From static to dynamic protocols: adapting timeouts for improved performance[*]

A. Casimiro[1], **M. Dixit**[1]

[1]University of Lisbon, Faculty of Sciences (FCUL)

{casim,mdixit}@di.fc.ul.pt

Abstract. *When network delays are unstable and susceptible to network contention, such as in wireless environments, it becomes important to dynamically adapt timeout values in order to address performance concerns. In this paper we discuss the problem of transforming static timeout-based protocols into protocols that dynamically select timeout values for improved performance. We propose a methodological approach and we present an example that illustrates how the methodology applies in practice.*

1. Introduction

In asynchronous distributed systems, a fundamental problem is to ensure the progress of a protocol despite the possible occurrence of process crashes or network failures. A typical solution is to use timeouts, which prevent waiting indefinitely for messages that may never arrive. When the timeout expires, the protocol assumes that a failure occurred and decides accordingly. However, since the decision might be wrong (if the system is just slow), special care must be taken to secure safety properties. In consequence, *the typical concern in the definition of protocols for asynchronous systems is to ensure safety, while the problem of selecting a good timeout value is often ignored.* In this paper we argue that in asynchronous systems it is also important to carefully select timeouts, namely in wireless settings, to improve performance. We propose an approach to transform static protocols into dynamic ones, in which timeouts are adaptive, and we provide a short example to illustrate how the approach may be applied to an existing protocol.

2. Motivation

Why is the problem of timeout selection often ignored? In general, this happens because of the following main reasons. First, sometimes timing assumptions are hidden from the protocol (e.g. encapsulated in a failure detector), making it impossible to directly deal with the issue of timeout selection. Second, independently of the specific timeout values, safety is preserved and only performance may become an issue. Finally, using large timeouts is often not a problem in the normal case, and just affects performance when faults indeed occur. In this case, the trade-off will be the possible large delay to deal with exceptions, that is, to detect faults. Performance ends up being dependent on: a) how large timeouts must be set, and b) how frequent are still the exceptions.

Therefore, it is fundamental to know about the temporal behavior of the execution environment. Local Area Networks (LANs) constitute a very favorable environment from

[*]This work was partially supported by the FCT, through project TRONE (CMU-PT/RNQ/0015/2009) and the Multiannual and CMU-Portugal programmes.

a temporal perspective. Network delays are very small (in the order of microseconds), very stable across different LANs, independent of the communicating nodes and they are not much affected by contention. In general, in LANs it is possible and easy to select some small upper bound for communication delays, which will hold with a very high probability. Therefore, distributed protocols usually perform well in these environments. On the other hand, in large-scale networks (WANs) delays are much higher and they strongly depend on the specific end points. Moreover, they are not so stable over time, due to route changes and global traffic load fluctuations. Therefore, dynamic timeout selection and adaptation becomes more important in these environments. The problem is even more relevant in wireless networks. Delays are also much larger than in LANs and, in addition, they are strongly affected by contention. Depending on the number of nodes that communicate within a single hop distance, the delay can significantly change. This is illustrated in Figure 1, which compares the execution time of a consensus protocol in a LAN and in a wireless network. The results were obtained using a LAN in our lab and the Emulab testbed [White et al. 2002], for a different number, n, of processes.

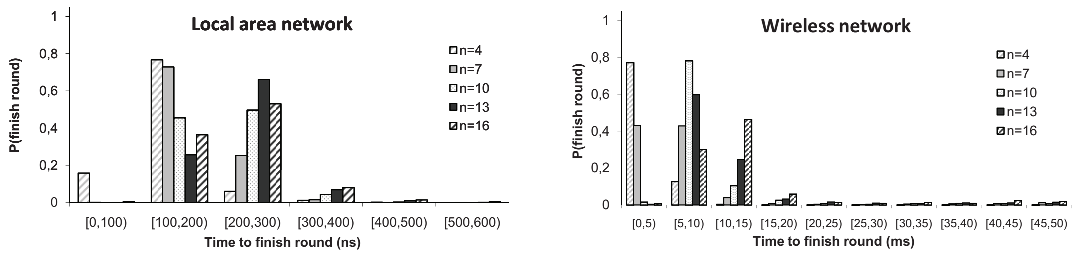

Figure 1. Protocol execution delays in a LAN and in a wireless network.

Given the previous discussion, we argue that the problem of timeout selection, which has been treated as a minor issue in the context of distributed protocols for asynchronous systems, should be given particular attention when considering large-scale and wireless networks, for performance and practicality reasons. Furthermore, we believe that it is possible to transform static protocols, which do not consider adaptive timeouts, into dynamic ones that perform better in environments of varying timeliness. This is what we discuss next.

3. A methodology for structured timeout adaptation

Given a static timeout-based protocol, the objective is to make it dynamic in the simplest possible way. We propose a modular approach, in which there is an independent service that provides the *right* timeout, which depends on the requirements specified by the protocol or application. A mapping layer is needed in order to translate performance objectives into requirements that the service can handle. Given that, we consider a composition involving the timeout provisioning service, the mapping layer and, on top, the timeout-based protocol or application.

3.1. Timeout provisioning service

The objective of the timeout provisioning service is to provide a timeout to be used by the protocol, which may be assumed to hold with a given probability, also called coverage. In our previous work we have designed *Adaptare* [Dixit et al. 2011], which is a probabilistic framework for dependable adaptation that may be used as a timeout provisioning service. *Adaptare* assumes that observed temporal variables can be described in probabilistic terms, that distributions change over time, but that they do not change arbitrarily fast. This allows to collect enough information for building useful probabilistic characterizations. The input to *Adaptare* are sample observations of the temporal variable (for instance, continuously measured network delays). Using these samples *Adaptare* tests their fit to various probabilistic distributions in order to obtain a probabilistic characterization of the current state. *Adaptare* is then able to determine a bound (a timeout), which will hold with (at least) the specified coverage, provided at its interface.

3.2. Mapping layer

In order to obtain a timeout from *Adaptare* it is necessary to provide the required coverage for the timeout. The coverage must be derived from performance requirements, although this might not be trivial. In the simplest case, it will be known that the best performance is achieved for a certain coverage. For instance, the performance of timeout-based failure detectors may be specified in terms of the mistake rate, which directly corresponds to the probability that a timeout expires too early. In this case, no specific mapping layer would be needed as the mistake rate would serve as the required coverage. In other cases, performance requirements are specified in terms of variables that cannot be directly translated into a coverage, like the execution time. In these cases, a mapping layer will be used to translate requirements into a coverage value, which will be bound to a specific protocol or application. In fact, the layer will implement a cost/benefit function expressed in terms of $\langle$coverage,timeout$\rangle$ pairs. These pairs are obtained from *Adaptare* whenever necessary, that is, when the protocol asks for a new timeout. When this happens, the mapping layer will execute a procedure that uses the cost/benefit function to determine if there is a new $\langle$coverage,timeout$\rangle$ pair that brings a relative improvement with respect to the currently considered pair. When the probabilistic characterization of the environment changes, it is expected that a new timeout will be selected.

3.3. Timeout-based protocol

For transforming a static timeout-based protocol into a dynamic one, it is fundamental to first identify the timing variable of interest, on which the timeout is applied. For instance, in a pull-based failure detector the variable of interest is the round-trip delay of the request/reply pair, while in some round-based protocol it may be the delay for completing a round. Then, the protocol must be instrumented for monitoring the temporal variable and for sending the measured values to *Adaptare*. Finally, whenever the timeout is used there must be a call to *Adaptare* for updating its value. The approach requires some changes to the protocol code, in particular to collect samples of the temporal variable. This is a fixed development cost, but the overhead in performance is not significant.

4. Application example

We applied the described approach to a consensus protocol designed for wireless environments [Moniz et al. 2009], for improving its performance. The protocol executes in rounds with several phases, in which every process sends a broadcast and waits for a given number of messages in order to make progress towards a consensus decision. A timeout prevents processes to wait indefinitely if no sufficient messages are received. In this case, processes start a new phase and resend the broadcast. In the original implementation, the timeout was fixed.

To be able to use an adaptive timeout, we first identified the temporal variable to be the time required for receiving a message, measured from the beginning of a new phase. All the delays are measured and used to feed a local instance of *Adaptare*, which is thus able to characterize the time it takes to receive a message. Concerning the mapping layer, the fast termination requirements of the consensus protocol were transformed to a coverage requirement in the following way. Given the protocol definition, it is possible to know how many messages must be received in each phase in order to make progress, which is $n/2 + 1$ for n participating processes. Therefore, a good criteria for achieving optimized performance is to use a timeout whose coverage (the probability that it will be large enough) allows to receive this precise number of messages. The coverage is thus set to the number of required messages over the total number of possibly received messages.

We ran some experiments in the Emulab wireless environment to compare implementations with static and dynamic timeouts. In particular, we wanted to know how the number of nodes would impact on performance. Therefore, the experiments were done with 4, 7, 10, 13 and 16 nodes. As expected, we were able to observe the benefits of using an adaptive timeout. In particular, we observed that the average timeout in the dynamic implementation was increasing linearly from 13ms (with 4 processes) to about 25ms (with 16 processes), thus adapting to the increased contention and communication delays, when increasing the number of nodes. In the static implementation, with a 10ms timeout, the average latency of consensus was increasing much faster, reaching 225ms for 16 processes, while it was just 114ms in the dynamic version. We also observed that the average number of broadcasts per round was kept almost stable in the dynamic version (between 3 and 6), while in the static version it increased from 4 to 23.

5. Conclusion

In this short paper we argued about the need for transforming static into timeout adaptive protocols, namely in wireless networks. We proposed an approach that requires a minimum amount of changes to static protocols, while still allowing them to benefit from dynamic timeouts. A timeout provisioning service is key to the approach, in conjunction with a mapping layer. The approach was applied to transform a static consensus protocol for wireless environments and the results allowed us to conclude that it is possible to achieve the intended performance improvements.

References

Dixit, M., Casimiro, A., Lollini, P., Bondavalli, A., and Verissimo, P. (2011). 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 Lisboa, Technical report TR-09-19.

Moniz, H., Neves, N. F., Correia, M., and Verissimo, P. (2009). Randomization can be a healer: consensus with dynamic omission failures. In *Proceedings of the 23rd International Conference on Distributed Computing*, pages 63–77.

White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. (2002). An integrated experimental environment for distributed systems and networks. In *5th Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, USA.