

Implementing Agreement Protocols in Sensor Networks

Andreas Achtzehn
Department of Electrical Engineering
RWTH Aachen University
D-52056 Aachen, Germany
andreas.achtzehn@rwth-aachen.de

Zinaida Benenson Christian Rohner
Department of Information Technology
Uppsala University
Box 337, SE-751 05 Uppsala, Sweden
{zina,chrohner}@it.uu.se

Abstract— We investigate application of agreement protocols tolerating malicious failures in sensor networks. We identify several scenarios where agreement can be used, and report on our experience with implementing an agreement protocol.

I. INTRODUCTION

A. Agreement in Distributed Systems

Agreement protocols come into the scene when the nodes in a distributed system need to reach agreement on data in a decentralized manner. For example, nodes in a replicated system may need to agree on the messages or values they receive from some system components in order to take identical steps in their program. Another prominent agreement task is clock synchronization.

Agreement becomes non-trivial in presence of node and communication faults, such as lost messages, node crashes, or confusing information (e.g., a faulty node may send different clock values to the protocol participants). Fault-tolerant and secure systems extensively use agreement, MAFT [1] and Rampart [2] being among the first such applications. Problem of reaching agreement in presence of faults remains an active research area for more than 25 years starting with the seminal paper [3].

Agreement protocols usually work according to the following scheme. Each node receives an input value (e.g., its own clock reading). Then the nodes exchange their values, and the values they received from other participants. In order to cope with failures, several rounds of communication are necessary [4].

B. Contribution and Outline

The goal of this work is to investigate the applicability of agreement protocols in sensor networks from the practical point of view. To our knowledge, this is the first attempt of this kind, although many secure protocols for

sensor networks rely on agreement, see, e.g., [5], [6], [7]. The kind of used agreement is usually not specified, but as a building block for secure sensor networks, we feel that it should also be implemented securely. We further discuss the uses of agreement protocols in Section II.

Our ultimate goal is to implement a secure and energy-efficient agreement protocol for sensor networks. However, in order to get the feeling for this problem, we decided to consider a “toy” application with a relatively benign adversary, and to choose the appropriate agreement protocol methodically, see Section III.

Finally, in Section IV, we report on our implementation and its resource consumption, and conclude in Section V.

II. AGREEMENT IN SENSOR NETWORKS

A. Scenarios

In classical agreement protocols, all nodes need to communicate with each other. Clearly, running a global agreement protocol in a sensor network consisting of hundreds of nodes is infeasible due to high communication costs.

However, we identified several scenarios where the classical decentralized agreement may be useful. All these scenarios assume that the agreement is localized, i.e., only a small part of sensor nodes from a particular region needs to reach agreement.

Aggregation: If decentralized, secure and fault-tolerant aggregation of sensor readings is to be implemented, there seems to be no alternative to the agreement protocols, as the sensors have to agree on the aggregation outcome in presence of arbitrary, perhaps even malicious, faults.

Group management: Leader election and group membership protocols from classical distributed systems could also be used in sensor networks for secure and

fault-tolerant decentralized selection of cluster heads or aggregators, or for exclusion of faulty nodes.

Coordinated action: Agreement on the event detection is needed in case sensor nodes need to take a further action based on the event detection. For example, after an agreement on the target detection, the next collaborative task is to localize the target [8]. Of course, this task can also be solved in a centralized manner where the sensors report target detection to the base station, and the base station then disseminates the next task of target localization. However, if the base station is far away from the observed region, making a local fault-tolerant decision on the presence of the target would be less expensive than having each sensor to notify the base station independently.

Agreement is especially useful if the further action is to activate an actuator. Consider a field irrigation system where each sensor node can turn on one irrigation unit. Without agreement, a sensor node with a faulty humidity sensor may perceive the ground to be too dry and start its irrigation actuator. Using agreement, this situation can be prevented.

B. The Centralized Alternative

Usually, localized decisions in sensor networks are made in the centralized manner. The nodes report their (possibly, preprocessed) measurements to an aggregator node, e.g., a cluster head, which takes the decision based on the received values, and disseminates it to the nodes. The advantage of this architecture is the low communication overhead $O(n)$ for the group of n sensors. Using agreement, at least $O(n^2)$ messages are needed.

On the other hand, apart from the usual argument that a centralized solution is susceptible to the failure of the central component, decentralized protocols also significantly increase robustness of the decision making in case of communication failures which are very common in sensor networks. For example, in Fig. 1, a majority voting should be performed based on local decisions of 4 sensor nodes. In case of the centralized solution, two link failures preclude the decision, whereas an appropriate agreement protocol still works.

C. Related Work

In [8] the authors investigate by analysis and simulation the performance of interactive consistency algorithms for target detection in terms of detection accuracy. They use the algorithm by Lamport et al. [9] which

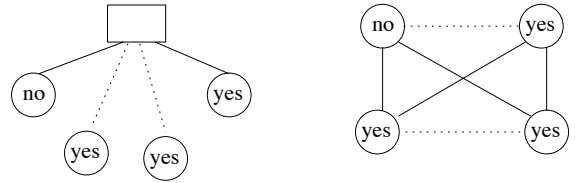


Fig. 1. In centralized agreement, two link failures preclude the majority decision. In a decentralized protocol, the decision “yes” still can be made.

assumes fully reliably communication, and do not consider communication costs. Some of the used scenarios are unrealistic in terms of resource consumption, e.g., 48 sensor nodes reaching agreement in presence of 8 faulty nodes. Implementing such a scenario in a real sensor network would be much too expensive in terms of energy, as our experiment showed. Moreover, the implementation would have to rely on the unrealistic assumption of fully reliable communication. To our knowledge, nobody considered implementing a fault-tolerant agreement protocol in sensor networks.

III. SYSTEM SPECIFICATION

In this section, we describe our toy application, and the choice of the appropriate agreement protocol.

A. *Hitzefrei*

The German term *Hitzefrei* (HF in the following)¹ is used to describe the following situation: If in summer the air temperature gets too high (around 30 degrees Celsius), the schools remain closed, such that the school children have a free day because of the heat.

In our scenario, the HF application monitors the air temperature in offices in a large building using a sensor network. There are several sensor nodes in each office. If the temperature gets above a certain threshold (*HF threshold*), the system uses agreement to decide that the *HF event* occurred, and notifies the employee, as well as all doors in the building, that the employee may leave.

The application should meet two goals. Firstly, an attacker (the employee) should not be able to fool the system into deciding that the HF event happened in case the air temperature is below the HF threshold. On the other hand, if the HF event happens, the system should not miss it.

We used the well known *threat tree* approach [10] to identify how the system could fail to meet its goals. In Figure 2, threat trees for these two goals are presented.

¹“Hitze” means “heat”, “frei” means “free”.

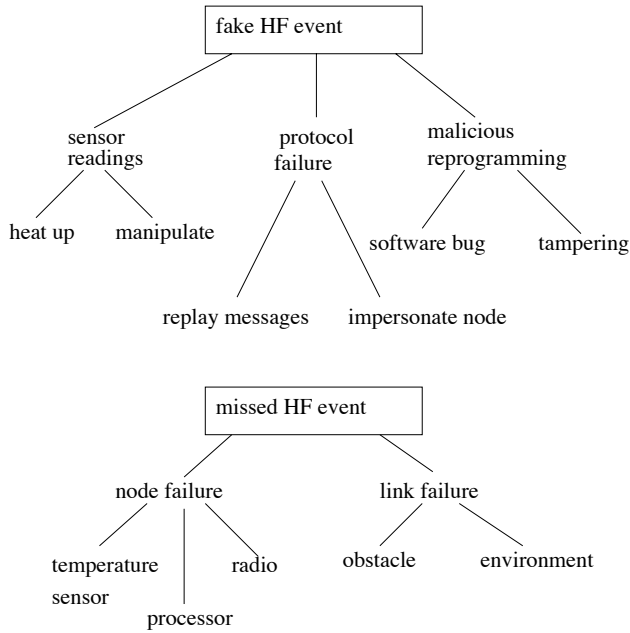


Fig. 2. Threat trees for the HF application. The goal is to recognize the HF threshold correctly while being well protected from malicious attacks.

To fake a HF event, the employee may attack any part of the system. He can try to directly manipulate *sensor readings*. One of the most likely attacks is heating the node up using, e.g., a lighter. The attacker can also try to exploit *protocol failures*, meaning logical mistakes in the protocol. *Malicious reprogramming* implies that the adversary gains full control over the node. The worst case attack would be finding *software bugs* in the node’s operating system, or in the program. This may allow the adversary to take over all sensors in the system. The second type of reprogramming attacks is known as *node capture*, meaning gaining full control over a sensor node by a physical attack.

Apart from being secure with respect to the attacking employees, Hitzefrei should be fault-tolerant with respect to HF events recognition. Failures which can happen to the sensor nodes are plentiful. Wrong temperature can be measured due to a failed sensor, node’s radio may fail, such that the communication is impossible, or due to some memory error the saved correct measurement, or, indeed, node’s program, can be changed unpredictably. Link failures can also happen in the office due to some environmental changes or to obstacles, in case something or somebody accidentally shields a node from the other nodes.

B. System Requirements

According to the famous result [11], designing a deterministic agreement protocol is impossible for asynchronous communication channels. Thus, we use timeouts as a parameter of the system in order to decide on the message processing times and on the message arrivals. This allows us to use agreement protocols designed for synchronous networks, where the upper bounds on node speed and message arrival are known.

Agreement protocols use either “written messages” or “oral messages” [9]. The former can be realized by digital signatures², and are too inefficient for the computationally weak sensor nodes. The benefit of a written message is that it can be verified by any node.

Using oral messages, any node can determine reliably from which node the message came, but is unable to prove the message origin to somebody else. Oral messages can be realized using message authentication codes, which are feasible for sensor nodes.

In sensor networks, link failures are reported to be one of the most common failures, followed by malfunctioning sensing units [13]. Therefore, we assume that the communication links are unreliable, and the nodes can measure wrong values due to failed temperature sensors.

C. The Adversary

As already mentioned in the introduction, we consider a relatively benign adversary. In our building, each office is occupied by a single employee. The employees are interested in attacking the system, such that they can get a free day, but they do not trust each other, so they do not cooperate in the attack. We also assume that the sensors are installed far apart from each other, such that simultaneous physical manipulation (such as heating up) of two or more sensors by a single attacker is impossible.

We also assume the sensor network to be robust against malicious reprogramming. Defenses against software bugs are presented, e.g., in [14], and countermeasures against node capture attacks are discussed in [15].

D. Interactive Consistency

We decided to implement an *Interactive Consistency (IC)* algorithm in order to enable the sensor nodes to reach agreement on the room temperature.

Consider a set of n nodes $\{p_1, \dots, p_n\}$, each having an initial value x_i . Some nodes may be *faulty*, meaning that they do not follow their program correctly. Otherwise, the nodes are called *correct*. To solve Interactive

²for another realization of written messages, see [12]

Consistency, the nodes should compute a vector $V = (y_1, \dots, y_n)$ satisfying the following requirements:

- (*Agreement*) All correct nodes compute exactly the same vector.
- (*Validity*) For any correct node p_i , the i th vector entry corresponds to its initial value, i.e., $V[i] = x_i$.

For example, in case of sensor readings, all correct sensor nodes will know the values read by all other correct sensor nodes. As faulty sensor nodes can exhibit arbitrary behavior (crash, measure an unrealistic value, etc.), the correct nodes are guaranteed to agree at least on some default value for these nodes.

E. Protocol Choice

Algorithms for Interactive Consistency are plentiful. Their applicability depends on the assumptions concerning the communication channels between the nodes, and the nature of possible faults.

Unfortunately, reaching agreement even in case of a single link failure is impossible [16]. However, if one reasonably restricts the impact of link failures, agreement becomes possible [17]. Therefore, we decided to implement the IC protocol based on the agreement protocol by Schmid et al. from [17], called OMH(m) in the following (m is an internal parameter). OMH(m) enables agreement on a single value. This protocol has additional advantages: it is formally verified, and it allows a trade-off between the number of nodes needed for the agreement, and the nature and number of tolerated failures.

OMH(m) is designed with respect to the following *node failures*:

- *Manifest failures* result in nodes sending messages which are detectably wrong, e.g., reporting temperature -20 degrees Celsius in midsummer, or sending a message in an unspecified format.
- *Omission failures* occur when the node fails to send or to receive a message. In this case if an expected message did not arrive, the receiver cannot distinguish its own omission failure from the sender's omission failure.
- *Symmetric failures* occur when a sender broadcasts the same "bad" (but not detectably wrong) value to several receivers. For example, if a temperature sensor reports wrong a value which is 5 degrees more than the temperature of the real environment, this can be a symmetric failure (depending on the application).
- *Asymmetric failures* occur if a node, instead of broadcasting the same value to all receivers, sends

at least two different values to different receivers.

Also the following *link failures* are possible:

- *Simple sending failures* occur if a (possibly broadcast) message does not arrive at all receivers, or arrives at some receivers in a detectably bad condition.
- *Simple receiving failures* occur if messages (possibly, from multiple senders) do not arrive at the receiver, or arrive in a detectably bad condition.

Note that one cannot simply substitute sending and receiving link failures with the node omission failures. Whereas a failed node, according to definition in Section III-D, is not required to reach the correct agreement, a node which experiences link failures should be able to do this.

The OMH(m) protocol runs in $m + 1$ rounds of communication. Let f_a , f_s , f_o , and f_m denote the number of asymmetric, symmetric, omission, and manifest node failures, respectively. Let f_l^s and f_l^r denote the number of simple sending and simple receiving link failures, respectively, per node in each protocol round. Then OMH(m) can withstand these failures using n nodes, where

$$n > 2f_l^s + f_l^r + 2(f_a + f_s) + f_o + f_m + m \quad (1)$$

for any $m \geq f_a + f_o + \min\{1, f_l^s\}$.

The above flexible failure model enables system designers to choose the parameters n and m according to the expected failure modes.

1) *Choice of the Protocol Parameters*: Equation 1 enables system designers to trade-off the protection level of their agreement protocol against available resources. We decided to take $m = 1$, such that the protocol runs for two rounds, to reduce communication overhead. As we only had 7 sensor nodes, i.e., $n = 7$, we carefully considered against which failures we can protect our system.

We tolerate one sender and one receiver link failure per node in each protocol round ($f_l^s = f_l^r = 1$).

With $m = 1$ we cannot tolerate asymmetric or omissive node failures due to the restriction $m \geq f_a + f_o + \min\{1, f_l^s\}$. However, this agrees well with our benign adversary model. We note that asymmetric and omission failures can still happen in reality due to node's unreliable hardware, and our implementation would not be able to tolerate such failures.

We decided to tolerate one symmetric failure ($f_s = 1$) which accounts for one node heated up by an attacker,

or for the case where a sensor node gets heated up incidentally, e. g., through direct sun light.

Besides, we also can tolerate one manifestly faulty node, which accounts for a crashed node, or for a node which goes out of synchronization with the other nodes, or for a node which reports a detectably bad temperature value due to the failed sensor.

In this case, according to equation 1, we need $n > 6$ nodes.

Of course, having more nodes would have made our choice less restrictive, especially, would allow to tolerate more link failures per node, and more manifestly faulty nodes, which would significantly improve the system robustness. However, we note that only increasing the number of nodes would not help us to tolerate maliciously reprogrammed nodes. To do this, we would have to increase m , the number of communication rounds.

F. The Protocol

In the following, we describe the Interactive Consistency protocol based on the protocol OMH(m) for the case $m = 1$ which we implemented. For the full protocol description, see [17].

round 1: In the first round, each node p_i sends its input x_i to all other $n - 1$ nodes. It receives messages $r_1(p_j)$ from all nodes $p_j \neq p_i$. In case no failures occurred, $r_1(p_j) = x_j$. If no message, or a detectably bad message was received, $r_1(p_j) = \perp$. Otherwise, $r_1(p_j)$ can be any admissible value. Thus, node p_i receives an $(n - 1)$ -vector V_{p_i} of values in the first round.

round 2: In the second round, each node p_i sends its vector V_{p_i} to all other $n - 1$ nodes. It receives messages $r_2(p_j)$ from all other nodes, where $r_2(p_j) = V_{p_j}$ in case no failures occurred. As in the previous round, V_{p_j} can be \perp if no message or a detectably bad message arrived, and otherwise, it can be any $(n - 1)$ -vector of admissible values, including \perp values.

After the second round, each node p_i assembles $n - 1$ possible values for each node $p_j \neq p_i$: In the first round it received $r_1(p_j)$, and the other values can be extracted from the vectors of values received in the second round. Then for each $p_j \neq p_i$, p_i discards all \perp values, and computes the majority on the remaining values (ties can be resolved in any deterministic way). The computed value y_j becomes the j th component of the final Interactive Consistency vector: $V[j] = y_j$.

IV. IMPLEMENTING INTERACTIVE CONSISTENCY

We implemented the above IC protocol and experimentally tested the robustness of the implementation with respect to specified attacks and failures.

For our testbed we chose Scatterweb sensor boards (ESBs) [18] that are equipped with a DS1629 IC temperature sensor. The ESBs run on three alkaline batteries (1,5V).

A. Message Authenticity and Freshness

As outlined in Section III-E, the origin of any message should be reliably determined by the receiver, including the situation where an entity replays a message. Thus, apart from carrying authentication information, each message also should be *fresh* (sent for the first time).

Since the radio channel is broadcast, an attacker can easily collect packets and reinject them. In our example, an attacker could collect packets during a legitimate HF situation when temperature is above the HF threshold. Later, the attacker reinjects those packets and fools the sensors into a HF decision.

For authentication, we use *message authentication codes (MAC)*, which are a cryptographically secure checksum. Each node shares a secret key with any other node in the system, and authenticates all its messages to each receiver by computing the MAC on the message using this secret key. We assume that secret keys are pre-distributed in the predeployment phase, such that any node in an office knows the identities of all other nodes in the office, along with the corresponding secret keys. The keys are shared pairwise, i.e., each node has a separate secret key with each of its $n - 1$ neighbors.

Our sensor nodes use counters to identify a particular protocol run. If a node senses the temperature above the HF threshold, it increases the protocol counter and starts a new protocol run. This already provides a weak freshness mechanism. However, if a counter overflow occurs, message replays would become possible.

We chose the 16 bit counter with respect to the ESB node characteristics. According to the described below energy consumption of an IC run, even in case the ESBs are permanently involved in IC runs with the counter increasing by 1 every run, they would run for 54 hours before the overflow occurs.

For the case of counter overflow, we implemented a challenge-response protocol for setting up the new counter. This measure resulted in appending a freshness identifier, which is a 16 bit random number, to the counter in order to distinguish between protocol runs with the same counter.

B. Performance

We ran our tests using 7 sensor nodes. Our implementation uses 53428 bytes ROM and 1702 bytes

RAM. The application was able to cope with several link failures, temporal absence of nodes, and wrong temperature values according to the fault model of our protocol. For example, two absent (temporarily shielded) nodes and one fake HF event can be tolerated.

The interactive consistency algorithm is still comparably greedy on communication resources. In Scatterweb, the packet header consists of 10 bytes. Each temperature value is 2 bytes long, the same size are also the counter, the freshness identifier, and the MAC. Thus, a step-one packet has a size of 18 bytes, while a step two packet has a size of 30 bytes for 7 nodes, as each node has to resend the values received from other nodes.

A single protocol run takes 3 s on average. In each round, each node has to compute 6 MACs on its messages to the other nodes, and to verify 6 MACs on the messages received from the others. This work takes approximately 240 ms. The most important factor in the time complexity of our protocol is due to the timeout values which determine the time during which the node waits for messages. In each round, nodes try to send their messages almost simultaneously, which leads to a high number of collisions. Thus, in the first round the timeout value had to be set to nearly 1 s, while in the second round, this value had to be further increased due to the increased message sizes.

V. CONCLUSIONS

We considered the applicability of the classical distributed agreement protocols in sensor networks. We identified scenarios where these protocols may be useful, and implemented an agreement protocol for one of these scenarios.

Requiring 3 s for a single run of an agreement protocol which tolerates such a benign adversary as ours is too slow, and also too resource-consuming. This performance problem can be partially contributed to the contention-based MAC layer of the ESB sensor nodes, because the protocol needs large timeouts. We think that the most promising way to enhance the efficiency of the agreement protocols is the cross-layer optimization, which is an acknowledged paradigm for building energy-efficient applications. Our ongoing work is to design a fast and an energy-efficient agreement protocol for sensor networks tolerating more powerful adversaries.

REFERENCES

- [1] R. Kieckhafer, C. Walter, A. Finn, and P. Thambidurai, "The MAFT architecture for distributed fault tolerance," *IEEE Transactions on Computers*, vol. 37, no. 4, pp. 398–405, April 1988.
- [2] M. K. Reiter, "The rampart toolkit for building high-integrity services," in *Selected Papers from the International Workshop on Theory and Practice in Distributed Systems*. London, UK: Springer-Verlag, 1995, pp. 99–110.
- [3] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in presence of faults," *Journal of the ACM*, vol. 27, no. 2, pp. 228–234, April 1980.
- [4] D. Dolev and R. Reischuk, "Bounds on information exchange for byzantine agreement," *J. ACM*, vol. 32, no. 1, pp. 191–204, 1985.
- [5] S. Zhu, S. Setia, and S. Jajodia, "LEAP: efficient security mechanisms for large-scale distributed sensor networks," in *Proceedings of the 10th ACM conference on Computer and communication security*. ACM Press, 2003, pp. 62–72.
- [6] S. Zhu, S. Setia, S. Jajodia, and P. Ning, "An interleaved hop-by-hop authentication scheme for filtering of injected false data in sensor networks," in *IEEE Symposium on Security and Privacy*, 2004, pp. 259–271.
- [7] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical en-route filtering of injected false data in sensor networks," *IEEE Journal on Selected Areas in Communications, Special Issue on Self-organizing Distributed Collaborative Sensor Networks*, 2005.
- [8] T. Clouqueur, K. K. Salujaand, and P. Ramanathan, "Fault tolerance in collaborative sensor networks for target detection," *IEEE Transactions on Computers*, vol. 53, no. 3, pp. 320–333, March 2004.
- [9] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, 1982.
- [10] R. J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, Inc., 2001.
- [11] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, 1985.
- [12] B. Pfitzmann and M. Waidner, "Unconditional byzantine agreement for any number of faulty processors," in *STACS '92: Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science*. London, UK: Springer-Verlag, 1992, pp. 339–350.
- [13] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler, "Lessons from a sensor network expedition," in *First European Workshop on Sensor Networks (EWSN)*, Jan. 2004. [Online]. Available: citeseer.ist.psu.edu/szewczyk04lessons.html
- [14] G. McGraw and J. Viega, *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley, Sept. 2001.
- [15] A. Becher, Z. Benenson, and M. Dornseif, "Tampering with notes: Real-world physical attacks on wireless sensor networks," in *3rd International Conference on Security in Pervasive Computing (SPC)*, April 2006.
- [16] J. Gray, "Notes on data base operating systems," in *Advanced Course: Operating Systems*, ser. Lecture Notes in Computer Science, M. J. Flynn, J. Gray, A. K. Jones, K. Lagally, H. Opderbeck, G. J. Popek, B. Randell, J. H. Saltzer, and H.-R. Wihle, Eds., vol. 60. Springer, 1978, pp. 393–481.
- [17] U. Schmid, B. Weiss, and J. Rushby, "Formally verified byzantine agreement in presence of link faults," in *The 22nd International Conference on Distributed Computing Systems (ICDCS '02)*, Vienna, Austria, July 2002, pp. 608–616.
- [18] "Scatterweb homepage," <http://www.scatterweb.net/>.