Consensus using aggregation - a wireless sensor network specific solution", Technical Report TKN-04-004, Telecommunication Networks Group, Technische Universität Berlin, April 2004.

TKN Telecommunication Networks Group

Technical University of Berlin

Telecommunication Networks Group

Consensus using aggregation – a wireless sensor network specific solution

Andreas Köpke, Holger Karl and Adam Wolisz
{koepke | karl | wolisz }@tkn.tu-berlin.de

Berlin, April 14, 2004

TKN Technical Report TKN-04-004

TKN Technical Reports Series Editor: Prof. Dr.-Ing. Adam Wolisz

Abstract. In distributed systems like Wireless Sensor Networks (WSNs) nodes often have to agree on actions or values. This is commonly called the consensus problem. To achieve such a consensus traditional protocols exchange a lot of message, which requires a lot of energy. To solve the consensus under WSN-typical constraints, protocols should reduce the number of necessary messages. One technique to reduce message overhead is aggregation, but it increases the susceptibility of protocol messages to channel errors.

In this paper, we describe how to probabilistically solve the consensus problem using aggregation and study the inherent tradeoffs between Bit Error Rate (BER), time to terminate, energy consumption and failure probability. Comparing the proposed protocol to a solution that does not use aggregation, the amount of energy spent is reduced (up to 60%) without increasing the failure probability.

1 Introduction

Consensus is an important problem in distributed systems, such as 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. Things become even more interesting when more than one motor controls a blind, enhancing the robustness of the mechanical system. The motors should agree on the direction, either all should open the blind or close it. 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.

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

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 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.

The paper is structured as follows. In Sec. 2 we introduce the consensus problem and explain it using the Two-Phase commit (2PC) protocol that solves such a consensus problem. The protocol can be implemented in several ways. Possible design options are discussed in Sec. 3. The parameters used for the simulation are explained in Sec. 4. Then we present the results we obtained for the different versions and conclude our work. Furthermore, an outlook on future work is given in Sec. 7.

2 Consensus

One protocol that solves a particular instance of the consensus problem is the 2PC protocol [1]. It ensures that e.g. all actuators perform an action or none does, or that all nodes activate a software patch or none does.

To achieve this consensus, the 2PC protocol proceeds in two rounds. In the first round the central coordinator sends a *proposal* (COMMIT) to all participants using a multicast. Within a certain time limit, all participants send their proposition (COMMIT or ABORT) back to the coordinator; this is called a convergecast. If the coordinator receives only COMMIT messages, it prepares to decide COMMIT. In case of only a single ABORT it prepares to decide for ABORT. It then multicasts the envisioned *decision* to the participants and decides. The participants receive the decision within a certain time, decide accordingly and send an acknowledgement to the coordinator.

This protocol ensures that – if no nodes fail, no messages are lost or delayed for an unknown time and if task execution time on the nodes is smaller than a known upper bound – the participants reach a consensus. They agree on the value that the coordinator sent to all participants as a decision.

The 2PC protocol is a specific protocol that we picked as an example for the evaluation. But the consensus problem is actually much broader. Whenever a protocol ensures three formal properties, it solves a consensus problem [1,2]:

Validity If a node decides for v, then v was proposed by some node.
Termination Every correct node eventually decides.
Agreement No two nodes (correct or faulty) decide differently.

The *Validity* property is needed to rule out trivial solutions, where the value for which all nodes decide never changes. It is possible to relax the agreement property and to require only correct nodes to decide. This is useful when nodes can behave arbitrarily or send wrong messages (in a Byzantine fault model, e.g.). The protocol

discussed here is designed for a simple crash fault model, where a failing node stops all forms of operation. However, we assume that nodes (esp. actuators) fail rarely. Other cases require more complex protocols and an adaptation of them for WSNs is planned for future work.

In a real system, it is very hard (if at all possible) to ensure that the task execution time and the time it takes to transmit a message is bounded by some known constant. We do not assume that this is attempted in a WSN. Instead, we assume the system to be *asynchronous* in the distributed systems sense: task execution times and message transmission times are not bounded.

In such a realistic system, the consensus problem can – strictly speaking – not be solved [3]. The solutions for distributed databases minimize the probability that the properties are violated by considering failure recovery (i.e., maintenance). In a WSN, however, maintenance is usually not feasible. As a result, only a probabilistic solution remains feasible where the protocol can occasionally fail but the probability of a failure should be minimized. Such a protocol may require large amounts of energy and a long time to reach a decision. A tradeoff between these goals (energy spent, time to reach a decision and failure probability) exists for each application individually. The following sections outlines possible design options that achieve different tradeoffs between these goals.

3 Design options for 2PC in WSN

The particular tradeoffs between the conflicting goals are specific for the concrete application that has to reach a consensus. In one application it may be reasonable to reach a consensus very energy-efficiently even if this implies that some nodes do not take part. In another application (e.g. software update) saving energy is not the ultimate goal; it may be more important to be as sure as possible that all nodes get the patch and activate it. It is not possible to predict the needs for every application that attempts to reach a consensus. Hence, several protocols are desirable with different tradeoffs.

3.1 Basic options without aggregation

Even a basic 2-phase commit protocol has a number of degrees of freedom as soon as we have to pay attention to the details of the implementation of such a protocol in a multi-hop environment. To describe the possible design choices we will discuss them separately for each step given in the overview in Sec. 2.

Two-phase commit. First, the central controller sends a proposal to all participants. Actually, there is no need for a central controller. The 2PC protocol can be implemented in a centralized or decentralized fashion. A decentralized protocol is more robust when nodes fail often, but requires a higher number of messages. A centralized protocol is more sensitive, it introduces a single point of failure. However, we assume that nodes actually fail relatively rarely and that a distributed protocol is an overkill: the higher number of messages make it less feasible in a WSN where communication is the largest consumer of energy.

In a centralized protocol, one node has to be the coordinator. The precise choice of a coordinator for a particular application is not within the scope of this paper (leader election in multihop networks is treated e.g. in [4]). We assume that for the applications mentioned in the introduction such a coordinator is given, for instance a special node that serves as a gateway.

Communication infrastructure: Trees or clusters. The design of a distributed protocol is considerably simplified if a "flat" network view can be assumed, where every node is a one-hop neighbor of another node – in effect, this presupposes a functional, identity-based routing layer. While such routing layers exist, they are not necessarily suitable for ad hoc networks (identity issues, scalability problems) and their superimposed topology is not necessarily well adapted to the distributed algorithm. Hence, we opted for constructing the necessary communication infrastructure in the protocol itself.

The coordinator's proposal must be forwarded to the participants. In our implementation we used plain flooding, where every node repeats the message exactly once. During the flooding of the proposal, the nodes arrange themselves in a tree rooted in the coordinator and simultaneously learn about their parents in the tree. The nodes use this information to convergecast their answers back to the coordinator. This operation is the main task of the tree.

In the non-aggregating case of the protocol, it is not particularly important for a node in the tree to know about its children or about the depth of the locally rooted tree. This information is only relevant in the aggregating case (as nodes have to wait for messages from all their children for aggregation to make sense); how to determine it is hence described in Section 3.2.

As a by-product the tree can also reduce the number of messages sent to distribute the decision (compared to plain flooding). Other options are to reduce the set of forwarding nodes by clustering the network – then only gateways and clusterheads have to forward the message. However, the use of clustering is completely independent and not considered in this paper.

Node failures and message losses. When the energy resources of a node are depleted, it fails. Although we assume that node failures are rare, they can occur and a consensus protocol has to cope with them. When the coordinator detects a node failure, the node is excluded and the remaining nodes proceed with the protocol. The same applies when the link to a node is broken and the node cannot be reached for a certain amount of time.

To detect node failures, one can use a pro-active approach or a re-active approach. The basic idea of a pro-active approach is that every node sends a message ("ping") once in a while; for possible implementations, see e.g. [5,6,7].

In our implementation, we did not rely on a pro-active failure detector, but rather a re-active one, better known as timeout. The 2PC protocol requires every node to reply with either ABORT or COMMIT. If this message does not arrive at the coordinator within a certain time, the coordinator asks the node to retransmit its message using an ID-centric routing. We assume that such an ID-centric routing (e.g. AODV, DSR or comparable) is present.

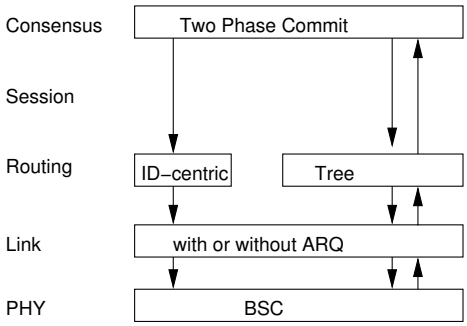

Fig. 1. Structure without aggregation

The choices for timeouts and the number of retransmissions imply a specific choice in the tradeoff between the spent energy, the time the protocol needs to reach a decision and protocol failure probability. A higher number of allowable retransmission requests leads to a higher energy consumption and a longer time to terminate, but lowers the failure probability. Shorter timeouts can result in unnecessary energy expenditure for superfluous retransmission requests. Also, nodes are (erroneously) suspected to have failed early and this increases the protocol failure probability.

Reliable link layer. Radio transmissions are error prone and a hop-by-hop error recovery strategy can be useful. Essentially there are two ways to protect a packet against message losses using a Forwad Error Correction (FEC) or an Automatic Repeat reQuest (ARQ) protocol. Both variants increase the overhead per message and this may outweigh the benefits. Therefore, simulations were done with and without ARQ.

In summary, Fig. 1 shows the interactions between the different parts necessary for the 2PC protocol without aggregation.

3.2 Integrating aggregation

Aggregation can be introduced into the structure as shown in Fig. 2. This has several ramifications also on other parts and we discuss them in this section.

Aggregation. The aggregation part introduces several functionalities in all nodes of the network (not only those that participate in the consensus itself). First of all it knows how to combine multiple values into a single one. One possible aggregation simply computes the minimum if ABORT $\equiv 0$ and COMMIT $\equiv 1$. We call this variant single-value aggregation. An important consequence is that this precludes retransmission requests, because the central node does not know whether an answer form a certain node was included or not. If the aggregation part also ensures that the answers of all children are included in the message then essentially the TREE-COMMIT protocol [8] is implemented. In our performance evaluation the aggregation part is not responsible for the reliability of the

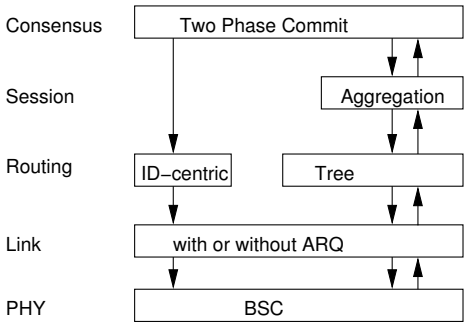

Fig. 2. Structure with aggregation

transmission; this is one possible variation. The aggregation into a single value without enhanced reliability is very energy-efficient: the packet size remains constant irrespective of the depth of the node in the convergecast tree. Another variant, which does allow retransmission requests, is to simply concatenate the values and the IDs of all children into a single message.

In order to collect the messages from all children the aggregation part has to know about the children, an information available from the tree part. But how long should it wait for answers from the children? A message loss could block the aggregation part and the messages that already arrived are not processed. Therefore, the aggregation block also maintains a timer. If some children fail to answer it just proceeds and sends the aggregated values on. Retransmission requests are left over to the part that uses aggregation. If the aggregation part receives a retransmission request, it tries to answer it using the information in its cached, aggregated message.

Locally determining waiting time. For aggregation to make sense, nodes have to wait for messages from all (or at least most of) their children. The waiting time is particularly important for the leafs in the tree as they have to start the actual convergecast back to the coordinator. It depends on how the tree is built, more precisely, how children in the tree behave with respect to their parents.

The children have three options how to behave with respect to their parents: They can simply do nothing, they can explicitly inform a parent node that it has a new child, or it can inform all nodes in radio vicinity about the newly formed parent-child relationship. This last option is potentially useful for tree repair (e.g., due to mobility) but is not pursued further in this paper. The first and second case are called “silent” and “announce” in the results section.[1]

[1] For completeness’ sake, the distinction between silent and announced tree building is also used for the non-aggregating case although it makes little difference there. The most important differences is obviously in the number of sent bits or messages and also in the fraction of decided nodes.

Depending on whether the tree is built in a silent or announced fashion, a node can use resulting information to decide whether it is a leaf. In any case, a node can make this decision only after waiting for a certain time in which it expects messages from its children. If children are silent, this information is implicitly included only in actual data messages (here, the convergecast) and the necessary waiting time hence depends on the length of the longest path to any child in the sub-tree of that particular node. In the initial phase, some estimate of this value is required; we use

$$(\mathrm{MAX\_HOP\_COUNT} - \mathrm{hops\_to\_coordinator}) \cdot \mathrm{max\_time\_per\_hop}\ .$$

After the first convergecast, these estimates can be adapted.

If, on the other hand, children explicitly announce their presence to their parents, this waiting time can be shortened considerably as only messages from the local neighborhood have to be waited for. The evident consequences of these options are differences in the time to terminate for the procedure, depending on how and when the convergecast is started by the children.

4 Evaluation

In order to obtain some understanding of the quantitative tradeoffs involved in these protocol variants, we programmed a simulation that reflects the most crucial aspects of these algorithms. It is based on the OMNeT++ simulation toolkit [9] and AKAROA [10] as a statistical evaluation tool.

4.1 Basic parameters

A number of parameters and protocol behaviors have to be fixed to allow for a simulation to run. These are:

Retransmission requests from coordinator If three such requests remain unanswered, the coordinator assumes that the node is not operational anymore and proceeds.

Timeout values The timeout value is the same for all 2PC implementations, and we set it to 40 s. The value is computed on the basis that we allow packets to travel at most 30 hops and the reliable link layer to transmit packets in less than 0.5 s. To this value a 10 s margin is added, to take some computational overhead into account.

Routing overhead For our simulations we assume an ideal routing protocol with no overhead. However, this overhead is not necessarily negligible and implementations of 2PC that do not use it can be beneficial. This has to be evaluated more closely in the future.

Coordinator rotation We rotated the coordinator node for two reasons. One reason is that we were interested in an average behavior over all possible choices. The second reason is that, whenever possible, the coordinator should be rotated lest the energy resources of a single node are exploited.

Probability to propose ABORT This probability is chosen as 10^{-3}.

4.2 Distribution of participants

The participants can be a subset of the nodes in the whole network or all nodes in the network. In our simulations we assume that the participants are all nodes in the network. We evaluated the protocol with different network sizes, ranging from 100 to 400 nodes. The nodes are distributed randomly in a plane using a two-dimensional uniform distribution. The transmission range was adapted to keep the average number of neighbors fixed at 14, making the network connected with high probability [11] and eliminating influences due to a varying number of neighbors. To keep the number of neighbors independent of the network size, one has to correct the edge effect.[2] For a 100 node topology the average number of neighbors is around twelve, while for a 400 node topology the average number of neighbors is around 14. To eliminate this effect and its influences, we used the toroidal distance [12] between nodes. With this distance nodes at opposite edges of the plane can communicate. This leads to a constant average number of neighbors, independent of the total number of nodes.

4.3 Link layer, physical layer and channel model

All forwarding takes place between neighboring nodes. In the simulation, this level was modeled in a very rudimentary way. The links are modeled as point to point links for unicast messages. For messages that have multiple recipients the broadcast nature of the radio medium was taken into account: a single send reaches all nodes in the neighborhood.

To transport the messages over the wireless medium they are put into packets. The overhead necessary for the PHY and MAC layer is small (88 bits), slightly smaller than the overhead of the CSMA type MAC protocol of the IEEE 802.15.4 standard. This takes only the minimal overhead into account, a possible additional control overhead due to e.g. the RTS-CTS exchange as in IEEE 802.11 is neglected.

Possible options for the link layer include an unreliable one (with error recovery completely left to the consensus protocol itself) and a link layer that uses an ARQ protocol (with up to two retransmissions) for unicast packets. In this case, only the convergecast is made more reliable, messages can still get lost during the multicast phase. Therefore, retransmission requests from the coordinator remain necessary.

The physical channel over which packets are transmitted is modeled with a Binary Symmetric Channel (BSC). This channel introduces independent bit errors in the packet. Although this model is often used in simulations and analytical evaluations, measurements [13] show that it is a special case. Future evaluations will use more than one channel model.

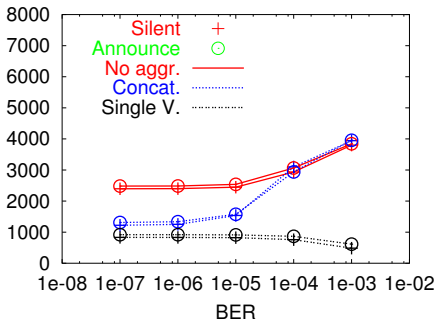

Fig. 3. Sent bits per node, 400 nodes, without ARQ

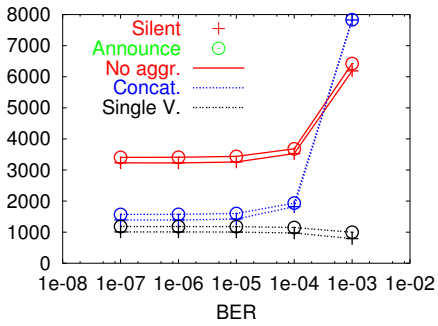

Fig. 4. Sent bits per node, 400 nodes, with ARQ

5 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 $\frac{1}{1.01}\bar{x} \leq \mu \leq 1.01\bar{x}$).

The first metric to look at is the number of sent bits. Fig. 3 and 4 show (with or without ARQ for the unicast communication) the average number of bits sent by each node per consensus attempt as a function of the BER for a sensor network consisting of 400 nodes, all taking part in the 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 explanation is that the concatenated values require larger packets (each value adds 24 bits to the packet) which are more susceptible to losses. Also, the retransmission requests are made for each missing value separately, resulting in a high number of packets sent for retransmission requests. This remains true although the aggregation seeks to an-

[2] This effect is introduced when the Euclidean distance is used to check whether two nodes can reach each other: the nodes at the edge of the plane have fewer neighbors than those in the middle of the plane. As a result, the average number of neighbors in topologies with few nodes is smaller than in topologies with a large number of nodes.

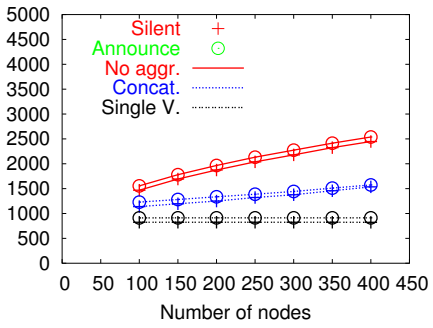

Fig. 5. Sent bits per node, BER $1e^{-5}$, without ARQ

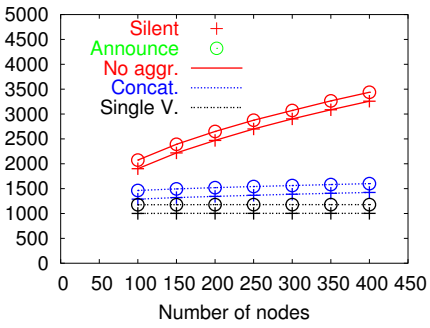

Fig. 6. Sent bits per node, BER $1e^{-5}$, with ARQ

swer retransmission requests as early as possible and if it can do so, the retransmission request is not forwarded.

Because the longer packets of the concatenating variant are more susceptible to losses, it may make sense to protect them with an ARQ scheme and put more effort into them. Interestingly, for small BER the additional cost is small for the aggregating variants, while it results in a high penalty for the non-aggregating variant. However, for high BERs the concatenating variant is still on the loosing end. This suggests that an adaptive protection should be used. One possibility is to avoid bad links altogether, or to add more error protection. However, information on the link quality may be hard to obtain whereas it is fairly easy to have a notion of an important packet: a packet that carries the aggregated information from many nodes should be protected in a more sophisticated way. Such an "aggregation-aware link layer" is suggested in [14].

Using ARQ has another benefit if we look at Figures 5 and 6 (which show sent bits over varying network size): it lowers the influence of the network size.

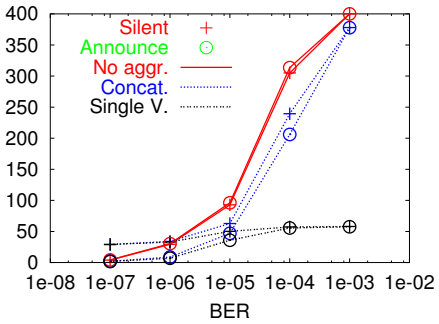

Fig. 7. Time to terminate (s), 400 nodes, without ARQ

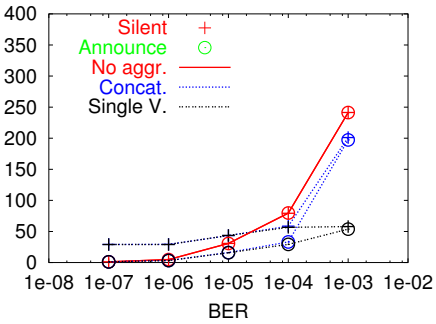

Fig. 8. Time to terminate (s), 400 nodes, ARQ

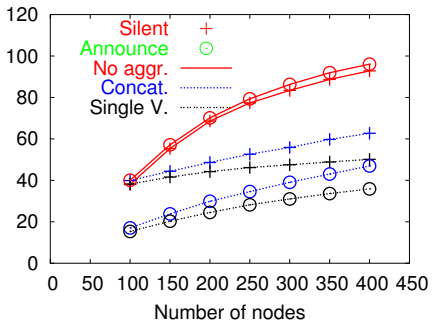

Fig. 9. Time to terminate (s), BER $1e^{-5}$, without ARQ

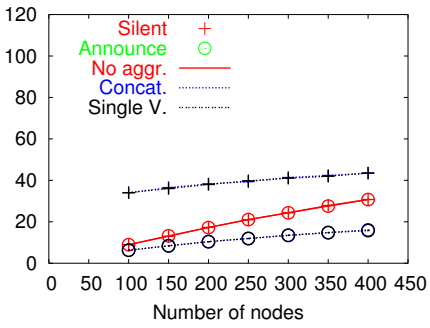

Fig. 10. Time to terminate (s), BER $1e^{-5}$, with ARQ

Another question is whether the use of aggregation has an influence on the time it takes for the algorithm to terminate. Figures 7 to 10 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 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 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.[3]

Figures 11 and 12 show that the fraction of decided nodes is different for each variant. The single-value 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 $1e^{-3}$. 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

[3] 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.

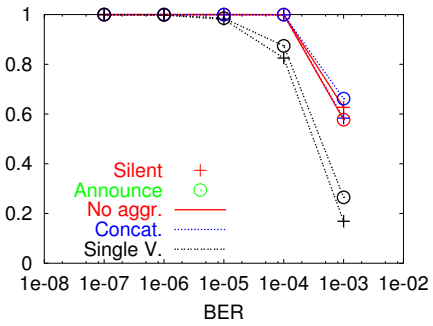

Fig. 11. Percentage of decided nodes, 400 nodes, without ARQ

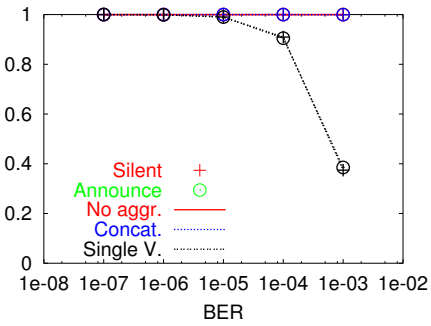

Fig. 12. Percentage of decided nodes, 400 nodes, with ARQ

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 ARQ the difference diminishes.

6 Conclusion

In this paper, we have shown that it is indeed possible to perform the comparably expensive operation of a consensus even under the severe resource constraints of a wireless sensor network. The most interesting conclusion from our results is that the choice of protocol considerably depends on the present bit error rate. In scenarios with a low to medium BER, interestingly enough, the concatenation protocol turns out to be the preferable choice. In these circumstances, single-value aggregation suffers considerably from the absence of information which node's proposal is missing; the savings in message length do not compensate for this shortcoming – only at BERs of about 10^{-7} do the advantages of single-value aggregation prevail. Non-aggregating protocols, on the other hand, have too high a message overhead to be competitive.

This conclusion changes for high BERs: Here, non-aggregating protocols are best suited and tend to send the smallest number of bits. This is mostly due to some problems of concatenating protocols, where important messages can get lost, resulting in a high overhead to repair this loss – overall more than the non-aggregating protocol requires. Moreover, the long packets of the concatenating protocols are not ideal for such circumstances anyway.

As a consequence, reliable link layers are highly recommended in most circumstances. However, it is beneficial to make the ARQ protocol conscious of the relevance of a given packet. This should considerably improve performance even further.

7 Future work

The evaluation in this paper can be extended in several directions. The first extension is to evaluate the protocol performance using more realistic and a wider variety of channel models. The channel models should also take correlated channel conditions into account. It may well happen that a certain node is hardly reachable. A comparable problem occurs when mobility is taken into account. Mobility leads to correlated breaking of links – in contrast to the usual assumptions in distributed systems textbooks where at best stochastically independent link failures/repairs are considered. With the approach used here, mobility causes a major problem as it destroys the tree used for aggregation. It is not clear how to recover from such problems and whether a pro- or re-active approach should be chosen. Another, simpler solution is to lower the time until the algorithm terminates, because this lets nodes less time to become unreachable due to their movement. The time an algorithm needs to terminate is closely connected with the values chosen for the timers, but choosing good values is tricky because of the inherent tradeoff with energy – too short timeouts cause unnecessary retransmission requests and increase the protocol failure probability. The protocol failure probability is also connected with retransmission requests. Currently, they are initiated by the central controller and forwarded to the node in question using an ID-centric routing. However, the TREE-COMMIT protocol uses parents in the convergecast tree to request retransmissions. The advantage is that the messages travel only short ways, but this also makes the computation of timeout values more difficult. For a reasonable estimate the parents need information about the ID of their children – which is fairly easy to get – but also the depth of the tree that starts at a particular child. It remains to be seen whether the energy necessary to obtain this information pays off in lower total energy spent.

References

1. Coulouris, G., Dollimore, J., Kindberg, T.: Distributed Systems – Concepts and Design. 3. edn. Addison-Wesley, Harlow (2001)
2. Guerraoui, R., Hurfin, M., Mostéfaoui, A., Oliveira, R., Raynal, M., Schiper, A.: Consensus in asynchronous distributed systems: A concise guided tour. In Krakowiak, S., Shrivastava, S., eds.: Distributed Systems. Volume 1752 of Lecture Notes in Computer Science. Springer Verlag, Berlin Heidelberg (2000) 33–47
3. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. Journal of the ACM **32** (1985) 374–382
4. Malpani, N., Welch, J.L., Vaidya, N.H.: Leader election algorithms for mobile ad hoc networks. In: Proc. 4th Intl. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, Boston, MA (2000)
5. Aguilera, M.K., Chen, W., Toueg, S.: Heartbeat: A timeout-free failure detector for quiescent reliable communication. In: Workshop on Distributed Algorithms. (1997) 126–140
6. van Reenesse, R., Minsky, Y., Hayden, M.: A gossip style failure detection service. In Davies, N., Raymond, K., Seitz, J., eds.: Middleware. International Conference on Distributed Systems Platforms and Open Distributed Processing, IFIP, Springer Verlag (1998) 55–70
7. Gupta, I., Chandra, T.D., Goldszmidt, G.S.: On scalable and efficient distributed failure detectors. In: 20th Symposium on Principles of Distributed Computing, Newport (2001)

8. Segall, A., Wolfson, O.: Transaction commitment at minimal communication cost. In: Proceedings of the sixth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, ACM Press (1987) 112–118
9. Varga, A.: Omnet++ discrete event simulation system. http://www.omnetpp.org/index.php (2003)
10. Ewing, G., Pawlikowski, K., McNickle, D.: Akaroa2: Exploiting network computing by distributing stochastic simulation. In: Proc. European Simulation Multiconference ESM'99, Warsaw, International Society for Computer Simulation (1999) 175–181
11. Xue, F., Kumar, P.R.: The number of neighbors needed for connectivity of wireless networks. Wireless Networks (2003) accepted for publication.
12. Bettstetter, C.: On the minimum node degree and connectivity of a wireless multihop network. In: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing, ACM Press (2002) 80–91
13. Köpke, A., Willig, A., Karl, H.: Chaotic maps as parsimonious bit error models of wireless channels. In: Proc. of IEEE INFOCOM, San Francisco, USA (2003)
14. Karl, H., Löbbers, M., Nieberg, T.: A data aggregation framework for wireless sensor networks. In: Proc. Dutch Technology Foundation ProRISC Workshop on Circuits, Systems and Signal Processing. (2003) http://www.stw.nl/prorisc/. To appear.

A Additional results

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	241.0	400.0	241.0	400.0
Concat.	201.0	378.0	197.0	378.0
Aggr.	57.9	58.0	53.7	57.6

Table 1. Time to terminate (in s), 400 nodes, BER 1e-03

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	6189	3801	6423	3843
Concat.	7826	3944	7832	3955
Aggr.	792	485	998	613

Table 2. Sent bits per node, 400 nodes, BER 1e-03

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	39.6	18.3	42.1	19.1
Concat.	37.9	17.8	37.5	18.3
Aggr.	4.66	2.33	6.96	3.52

Table 3. Sent packets per node, 400 nodes, BER 1e-03

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	1.000	0.627	1.000	0.577
Concat.	0.999	0.584	1.000	0.662
Aggr.	0.374	0.169	0.385	0.265

Table 4. Fraction of decided nodes, 400 nodes, BER 1e-03

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	79.2	305.0	79.7	314.0
Concat.	58.5	239.0	33.2	206.0
Aggr.	56.7	57.3	29.2	55.6

Table 5. Time to terminate (in s), 400 nodes, BER 1e-04

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	3528	2935	3681	3064
Concat.	1821	3099	1934	2937
Aggr.	974	758	1150	866

Table 6. Sent bits per node, 400 nodes, BER 1e-04

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	23.0	14.1	24.8	15.3
Concat.	9.33	11.9	11.1	11.1
Aggr.	5.79	3.65	7.80	4.74

Table 7. Sent packets per node, 400 nodes, BER 1e-04

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	1.00	1.000	1.00	1.000
Concat.	1.00	0.999	1.00	1.000
Aggr.	0.909	0.825	0.905	0.874

Table 8. Fraction of decided nodes, 400 nodes, BER 1e-04

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	30.8	92.8	30.7	96.0
Concat.	43.4	62.7	16.1	46.9
Aggr.	43.6	50.1	15.7	35.9

Table 9. Time to terminate (in s), 400 nodes, BER 1e-05

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	3258	2447	3438	2538
Concat.	1423	1536	1600	1574
Aggr.	1003	823	1179	913

Table 10. Sent bits per node, 400 nodes, BER 1e-05

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	21.2	11.8	23.2	12.8
Concat.	7.17	6.00	9.18	6.70
Aggr.	5.97	3.96	7.97	4.96

Table 11. Sent packets per node, 400 nodes, BER 1e-05

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	1.00	1.00	1.00	1.00
Concat.	1.00	1.00	1.00	1.00
Aggr.	0.990	0.981	0.990	0.985

Table 12. Fraction of decided nodes, 400 nodes, BER 1e-05

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	4.76	29.3	4.74	30.3
Concat.	29.2	33.9	3.23	8.79
Aggr.	29.0	33.0	3.13	6.85

Table 13. Time to terminate (in s), 400 nodes, BER 1e-06

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	3227	2400	3409	2484
Concat.	1391	1250	1573	1336
Aggr.	1007	830	1182	918

Table 14. Sent bits per node, 400 nodes, BER 1e-06

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	21.0	11.5	23.0	12.5
Concat.	6.99	5.09	9.00	6.07
Aggr.	5.99	3.99	7.98	4.99

Table 15. Sent packets per node, 400 nodes, BER 1e-06

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	1.00	1.00	1.00	1.00
Concat.	1.00	1.00	1.00	1.00
Aggr.	1.00	0.998	0.999	0.999

Table 16. Fraction of decided nodes, 400 nodes, BER 1e-06

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	1.17	3.95	1.16	4.11
Concat.	29.1	29.1	1.20	2.01
Aggr.	29.0	29.0	1.07	1.69

Table 17. Time to terminate (in s), 400 nodes, BER 1e-07

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	3229	2395	3405	2483
Concat.	1393	1221	1571	1309
Aggr.	1007	831	1182	919

Table 18. Sent bits per node, 400 nodes, BER 1e-07

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	21.0	11.5	23.0	12.5
Concat.	6.99	5.00	8.98	6.00
Aggr.	5.99	4.00	7.99	4.99

Table 19. Sent packets per node, 400 nodes, BER 1e-07

	Silent		Announce	
Aggr.type	ARQ	No ARQ	ARQ	No ARQ
No aggr.	1.00	1.00	1.00	1.00
Concat.	1.00	1.00	1.00	1.00
Aggr.	1.00	1.00	1.00	1.000

Table 20. Fraction of decided nodes, 400 nodes, BER 1e-07