packets. In this paper we do not study faults at the higher layers. To study such faults, additional experimentation using a SOAP fault injection tool like [LX03] is in order.

Fig. 2 shows the basic setting for the experiments. In our simple application the client sent one 50 byte message every 500 ms. Transmission dependability was provided by the WSRM implementation Sandesha 1.0 RC1 [Apab] with in-memory storage of messages and some modifications, as follows. First, Sandesha and its SOAP engine Axis 1.2RC2 [Apaa] were de-coupled so that new messages could be sent without waiting for previous SOAP invocations to finish. Second, code to sample dispatch and receive timestamps was added. Third, a non-standard WSRM header entry was added to track transmission numbers for messages, and thus to uniquely identify them.

For SOAP over HTTP, the WSRM destination was deployed on a Tomcat 4.1 application server. For the Mail transport we enabled the prototypical implementation in Axis. Every outgoing SOAP message is encapsulated in an E-Mail and handed over to an SMTP server for delivery. The destination then periodically polls a POP3 server for new mails. Each of our machines has its own set of SMTP/POP3 servers (Postfix/Qpopper), and faults were injected on the link between the SMTP servers. Mail servers retry delivery after non-fatal faults. Deferred mails are stored in a queue and assigned a timeout for retransmission. If the fault was caused by the destination (e.g., a failed connection), no delivery to that host will be attempted for some time. The queue is periodically checked for mails to be retransmitted. In order to reduce load, and since E-Mail is not considered time-critical, all of these intervals are usually large. When delivery time becomes more important, lower timeouts seem more appropriate. We set the queue check interval and the minimum and maximum waiting times to 60 s, 60 s and 120 s, respectively, and polled the POP3 server with an interval of 6 s.

Two Linux machines, connected by a 100Mbit network and kept synchronised through NTP provided the physical environment. Paxson [Pax98] notes that NTP is designed for long-term synchronisation and thus may not be sufficiently accurate in small timescales. We tried to minimise these issues by explicitly synchronising one machine to the other before each experiment run and disabling synchronisation during the runs. On this basis we emulated a 1Mbit/1Mbit up/downstream network, which seems to us a minimum requirement for most typical applications using web services. Faults typical of poor network conditions were injected into the IP layer using a combination of Linux's NetEm queueing discipline and IP firewalling facilities (see, e.g., [HGM+]).

A. Metrics for the Fairness-Timeliness tradeoff

To describe precisely the metrics we analysed, we formalise as follows. Each message $m_i, i = 1, \ldots M$ out of a sequence of M messages will be sent $n_i \geq 1$ times, and thus there are n_i tuples (s_{ij}, r_{ij}) of dispatch/receipt times $s_{ij}, r_{ij} \in \mathbb{R}$ for each m_i. The n_i one-way transmission times for every message are defined as $t_{ij} = r_{ij} - s_{ij}$, with $t_{ij} = \infty$ for failures.

We measure the timeliness of individual messages m_i in terms of the Effective Transmission Time (ETT), defined as the time between m_i's first dispatch and its first arrival:

$$\begin{aligned} r_i^* &:= \min\{r_{ij}\} \\ ETT_i &:= r_i^* - s_{i1} \end{aligned}$$

The following line of thought yields a generic measure of fairness. m_i is sent $n_i \geq 1$ times. A number of these transmissions may fail, but at time $r_i^* < \infty$ the first one (with index k_i^*) is successful. Now while other transmissions initiated before r_i^*, but completed afterwards might have taken longer than k_i^*, they were not strictly necessary. Additionally, every retransmission after r_i^* is clearly obsolete. Thus, the only unavoidable retransmissions are those that started before r_i^*, but failed, and k_i^*. With every other transmission the sender consumed resources it did not really need. We term this behaviour Unnecessary Resource Consumption (URC)[2]:

$$\begin{aligned} k_i^* &:= \operatorname*{argmin}_j \{r_{ij}\} \\ URC_i &:= n_i - |\{r_{ij} = \infty | j < k_i^*\}| - 1 \end{aligned}$$

Since URC is independent of the nature of the resource in question (e.g., network bandwidth, processing power) and of assumptions regarding other parties accessing it (i.e., those to be fair towards), it allows us to compare fairness across many applications. It is limited in that it centers on individual messages and does not show aggregate load. The latter, however, is usually determined by the application's sending rate, and therefore out of scope for restart oracles.

III. Oracles for Restart

In our experiments, we compared time-out oracles based on different algorithms: fixed, fixed with back-offs, dynamic intervals using Jacobson/Karn and dynamic intervals using an algorithm we proposed earlier [RvMW04]. These respective oracles are representatives of different classes of time-out oracles. We explain the classification and provide the details of the oracles in this section.

In short, oracles compute timeouts for messages. Our concept of an oracle differs from that of an algorithm in that it describes a distinct component (and thus encompasses structural information) potentially combining several timeout algorithms. For example, an oracle might choose between a fair and a fast algorithm based on additional system state data. Furthermore, an oracle can be a simple heuristic, as well as an elaborate algorithm using known stochastic characteristics of the system. The four oracles we study in this paper cover the whole range.

The oracle mechanism is an almost non-intrusive way to improve performance. It only changes retransmission intervals in the sender, hence it enhances a reliability mechanism already present in virtually any communication system. Upon receipt of a positive message acknowledgement at time t the sender can determine the *transmission status* and estimate the *transmission time.* (Whereas from a negative acknowledgement it can only infer the status.) Two aspects about estimating transmission time in this way are noteworthy. First, this notion of transmission time corresponds to the complete Round-Trip Time (RTT), rather than the one-way time from sender to receiver, and secondly, if acknowledgements do not carry additional information to assign them to a particular transmission, a 'retransmission ambiguity' [KP91] may reduce accuracy of these samples.

To discuss the oracles precisely, we build forward on the formalisation in the previous section. Recall that each message $m_i, i = 1, \ldots M$ out of a sequence of M messages will be sent $n_i \geq 1$ times, and thus there are n_i tuples (s_{ij}, r_{ij}) of dispatch/receipt times for each message. The n_i transmission times for every message then are $t_{ij} = r_{ij} - s_{ij}$, with $t_{ij} = \infty$ for lost messages.

Without additional means, the sender cannot measure r_{ij}. Instead, it guesses the transmission time from positive acknowledgements. The sender observes

$$a_{ij} := \begin{cases} z & \text{an acknowledgement for } m_i \text{ arrived at time } z \\ \infty & \text{else,} \end{cases}$$

the arrival times of acknowledgements, from which it infers that the message has reached its destination some time before a_{ij}. It thus estimates t_{ij} by $\hat{t}_{ij} < a_{ij}$, e.g., by the RTT $\hat{t}_{ij}^{RTT} := a_{ij} - s_{ij}$. We distinguish four types of oracles by how they utilise this data.

A. Oracles using Fixed Intervals

A timeout τ is given at compilation time or startup, and every τ time units the sender checks whether an acknowledgement for m_i has arrived, and resends m_i otherwise. I.e., if m_i has been sent at time T_i, the sender will look for an acknowledgement at $T_i + \tau, T_i + 2\tau, T_i + 3\tau, \ldots$.

Performance of a fixed intervals oracle hinges on whether τ is consistent with actual medium characteristics. In extremis, a timeout lower than the RTT leads to retransmissions of every message, and therefore a fixed interval will rarely be used in practical systems.

In general, the larger the timeout, the more likely it becomes that expiry signals message loss and hence the need for re-

[2]Where $\operatorname{argmin}_i \{x_{i \in \mathbf{I}}\} := i \in \mathbb{I} : (x_i = \min\{x_i\})$.

transmission to ensure dependability in the strict sense. Large timeouts increase fairness by avoiding unnecessary restarts but sacrifice possible speed gains to be had by restart and result in longer loss detection times. Setting a lower timeout allows for faster failure detection and might also help avoid unusually large transmission times, but makes unnecessary retransmissions more likely.

B. Oracles using Growing Intervals

These augment Fixed Intervals by additionally tracking oracle decisions. That is, if a timeout τ_{ij} for m_i expires, they assume τ_{ij} to have been too short and increase it by a substantial amount, e.g., exponentially:

$$\begin{aligned} \tau_{i1} &= \tau \\ \tau_{i\ (j+1)} &= 2 \cdot \tau_{ij}. \end{aligned}$$

While this reduces unnecessary retransmissions, fairness does still depend on the initial timeout. Furthermore, any such approach hurts timeliness, which is most pronounced in scenarios where messages are sent or consumed synchronously.

C. Oracles using Basic Adaptive Intervals

With these oracles, the timeout is allowed to shrink as well as to grow. In essence, they parametrise an assumed probability distribution F_c for RTTs based on estimates of the transmission time, and base τ for all m_i on F_c. Because this global τ is usually closer to actual round trip times than one set beforehand, adaptive oracles can avoid unnecessary retransmissions and are able to detect loss in a timely manner. Their timeliness may be further improved by their potential to also detect (and restart) exceptionally slow invocations. However, their performance ultimately depends on (i) the similarity between the assumed and the actual distribution, (ii) the accuracy of RTT measurements, and notably (iii) the impact of the retransmission ambiguity as a factor in the accuracy of RTT measurements.

Perhaps the best known example is the TCP RTO computation which keeps track of the mean and variance of RTTs using successive samples $\hat{t}_{ij}^{l}, l = 1, 2, \ldots$. After the smoothed round-trip time (SRTT) was initialised with the first observed RTT, subsequent $SRTT_{l+1}$ are computed using

$$SRTT_{l+1} := (1-\alpha_1)SRTT_l + \alpha_1 \cdot \hat{t}_{ij}^{l+1}.$$

Fixed Intervals	Exp. Backoff	Jacobson/Karn	QEST Oracle
HTTP Transport			
$\tau_{ij} := 4\,\mathrm{s}$	$\tau_i = 4\,\mathrm{s}, 8\,\mathrm{s},$ $16\,\mathrm{s}\ldots$	$\tau := 4\,\mathrm{s}$ $\alpha_1 = 1/8$ $\alpha_2 = 1/4$ $k = 4$	$\tau := 4\,\mathrm{s}$ $\tau_{max} := 60\,\mathrm{s}$ $H = 1000$ $c = 0\,\mathrm{s}$
Mail Transport			
$\tau_{ij} := 14\,\mathrm{s}$	$\tau_i = 14\,\mathrm{s}, 28\,\mathrm{s},$ $56\,\mathrm{s}, \ldots$	$\tau := 10\,\mathrm{s}$ $\alpha_1 = 1/8$ $\alpha_2 = 1/4$ $k = 4$	$\tau := 10\,\mathrm{s}$ $\tau_{max} := 60\,\mathrm{s}$ $H = 1000$ $c = 0\,\mathrm{s}$

TABLE I

ORACLE PARAMETERS. WE CHOSE LONGER INITIAL TIMEOUTS FOR THE MAIL TRANSPORT TO ACCOUNT FOR THE LARGER ETT OBSERVED WITHOUT RESTART. SINCE STATIC ORACLES CANNOT ADAPT, WE ADDED AN EXTRA SAFETY MARGIN TO THEIR TIMEOUTS. PARAMETERS FOR THE JACOBSON/KARN ORACLE WERE SET BASED ON [AP99].

The variance measure

$$\begin{aligned} RTTVAR_1 &= \frac{1}{2}SRTT_1 \\ RTTVAR_{l+1} &:= (1-\alpha_2) \cdot RTTVAR_l + \\ &\quad +\alpha_2 \cdot |SRTT_{l+1} - \hat{t}_{ij}^{l+1}|, \end{aligned}$$

was introduced by Jacobson to improve similarity (i). In regard to the retransmission ambiguity, Karn and Partridge proposed to only use fresh samples $\hat{t}_{i1}$ from acknowledgements for which $n_i = 1$, and to back off exponentially upon encountering a timeout [KP91]. The retransmission timeout (RTO) is then computed as

$$\tau := RTO_{l+1} := \begin{cases} 2 \cdot RTO_l & \text{if } RTO_l \text{ expired} \\ SRTT_l + k \cdot RTTVAR_l & \\ \quad \text{if a new } \hat{t}_{i1}^{l+1} \text{ is available,} & \end{cases} \quad (1)$$

where k is a flexible parameter, typically set to 4 [AP99], and starts with $RTO_0 = 3\,\mathrm{s}$. Our basic adaptive intervals oracle (Jacobson/Karn) implements (1), and thus follows the notion of combining dependability and fast fault-recovery with conservative medium usage.

D. Oracles using Advanced Adaptive Intervals

Starting from the premise that the completion times T of a task (e.g., the RTT for message transmission) are drawn from independent and identically distributed random variables, governed by some probability density function $f(\hat{t}_{ij})$, it can be shown that for certain distributions $F(\hat{t}_{ij})$ (e.g., heavy-tailed) restart minimises transmission time [vMW04]. Based

on this idea, an oracle can find the optimal timeout, provided the distribution function $F(\hat{t}_{ij})$, or, more often, an estimate $\hat{F}(\hat{t}_{ij})$ thereof is known. Since timeout selection is based on a comparison of anticipated transmission times, and because $\hat{F}(\hat{t}_{ij})$ can be learned from observations, oracles employing this approach potentially improve transmission times in many scenarios, even when the distribution for T is not known beforehand. Shortcomings lie in an inherent assumption that restart does preempt previous attempts, and, as with the Basic Adaptive Intervals, in the sensitivity to inaccuracies in measurements or estimates of the transmission time of message m_i, $\hat{t}_{ij}$. If the former is not true, restart may lead to overload. We therefore applied Karn's and Karn/Partridge's RTO algorithm modifications (viz. exponential backoff and ignorance towards samples from retransmissions) to the basic oracle to enhance its fairness.

The on-line algorithm in Appendix A to [RvMW04] learns $\hat{F}(\hat{t}_{ij})$ by building a histogram. It divides the range of observed transmission times $\hat{t}_{ij} < \tau_{max}$ into H buckets $k = 1, \ldots, H$ of size $h = \tau_{max}/H$. To find the optimal timeout, it computes estimates $\hat{E}_{\tau_k}$ of the expected transmission time for restart after each of the intervals and then selects the interval with the lowest $\hat{E}_{\tau_k}$. Hence, all intervals $\tau_k = k \cdot h$ are candidate retry times. To account for known constant delays directly associated with restart, a cost value c can be set. The estimate

$$\hat{E}_{\tau_k} = \frac{\hat{M}(\tau_k)}{\hat{F}(\tau_k)} + \frac{1 - \hat{F}(\tau_k)}{\hat{F}(\tau_k)} \cdot (\tau_k + c) \qquad (2)$$

is obtained using the average return times M_k and number of samples N_k within the intervals $[(k-1) \cdot h, k \cdot h)$. If we label all known observations $\hat{t}_{ij}$ in bucket k $t_1^k, \ldots t_{N_k}^k$, M_k is estimated by

$$\hat{M}_k = \frac{1}{N_k} \sum_{l=1}^{N_k} t_l^k. \qquad (3)$$

Using the number N_k of observations that take at least τ_k time units, the estimators of the distribution function of the job completion time and of its first partial moment evaluate to:

$$\hat{F}(\tau_k) = \frac{\sum_{l=1}^{k} N_l}{\sum_{l=1}^{H} N_l + N_{\tau_{max}}}$$

and

$$\hat{M}(\tau_k) = \frac{\sum_{l=1}^{k} N_l \cdot \hat{M}_l}{\sum_{l=1}^{k} N_l}.$$

The global timeout τ is chosen to be the optimal τ_k, i.e., the one that minimises the expected RTT in (2):

$$\tau := \operatorname*{argmin}_{\tau_k} \hat{E}_{\tau_k}.$$

For a derivation of the theoretical result corresponding to (2) see [vMW04]. We label this oracle QEST, because of the venue of the original publication of the algorithm.

	No Restart	Fixed Int.	Exp. Backoff	Jac./Karn	QEST
No Faults					
avg. ETT	0.17 ± 0.01	0.16 ± 0.00	0.18 ± 0.00	0.18 ± 0.00	0.16 ± 0.00
avg. URC	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.03 ± 0.00	0.03 ± 0.00
10 s Disruption					
avg. ETT	0.27 ± 0.01	0.22 ± 0.01	0.23 ± 0.01	0.23 ± 0.01	0.27 ± 0.01
avg. URC	0.00 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.04 ± 0.00	0.04 ± 0.00
60 s Disruption					
avg. ETT	3.03 ± 0.18	2.20 ± 0.12	2.12 ± 0.12	2.34 ± 0.13	2.38 ± 0.13
avg. URC	0.00 ± 0.00	0.42 ± 0.03	0.14 ± 0.01	0.06 ± 0.00	0.07 ± 0.00
2% Packet Loss					
avg. ETT	–	0.76 ± 0.02	0.77 ± 0.02	0.79 ± 0.02	0.73 ± 0.01
avg. URC	–	0.07 ± 0.00	0.08 ± 0.00	0.18 ± 0.01	0.34 ± 0.01
10% Packet Loss					
avg. ETT	–	3.83 ± 0.06	4.57 ± 0.10	4.78 ± 0.08	2.87 ± 0.04
avg. URC	–	0.25 ± 0.01	0.22 ± 0.01	0.25 ± 0.01	0.33 ± 0.01

TABLE II

SUMMARY OF RESULTS FOR THE HTTP TRANSPORT. ETTS GIVEN IN SECONDS, WITH 95% CONFIDENCE INTERVALS. WE DO NOT SUPPLY ETT AND URC FOR 'NO RESTART' IN THE LAST TWO SCENARIOS BECAUSE IN THESE ALL RUNS LOST MESSAGES.

IV. RESULTS

We studied two sets of scenarios. In the first one, an otherwise perfect network connection temporarily exhibited 100% packet loss of variable duration ($10\,\mathrm{s}$ and $60\,\mathrm{s}$) after an initial 60 s warm-up period. In the second, network conditions remained stable throughout the experiment, with several levels of random packet loss enforced on outgoing packets on both sides. Based on Paxson's observation that average packet loss on the Internet may reach up to 5.2% [Pax97], we injected loss at 0%, 1% and 5% in each direction (resulting in an effective loss of 0%, 2% and 10%, respectively, for bidirectional communication). Table I shows the parameter values for all oracles in our experiments. In addition to these oracles, we also conducted a series of experiments without restart.

Care has been taken to both prevent software aging from changing the results and to minimise the influence of outliers. Preliminary runs indicated severe aging within Sandesha that

	No Restart	Fixed Int.	Exp. Backoff	Jac./Karn	QEST
No Faults					
avg. ETT	4.63 ± 0.05	4.60 ± 0.05	4.69 ± 0.05	4.64 ± 0.05	4.56 ± 0.05
avg. URC	0.00 ± 0.00	0.05 ± 0.00	0.06 ± 0.00	0.01 ± 0.00	0.02 ± 0.00
10 s Disruption					
avg. ETT	4.72 ± 0.06	4.76 ± 0.05	4.71 ± 0.05	4.69 ± 0.05	4.66 ± 0.05
avg. URC	0.00 ± 0.00	0.06 ± 0.00	0.06 ± 0.00	0.02 ± 0.00	0.00 ± 0.00
60 s Disruption					
avg. ETT	13.79 ± 0.58	11.58 ± 0.38	11.70 ± 0.44	13.37 ± 0.55	11.93 ± 0.46
avg. URC	0.00 ± 0.00	0.61 ± 0.03	0.32 ± 0.01	0.04 ± 0.00	0.12 ± 0.01
2% Packet Loss					
avg. ETT	5.11 ± 0.06	5.16 ± 0.06	5.25 ± 0.06	5.18 ± 0.06	5.15 ± 0.06
avg. URC	0.00 ± 0.00	0.10 ± 0.01	0.12 ± 0.01	0.02 ± 0.00	0.01 ± 0.00
10% Packet Loss					
avg. ETT	9.07 ± 0.17	9.60 ± 0.11	8.97 ± 0.12	8.61 ± 0.14	8.85 ± 0.15
avg. URC	0.00 ± 0.00	0.60 ± 0.01	0.49 ± 0.01	0.03 ± 0.00	0.02 ± 0.00

TABLE III

RESULT SUMMARY FOR THE MAIL TRANSPORT. ETTS GIVEN IN SECONDS, WITH 95% CONFIDENCE INTERVALS. DUE TO LIMITED TIME, THESE ARE BASED ON ONLY 5 RUNS PER ORACLE, I.E., 10000 SAMPLES.

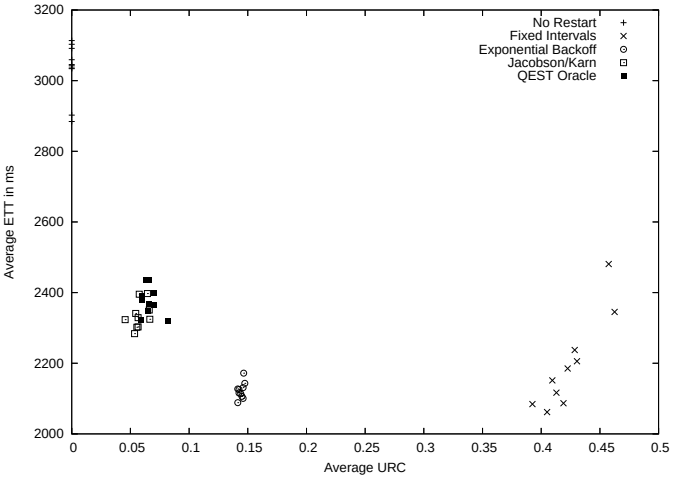

Fig. 3. Performance with 60 s disruptions (HTTP Transport). All scatter plots show 95% confidence bars for both measures.

resulted in exponentially growing transmission times when more than 2500 messages were sent. As a consequence we divided the experiments into runs of 2000 messages each and restarted both server and client before each run. Since results from these runs are sensitive to random influences, we repeated experiment runs for each oracle ten times per scenario.

In a summarised view (Table II and III), no single oracle clearly outperforms all others in all scenarios and with all transports. We will see, however, that the adaptive oracles perform better under more complex network and system conditions, such as exemplified by the SMTP experiments.

A. HTTP Transport

a) 60 s Disruption: The scatter plot in Fig. 3 shows average measures for each of the ten runs per oracle in the scenario with a 60 s disruption. ETT is depicted over URC, hence, the closer to the origin they are, the better the oracles perform regarding the tradeoff. We note that all oracles are faster and less fair than the runs without restart, and that there is little variation between runs. The adaptive oracles are fairer and slower than the static ones, and Exponential Backoff performs best in both dimensions.

Both the gain in timeliness through restart and the differences between oracles can be explained by the way TCP handles packet loss. Fig. 4 shows ETT and URC for messages hit by a 60 s disruption. Without restart, ETT depends entirely on the TCP's fault-handling. In this case, we see step-wise constant ETT that start slightly above 90 s and then decrease sharply. This reflects the TCP RTO timeout mechanism. Each transmission attempt required one TCP connection to the server. To set up a connection, TCP engages in a three-way handshake with the server. The initiating party first sends a SYN packet to the destination. If there is no reply, it retransmits the packet. The interval between retransmissions is determined by the RTO, which starts at 3 s and doubles on every retransmission: 6 s, 12 s, ... Waiting times for the connections (and hence for the messages) grow accordingly, for instance $3\,\mathrm{s}+6\,\mathrm{s}+\ldots 48\,\mathrm{s} = 93\,\mathrm{s}$ for messages whose setup phase experiences 5 TCP timeouts, and $3\,\mathrm{s}+6\,\mathrm{s}+12\,\mathrm{s}+24\,\mathrm{s} = 45\,\mathrm{s}$ if the 4th retransmission is successful. In effect, TCP delayed messages by up to 93 s seconds—much longer than the duration of the fault (60 s).

All oracles yielded ETTs at most slightly above 60 s. By restarting, they initiated new TCP connection setups, i.e., for each restart the TCP entered a new three-way handshake and immediately sent a new SYN packet. Since there were more connection setups, one of them was likely to hit the end of the disruption without engaging the RTO. This connection would then be established faster than one that had to wait for its backed-off RTO to time out to detect the end of the fault and complete its connection setup. Fixed Intervals exemplify this best. A message sent exactly at the beginning of the fault ($t = 60\,\mathrm{s}$) could be subject to at most ${}^{60\,\mathrm{s}}/_{4\,\mathrm{s}} = 15$ timeouts, the last of which was at $t = 120\,\mathrm{s}$. Since the fault was over by then, the connection setup for the 16th transmission

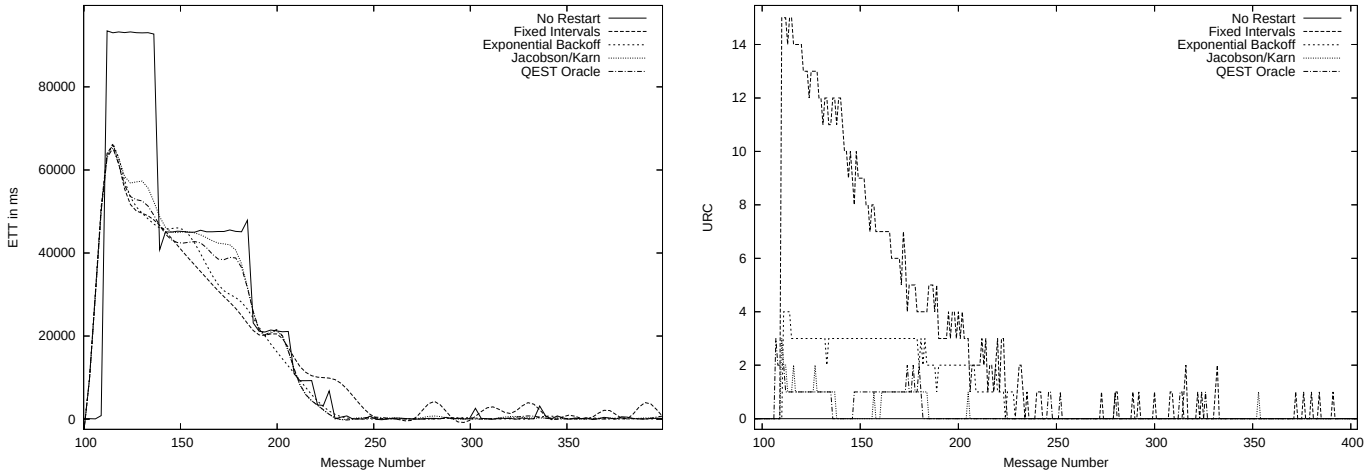

Fig. 4. ETT and URC for messages hit by a 60 s disruption (HTTP Transport). ETT curves have been smoothed to improve readability.

(15th retransmission) succeeded, and the message could be transmitted almost immediately, with a low transmission time $t_{i16} = r_{i16} - s_{i16}$. ETT is thus $r_{i16} - s_{i1} \approx 60\,\mathrm{s}$. On the other hand, none of the other transmissions failed. They reached the destination as well, albeit with higher completion times dominated by the TCP (e.g., $t_{i1} \approx 93\,\mathrm{s}$, see above). Therefore, URC for this message is $16 - 1 = 15$. Exponential Backoff achieved the same ETT in a fairer manner. With restarts at $t = 60 + 4\,\mathrm{s}, 60 + 12\,\mathrm{s}, 60 + 28\,\mathrm{s}, 60 + 60\,\mathrm{s}$, there were only five transmissions, four of which were unnecessary. Both adaptive oracles' even lower URC is a result of their global timeout. With every elapsed τ_{ij}, be it for a new transmission or one attempted previously, τ grew exponentially, which, due to the number of such events, yielded a very rapid increase. When the first message was again transmitted without a restart, τ_{ij} dropped to about the same value as before the disruption, and then all messages previously held back by the higher timeout were retransmitted at once.

b) Packet Loss: Without retransmissions, packet loss rates of 2% and 10% led to message loss, i.e., not all messages that were sent reached the destination. This highlights the need for a reliability mechanism on top of HTTP. Although TCP provides reliable connections for HTTP, the HTTP transport can still fail. Again, this can be attributed to the way TCP handles faults. As pointed out above, packet loss in the setup phase delays a connection by at least 3 s. Furthermore, the number of TCP segments exchanged during HTTP transfers is usually small. In consequence, TCP connections that carry HTTP often do not leave the slow-start phase, and thus congestion control prevents fast fault-handling (via duplicate ack detection) from

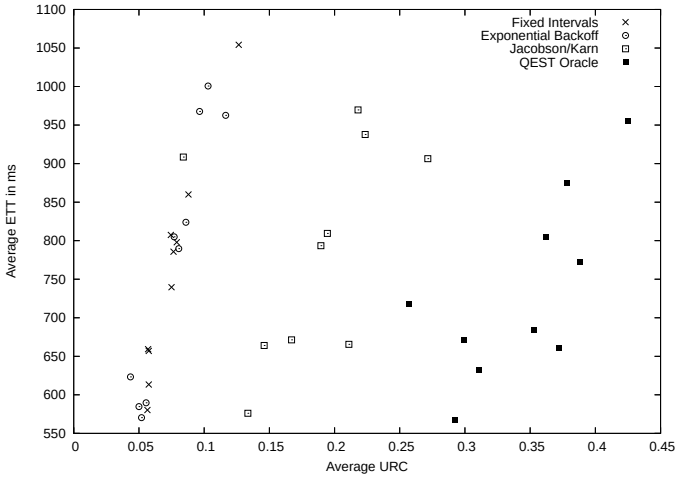

Fig. 5. Performance with 2% packet loss (HTTP Transport).

taking effect. (See pp. 303–306 in [KR01] for details.) With loss rates as high as those studied here, TCP's fault-handling is therefore likely to manifest in connections that are delayed for large amounts of time. Actual implementations, however, cannot wait forever and have to give up eventually. Since HTTP does not retry failed connections, these timeouts transform into message loss.

In regard to oracle performance, we observed different outcomes depending on the packet loss rate. With 2% loss, there were only small differences between oracles (see Table II). Fairness, on the other hand, varied considerably. Both static oracles are much fairer than the adaptive ones. Obviously, more frequent restarts did not help timeliness. If we look at the scatter plot for this scenario (Fig. 5), we observe that for all oracles higher URCs tend to correspond to higher ETTs, which

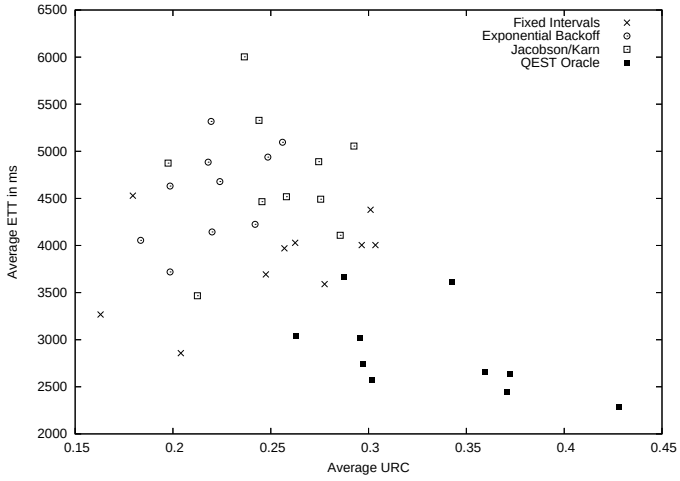

Fig. 6. Performance with 10% packet loss (HTTP Transport).

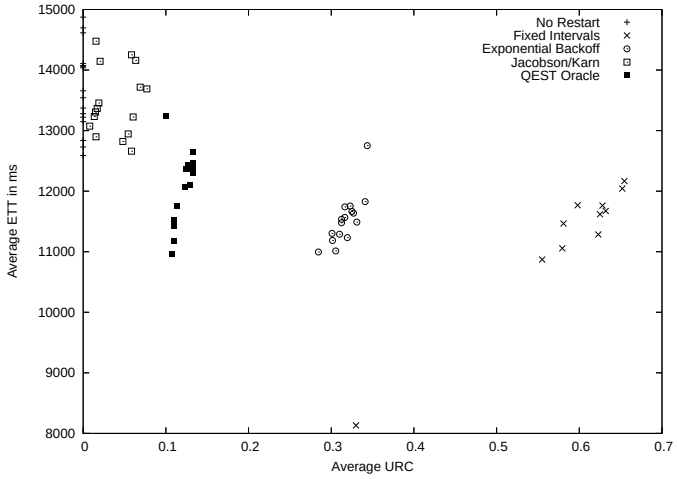

Fig. 7. Performance for 60 s disruptions (Mail Transport).

is contrary to our notion of a tradeoff between timeliness and fairness. This fact indicates that the costs associated with restart might indeed by high enough to offset its benefits.

Results for the 10% scenario demonstrate that, as the loss rate increases, restart begins to pay off in terms of an ETT improvement. Here, the QEST Oracle as the least fair is also the fastest. Our finding that with 10% packet loss more frequent restarts yield better timeliness is corroborated by the scatter plot (Fig. 6), where we note a general trend towards lower ETT with higher URC even within observations for the QEST Oracle.

B. Mail Transport

As with HTTP, the scatter plot for a 60 s disruption when using Mail (Fig. 7) shows clearly defined clusters for all oracles. Here, the QEST Oracle performs best, and both static ones are less fair, but not faster. The Jacobson/Karn Oracle exhibits a fairness and timeliness almost identical to runs without restart.

These findings can again be explained by characteristics of the SOAP transport. SMTP servers utilise TCP to transfer mails to their destination. As we laid out in the discussion of HTTP results, TCP connections can be delayed by packet loss. When using the HTTP transport, very long delays lead to SOAP message loss. Unlike the HTTP transport, Mail does employ a reliability mechanism: SMTP servers retry delivery attempts that failed due to non-permanent faults. TCP failures, and failed connection setups in particular, are usually considered transient. The SMTP server detects these faults by means of its own timeouts. To the client's SMTP server, the 60 s disruption manifested itself as repeated timeouts when connecting to the destination. The SMTP server then used the queueing system we sketched earlier to retransmit the deferred messages.

We can, unfortunately, not endeavour to fully explore the intricate interactions between timeouts and queues in the SMTP server and restarts initiated by the WSRM component. We will thus only point out interesting aspects of the ETT curves shown in Fig. 8 and their implications for oracle performance. What first meets the eye is a pronounced two-peak 'saw-tooth' pattern for transmissions without restart, probably caused by periodic flushing of a queue. Second, both static oracles generally manage to achieve ETTs below these peaks. The apparent gain is especially large with the second peak. Here, both static oracles were about 80 s faster than No Restart. Third, the transport exhibits some sort of memory effect. In all runs, ETT curves for the static oracles sport at least one additional peak with a height of about 40 s and a width of about 120 messages that absorbed previous gains. Fourth, even though the QEST Oracle reduces ETT to roughly 70 s during the second peak, it does not provoke delays in later messages. This fact explains its optimal timeliness. Finally, with the Jacobson/Karn Oracle restarts are almost non-existent (see URC plot in Fig. 8), hence the similarity of its results to those without restart.

V. Conclusions

In this paper we analysed restart oracles in Web Services Reliable Messaging, using a fault injection test bed. The faults are injected at the IP level, and represent a variety of lower-level network and system faults. Fault injection at the IP level

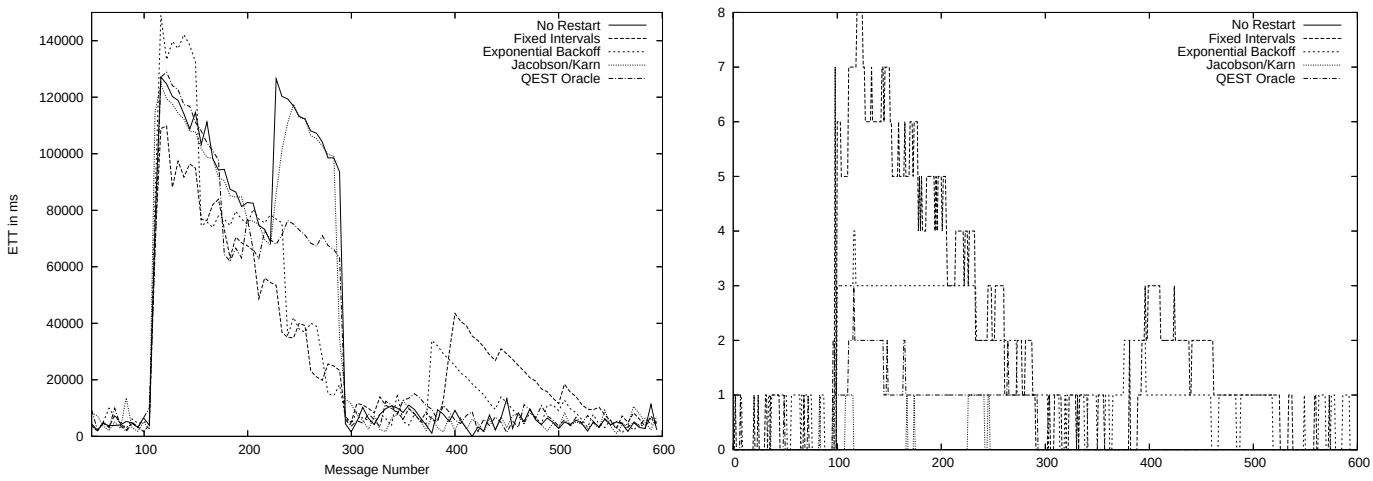

Fig. 8. ETT and URC for messages hit by a 60 s disruption (Mail Transport). ETT curves have been smoothed.

allows us to investigate experimentally the consequences of the intricate relation between TCP and WSRM time-out and retransmission mechanisms. We carried out experiments for both HTTP and SMTP as SOAP transports.

The recommendation supported by our study is that when using the HTTP transport on top of a simple network, static oracles such as fixed intervals can suffice to maintain a good balance between fairness and timeliness. However, more sophisticated adaptive oracles will be fairer when the system exhibits long periods of packet loss, without being much slower, and faster with continuously high-loss rates. For the Mail transport, which exhibits more intricate timing behaviour, adaptive oracles are a better choice.

REFERENCES

[AP99] M. Allman and V. Paxson. On Estimating End-to-End Network Path Properties. *ACM SIGCOMM*, September 1999.

[Apaa] Apache. Apache Axis. `http://ws.apache.org/axis/`.

[Apab] Apache. Apache Sandesha. `http://ws.apache.org/sandesha/`.

[BIMT05] BEA Systems, IBM, Microsoft Corporation Inc, and TIBCO Software Inc. Web Services Reliable Messaging Protocol (WS-ReliableMessaging), February 2005.

[HGM+] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Schroeder, and P. Larroy. Linux Advanced Routing and Traffic Control. `http://lartc.org`.

[KP91] P. Karn and C. Partridge. Improving Round-Trip Time Estimates in Reliable Transport Protocols. *ACM Transactions on Computer Systems*, 9(4):364–373, November 1991.

[KR01] B. Krishnamurthy and J. Rexford. *Web Protocols and Practice.* Addison Wesley, 2001.

[LX03] N. Looker and J. Xiu. Assessing the Dependability of SOAP-RPC-Based Web Services by Fault Injection. In *9th IEEE International Workshop on Object-oriented Real-time Dependable Systems*, pages 163–170, 2003.

[Pax97] V. Paxson. End-to-End Internet Packet Dynamics. In *Proceedings of the ACM SIGCOMM '97 conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, volume 27,4 of *Computer Communication Review*, pages 139–154, Cannes, France, September 1997. ACM Press.

[Pax98] V. Paxson. On Calibrating Measurements of Packet Transit Times. In *SIGMETRICS '98/PERFORMANCE '98: Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and Modeling of computer systems*, pages 11–21, New York, NY, USA, 1998.

[PFY+] S. Pallickara, G. Fox, B. Yildiz, S. L. Pallickara, S. Patel, and D. Yemme. An Analysis of the Costs for Reliable Messaging in Web/Grid Service Environments. `http://grids.ucs.indiana.edu/ptliupages/publications/Wsrm-Performance.pdf`.

[RvMW04] P. Reinecke, A. van Moorsel, and K. Wolter. A Measurement Study of the Interplay between Application Level Restart and Transport Protocol. In *Proc. International Service Availability Symposium (ISAS)*, volume 3335 of *Lecture Notes in Computer Science*, Munich, May 2004. Springer.

[vMW04] A. van Moorsel and K. Wolter. Analysis and Algorithms for Restart. In *Proc. 1st International Conference on the Quantitative Evaluation of Systems (QEST)*, pages 195–204, Twente, The Netherlands, September 2004.