Consensus in WSN – Identifying critical protocol mechanisms

A. Köpke H. Karl
A. Wolisz
Telecommunication Networks Group, Technische Universität Berlin
Sekr. FT 5, Einsteinufer 25
10587 Berlin, Germany
{koepke, karl, wolisz}@tkn.tu-berlin.de
http://www.tkn.tu-berlin.de

1 Introduction

Consensus is an important problem in distributed systems, such as Wireless Sensor Networks (WSNs). It occurs in many places, for instance in the tedious task of updating the software in the nodes in a WSN installation, but also whenever sensors need to agree on a value or actuators agree on an action.

As an example, consider the case where a WSN has been retrofit to a building to save energy, e.g. by using sensor and actuators to open or close the blinds of a room. In such a system, many decisions depending on e.g. temperature, sun shine and wind are possible. The actuators could for instance agree to close all blinds. Or they could agree to half-close all blinds, or to close blind A, while opening blind B and C.

The dependencies between actuators lead to a strict meaning of decision: once a decision is made, an actuator relies on the fact that the other actuators behave according to the decision – there must be a consensus among the actuators.

The question arises how such a consensus can be reached in an energy-efficient manner in a WSN. In a typical solution, one node would start by proposing a value to all participating nodes. After this multicast it expects an answer from all other nodes back. These answers will usually be correlated in time (all nodes start sending shortly after receiving the proposal). Correlation in time means that the network has to deal with a lot of messages within a short period – in the extreme case this can exceed its short-term capacity. However, in a WSN such correlation can be used to aggregate values, turning the apparent disadvantage into an opportunity.

Consequently, we derive a protocol that solves the consensus problem using aggregation. This use of aggregation considerably improves energy efficiency without influencing the quality of the consensus or increasing the time it takes to reach a decision.

The protocol can be formulated using two kinds of aggregation: concatenation or computation of a single value that summarizes all the information. Concatenation allows to find out which nodes did not answer. This can be used to initiate retransmission requests, hence making the transmission more reliable, but in turn increases the packet size. If only a single value is computed, this is not possible. Still, this

This work has been partially sponsored by the European Commission under the contract IST-2001-34734 – Energy-efficient sensor networks (EYES).

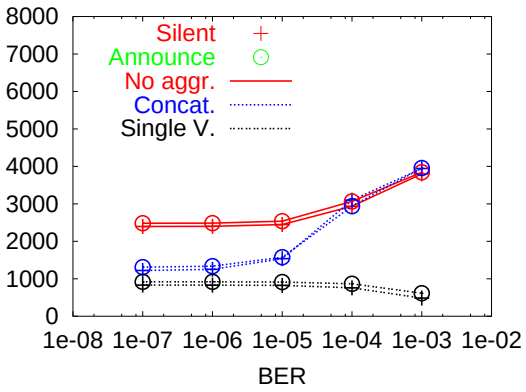

Figure 1: Sent bits per node, 400 nodes, without ARQ

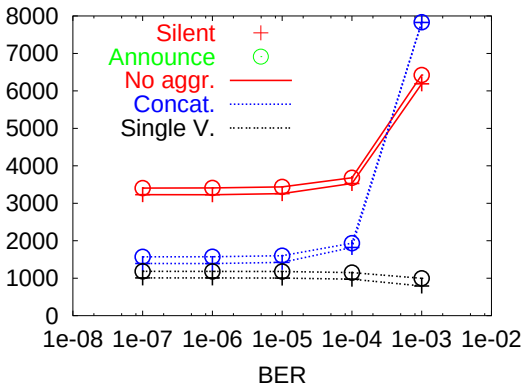

Figure 2: Sent bits per node, 400 nodes, with ARQ

low-cost solution may be sufficient for some applications. In this paper we investigate how reliable the three versions (no aggregation, concatenation, single value) are, measured in the fraction of nodes that correctly decide. We also evaluate how much time these versions need until the protocols terminate and how much energy is spent, measured in the number of bits sent.

2 Selected results

The results presented in this section are averaged for 20 random topologies, simulations were run until a relative precision of 0.01 had been reached (i.e., the actual value is with 95% confidence between $\frac{1}{1.01}\overline{x} \leq \mu \leq 1.01\overline{x}$).

The first metric to look at is the number of sent bits. Fig. 1 and 2 show (with or without ARQ for the unicast communication used to request retransmissions of missing proposal/acknowledgements from individual sensors) the average number of bits sent by each node per consensus attempt as a function of the Bit Error Rate (BER) for a sensor network consisting of 400 nodes, all taking part in the Two-Phase commit (2PC) protocol. The graphs show all the six possible combinations: two ways of building a tree and three forms of value aggregation. The way how a tree is built is denoted by different points in the graphs: if no messages are sent to built the tree, we call this a “silent” tree (the graphs have crosses) whereas when the tree is built using announcements “I am your child” to inform the parent node about the presence, the graphs have circles.

From these figures it becomes clear that how the tree is built has only a minor influence. A more pronounced influence is seen in the way how values are aggregated. The variant that aggregates the values in a single value clearly sends the smallest number of bits and this number even drops with a rising BER. This is due to the fact that with a higher BER more and more messages are lost and not forwarded – resulting in a dropping number of sent bits in downstream nodes. The concatenating variant sends more bits, and this can even exceed the number of bits send by the non-aggregating version – the reason is that the long packets of the concatenating version are more susceptible to losses.

Another question is whether the use of aggregation has an influence on the time it takes for the algorithm to terminate. Figures 3 and 4 show that the use of aggregation does not necessarily increase the time until the protocol terminates. In fact, this is more related to the way timers are chosen for the leafs to start the convergecast, especially for small BER. The problem here is that the possible depth of the tree is vastly overestimated with the maximum number of hops of 30, usually the average number

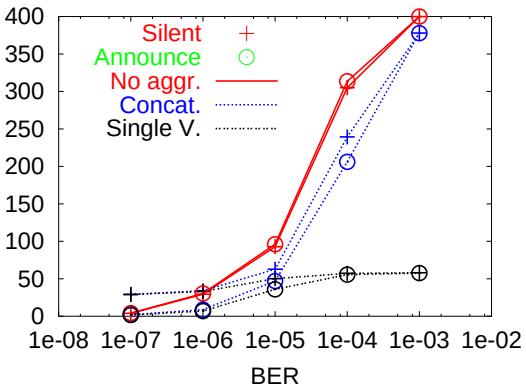

Figure 3: Time to terminate (s), 400 nodes, without ARQ

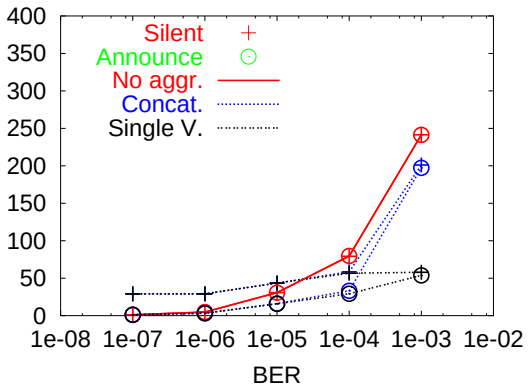

Figure 4: Time to terminate (s), 400 nodes, ARQ

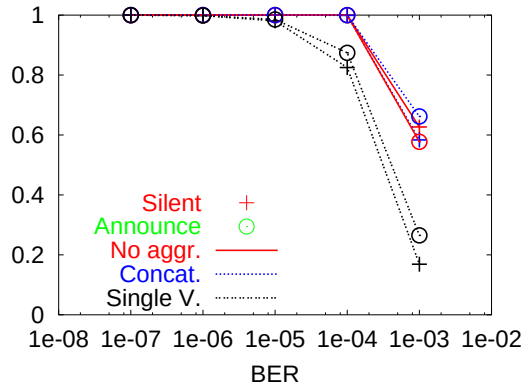

Figure 5: Fraction of decided nodes, 400 nodes, without ARQ

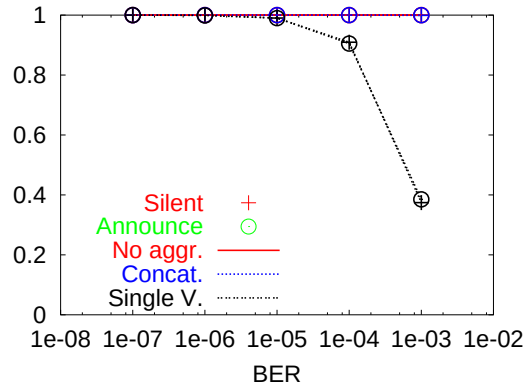

Figure 6: Fraction of decided nodes, 400 nodes, with ARQ

of hops is smaller than 5. This error in the estimated tree depth leads to timeout values (for aggregation and retransmission requests) that are too large. A more sophisticated scheme should be adopted. Still, the variant that does not use aggregation often needs more time – even with the small MAC overhead the large number of messages need time until they are transmitted.

The next (and perhaps most important) question is how aggregation influences the quality of the consensus. The quality of a consensus protocol can be measured in how likely it violates the agreement property. The problem here is that a message might not reach a given node. If a node does not receive the decision, it does not decide. This can happen despite retransmission requests. After three retransmission requests the coordinator terminates the protocol, but some nodes may remain undecided. The fraction of nodes that actually decide gives an impression of the quality of the protocol.[1]

Figures 5 and 6 show that the fraction of decided nodes is different for each variant. The single-value

[1]This is actually a better performance metric than using the more intuitive fraction of incorrectly decided nodes, as this would largely depend on the (relatively small) probability of proposing ABORT and would hence lead to distorted figures – this product of two small probabilities (ABORT probability and the probability that a message of one of the objecting nodes is concerned) is very small and difficult to simulate.

aggregation has the smallest number of decided nodes, because it does not care whether nodes answered or not. The other variants that do care about individual nodes have a very similar performance. The difference between these two variants is not significant up to a BER of 10^{-3}. The difference between the two variants is caused by the way how the tree is built. If the children in the tree do not announce their presence, they will be included in the tree on the first convergecast received from them – this can also be an answer to a retransmission request. If the children have to announce their presence within a certain time limit, then no new children will be included in the tree after that. When the multicast message or their announcement is lost they are not included in the tree – and the next multicast, restricted to the tree, is not forwarded to them. In effect this lowers the number of messages they can potentially receive, lowering the success probability. If the tree is built with more care using Automatic Repeat reQuest (ARQ) the difference diminishes.

3 Future work

The implementation of the 2PC protocol led to some more informal results. Using the simulation we observed that the fraction of decided nodes (the reliability measure) not only depends on the "usual suspects" like number of retransmission requests, reliability of the link layer and reliability of the convergecast structure but also in a subtle way on the timer values.

In general, larger timer values increase the reliability of the protocol but make it more susceptible to errors due to mobility and environmental changes. Furthermore, the increase in the time to terminate may not be acceptable for all applications. A more sophisticated choice of the timer values is thus a natural place to trade off speed versus reliability. Better timer values may also allow to increase the reliability while at the same time lowering the time to terminate. In this sense, the protocols described here are likely to still warrant some further optimizations.

Interestingly, this problem can be formulated in a more abstract way that also implies a possible solution. The number of votes aggregated in a messages constitute the informational value of the message. If a node waits longer it will potentially receive more messages from its children and hence increase the benefit from aggregation. However, waiting is expensive: after some time the gathered information may be outdated or the energy consumed during waiting might be excessive. When should a node stop waiting for answers from its children and send the aggregated value on? This question can be answered using the mathematical theory of optimal stopping. The problem is to find a good stopping rule that can be evaluated on a sensor node.