Byzantine Agreement Service for Cooperative Wireless Embedded Systems

Wenbo Xu, Martin Wegner, Lars Wolf, Rüdiger Kapitza
Institute of Operating Systems and Computer Networks, TU Braunschweig, Germany
Email: wxu,wegner,wolf,kapitza@ibr.cs.tu-bs.de

***Abstract*—Recently there is a number of scenarios such as vehicle platooning, UAV swarms and cooperative robots, where a group of autonomous entities needs to make joint decisions to operate effectively. Distributed agreement protocols can play a key role to establish a common view on context parameters and to coordinate joint actions. Due to the harsh environmental conditions in some of these scenarios and their safety critical nature such protocols need to tolerate arbitrary faults and even malicious attacks.**

This paper presents a framework for Byzantine fault tolerant agreement for small-sized groups of autonomous, wirelessly connected systems. It focuses, besides providing support for general purpose value agreement, on the agreement of distributed sensor readings. The latter enables to establish a joint view on context conditions, thereby building the basis for joint coordinated actions. As classical Byzantine fault tolerant agreement protocols require $3t + 1$ participants to tolerate t faulty nodes, we also consider a hybrid fault model by utilizing a trusted subsystem, which can only be subject to crashes. The latter reduces the required group size for agreement to $2t + 1$ nodes and reduces the message complexity of the protocol, which is essential for the targeted scenarios. The experiment results show that the trusted subsystem can effectively increase the efficiency. [1]

***Index Terms*—Byzantine fault tolerance, distributed agreement, wireless embedded system**

I. Introduction

Reaching agreement to make joint decisions is important for many distributed applications. Especially in recent years, embedded mobile systems are rapidly developing, and more powerful capabilities of them are exploited by building cooperative groups or clusters. Examples of these scenarios are vehicle platooning, unmanned aerial vehicles (UAVs) swarms, cooperative robots, etc. These scenarios have in common that all entities in the group have their own perception of the environment, e. g., through individual sensors. Simultaneously, they are equipped with actuators that can in turn change the environment. So most of the time, such systems need to achieve a joint view about the state of the environment before they take an action. In conclusion, agreement has to be provided as a service for these systems.

These systems are commonly safety-critical, thus they require a certain degree of fault tolerance, i. e., the group will not fail even if a limited number of members is not working correctly. Here, faults are not only limited to fail-stop ones like crashes. Because of an unstable environment, systems might suffer physical damages, transient or permanent hardware faults, sensor malfunctions and even malicious attacks, all of which are referred to as Byzantine faults [1]. Consequently, a Byzantine fault tolerant agreement is helpful, on one hand, to protect members from being impacted by faulty ones, and on the other hand, to keep a member still functional even if it suffers some kinds of malfunctions, e. g., a broken sensor.

In related work, iterative approaches can be found, i. e., the individual values are converging to an agreement [2, 3]. Via simulation the capability is shown for a large group with dozens of nodes.

In this work, we focus on directly making an exact agreement in a comparably smaller group. Besides the "obey-the-general" of [1], we also consider a special case of valid sensor value agreement. We provide several options of algorithms to solve the periodical agreement problem, and we also discuss the use of a special trusted subsystem to decrease overhead. We implement the Byzantine agreement protocol on our testbed of RaspberryPis in ad-hoc wireless mode.

The paper is organized as follows. Section II discusses related work. Section III defines the system model and gives the problem statement. Section IV presents the design of algorithms solving both follow-command agreement and value-consensus. Section V shows the evaluation results and Section VI concludes the paper.

II. Related Work

Distributed agreement has been studied for many years, but most researches focus on the application of state machine replication (SMR). Most famous examples are Paxos [4] tolerating crash faults and PBFT [5] tolerating Byzantine faults. They and almost all the derivative agreement protocols are shaped to fit into the SMR application. They exploit the powerful computation resource and multiple Ethernet network interfaces to achieve maximum throughput. However both the system assumption and goal are different for embedded mobile systems,which are more resource and energy limited and normally communicate via more unstable wireless ad-hoc networks. They also do not pursue such objective of high throughput.

Reiser et al. optimize distributed agreement for embedded systems [6]. They discuss the situation when both ad-hoc and infrastructure networks are available, and how to utilize the infrastructure to achieve stronger guarantees. Mocanu et al. implement Paxos in multi-agent system [7], but Byzantine

[1]This work is part of the DFG Research Unit *Controlling Concurrent Change*, funding number FOR 1800.

faults are not covered. KARYON is a system architecture for safe coordination among autonomous cooperative vehicles [8]. The authors also realized that a consistent view of the state is indispensable, therefore agreement protocols are needed as an important middleware. But rather than solving the agreement problem itself, this work is focusing on an architecture level support. Similarly Köpke's thesis offers a support to consensus in WSN from the communication protocol stack level [9].

Some works directly present Byzantine agreement algorithms for embedded wireless systems. Turquois is such a randomized binary consensus protocol designed for wireless ad-hoc networks [10]. It utilizes UDP broadcast to benefit from the shared communication medium. Different from binary consensus, sensor values form a huge (if not infinite) value domain, which brings a question to the value validity. Marzullo already addressed the issue of fault tolerance for continuous-valued sensors [11]. His approach is similar to an n-modular redundancy that votes for not a scalar but an interval of the sensor value. It can be applied in an individual system requiring reliable sensors, but does not solve the distributed agreement among autonomous systems without a centralized voter. In the latter case, each process has a value read from its sensor, and wants to agree on a common value. Ideally the value is from a correct process. But Neiger proved that a deterministic algorithm requires at least $n > t \cdot |\mathcal{D}|$ nodes to tolerate t Byzantine faulty nodes where $\mathcal{D}$ is the value domain [12]. A recent work of Stolz and Wattenhofer overcomes this issue with another validity definition, called *median validity* [13]. It only requires that the agreed value is close to the median of all correct processes. Although their algorithm is designed for synchronous system, which is not applicable to our target domain, the validity definition is useful when the values are in a continuous space, such as the physical sensor value.

There are also several orthogonal directions to our work. A number of works about distributed agreement in cooperative systems deal with the adaptive and convergent consensus (e. g. [2, 3]). They adopts the discrete time dynamic model and converge iteratively. There are also models which only requires the values of different processes are within ε of each other, namely *approximate consensus* [14]. But we focus more on an exact agreement and an explicit termination criterion. Wagner et al. propose a middleware support for context-as-a-service [15]. It manages different sensors serving for the same device to build a better overview of the context.

III. System Model and Problem Statement

In this section we specify our system model and define our Byzantine agreement problem.

A. Process and Initial Value

For simplicity we only consider a static group in which everyone has a unique ID that is known to each other. The group consists of n processes: $\{p_0, \ldots, p_{n-1}\}$. Every process might have an initial value v_i, e. g., from its sensor. A process is called *correct* if 1) its initial value is correct and 2) it exactly follows the algorithm specification. Meanwhile up to t ($< n/3$) processes can be *faulty*, meaning that they can behave arbitrarily such as crashing, taking an incorrect value from a malfunctioning sensor or actively working against the protocol. Later we show that with the help of a trusted subsystem we can relax the requirement to $t < n/2$. It is worth noting that t is distinctive from the actual faulty processes number $f (\leqslant t)$. The former is a system parameter indicating the upper bound, while f is an unknown non-deterministic variable of runtime. The correctness of an initial value is application dependent. For example in the case of a physical value sensor, assume the actual value is v and there is a permissible error δ of the sensor. Then a correct value is within the range $v \pm \delta$.

B. Problem definition

The processes in the group are periodically agreeing on a value. Each period is called an *agreement instance*. The outcome of an agreement instance is that every process decides a value, with the following properties: 1)*Agreement*: No two correct processes decide differently. 2)*Termination*: Every correct process eventually decides. 3)*Validity*: The decided value v is *valid* (defined later). Based on who is allowed to propose a value, the problem is classified into two scenarios. One is the *follow-command*, where only one special process called *primary* proposes a value, and all the other processes decide to accept it or not. The Byzantine General Problem [1] is exactly this case. An example of this scenario can be a platoon of cars deciding which route to choose to the destination. The primary could be the car in the most front, and gives an instruction to the followers. If the primary is faulty and prevents an agreement, it should be detected and replaced by a new primary. The validity property in this case requires that if the primary is correct, its proposal should be the final decision of everyone. This also implies which use cases are suitable for the follow-command scenario: either all options that the primary can propose are equally good and harmless, or a bad proposal can be easily detected and rejected by everyone.

The other scenario is the *consensus*, where all processes has their own initial values. For example a platoon is agreeing on a sensor value, e. g. temperature. If a follower received a proposal of 5 °C but its local sensor reads −30 °C, it cannot tell which one is correct. But they have to agree on a "good" value. To define what is a good value, we adopt the median validity of Stolz et al. [13]: Denote the median of a sorted vector V with length n (indexed from 0) as: $\texttt{median}(V) := V[\lfloor \frac{n-1}{2} \rfloor]$ [2]. Assuming there are actually f ($\leqslant t$) faulty processes during runtime yet not known to the algorithm. Let $Sorted\_Correct$ denote the initial values from all correct processes, sorted in an ascending order. So it has a length of $(n - f)$. Denote $c := \lfloor \frac{n-f-1}{2} \rfloor$, namely the middle index of $Sorted\_Correct$.

Definition 1. Median validity*: assuming* $n \geqslant 3t+1$*, a decision* v *is* valid*, if*

$$Sorted\_Correct[c - t] \leqslant v \leqslant Sorted\_Correct[c + t] \quad (1)$$

[2]We choose the smaller value close to the middle as median if the array length is even. It can also be configured as the greater one.

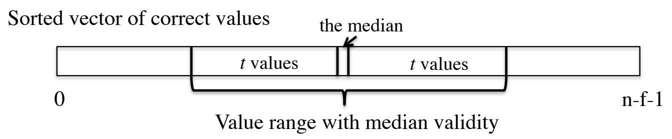
Sorted vector of correct values
the median
t values
t values
0
n-f-1
Value range with median validity

Figure 1: Median Validity

In other words, a valid value is the one within the range of the middle $(2t+1)$ correct values as illustrated in Figure. 1. Because $n \geqslant 3t+1$ and $t \geqslant f$, it must hold $n-f-1-t \geqslant c \geqslant t$. So we do not have to worry about the index $(c-t)$ and $(c+t)$ being out of bounds. We will discuss the case of $2t+1 \leqslant n < 3t+1$ in Section III-D.

Under this definition, a valid value does not necessarily come from a correct process (which might be hard [12], as explained in Section II). A faulty process can still influence the outcome of agreement, but it has to choose a good enough value within the valid range, otherwise it will be ignored. Figure 1 also indicates that the definition is a stronger condition than that "a valid value is from the range of all correct values". They are equivalent only when $n = 3t+1$. If n becomes greater with the same t, the median validity range still keeps close to the median of the correct values.

C. Network

Processes communicate via network messages. Basically we do not assume a synchronous network. Thus, messages can experience an unbounded delay. However, in order to achieve both agreement and termination, we assume the partially synchronous model to overcome the FLP impossibility [16]. In this model the system is switching between synchronous and asynchronous periods. So we classify each agreement instance as *good* or *bad* one. In a good instance the system eventually becomes synchronous and all messages arrive in time so that an agreement can be achieved with termination. But in a bad instance we have to trade one for the other. In most state machine replication systems, termination (liveness) is sacrificed to keep agreement (correctness) for a fail-stop service. But this is not an absolute rule, especially in systems with certain time-critical tasks. We can let processes abort an agreement instance in an "elegant" way if they cannot reach agreement in the end. But that is out of scope of this paper.

Definition 1 is the tight bound for all deterministic asynchronous agreement algorithms with the following lemma:

Lemma 1. *Given $n > 3t$, without further information about the values, no deterministic agreement algorithm for asynchronous system guarantees that the agreed value v always satisfies either*

$$Sorted\_Correct[c-t] < v \leqslant Sorted\_Correct[c+t] \quad (2)$$

$$\text{or } Sorted\_Correct[c-t] \leqslant v < Sorted\_Correct[c+t] \quad (3)$$

We omit the formal proof because of space limit. The basic idea is that in an asynchronous system, we cannot distinguish a slow process from a crashed one. As a result, in the worst case a decision is based on only $(n-2t)$ correct values, plus t faulty values but excluding the other t correct but slow ones. Faulty ones can then collude all together to propose very small or big values, to "shift" the result to either direction. [3] From this lemma we can also conclude that: even if faults are restricted to crashing or providing an incorrect initial value, Definition 1 is still a tight bound in asynchronous system.

D. Encryption and trusted subsystem

Although Byzantine processes can be arbitrarily faulty, there are some limits. Firstly they are computationally unable to break the encryption mechanisms. Though not mentioned in most simulation-based works, basically all messages should be certified with a message authentication code (MAC). Some messages even need a digital signature to proof that they are not modified during forwarding. In the rest of the paper, a message with MAC is not explicitly denoted. We assume the symmetric and asymmetric key pairs are already correctly distributed between processes before the algorithm starts.

Furthermore we consider a variant of the fault model, namely the *hybrid fault model*, that every process is equipped with a trusted subsystem to provide a relatively simple but critical function. It should be more trustworthy than the other part of the system for certain reasons, so that even a malicious process cannot bypass it, unless stop its service (fail-stop). Examples of implementations include the use of special hardware technology like Intel SGX [17] or ARM TrustZone [18]. We use a similar trusted function as in CheapBFT [19] to prevent equivocation attack, namely a Byzantine process sending contradictory messages to different recipients in a broadcast. It is achieved by keeping the process ID, a monotonic counter together with the encryption operation inside the subsystem. Every time a process broadcasts a message, the messages is combined with the ID and the counter and then encrypted with the secret key in the subsystem. Then the counter increases by one. In this way a malicious process cannot send different messages with the same counter. As a result the tolerable faulty processes can increase from $\lfloor \frac{n-1}{3} \rfloor$ to $\lfloor \frac{n-1}{2} \rfloor$. Another benefit is that digital signature can be replaced by the more efficient symmetric encryption, because the subsystem will not leak the secret keys even to its host process.

IV. Algorithm Design

Now we show the solution to both the follow-command and consensus scenario with the same prototype. We also explain how the trusted subsystem can be used and influence the agreement protocol.

A. Separate value exchange and agreement phases

We use the primary-based protocol PBFT [5] as the prototype. The reason is that the follow-command scenario is naturally matching a primary-based agreement. And we want to make our implementation so flexible that switching to the consensus scenario does not need a totally new protocol.

To let a consensus scenario also benefit from the primary-based agreement, we firstly revisit the PBFT protocol in more details. It consists of three modules:

[3]Note that this lemma does not hold if we change our validity definition to "a valid value is within the range of all correct values". As explained previously, our definition is stronger if n becomes greater than $(3t+1)$.

1a) Propose: the primary proposes (preprepare) a value, and the follower verify the validity of it.
2) Decide: after two rounds of message exchange (prepare and commit) every one tries to decide. If any two correct processes decide, they decide the same value. It is possible that no one decides at all.
3) View-change: if too many processes fail to decide in the end, the system enters a new view, namely a new primary replaces the old one. It also guarantees that if any correct process has already decided, this information is passed to the new view to everyone, so that correct processes can only decide that value afterwards.

Clearly, the follow-command scenario can directly use this protocol with little modification, and the consensus scenario only differs in the first module. In consensus, processes need a value exchange procedure to collect information about others' values. So the propose module is modified as:

1b) Propose: processes exchange their values. Then the primary proposes a value, and the follower verify the validity of it based on the collected information.

B. Algorithm for value exchange in consensus scenario

The value exchange algorithm relies the following lemma:

Lemma 2. *Let A be an vector which contains the initial values of any $(n - t)$ processes. Then* $\texttt{median}(A)$ *is always valid according to Definition 1.*

The proof is omitted. This lemma implies that if the processes can agree on a vector of initial values from $(n-t)$ processes, they automatically agree on a median valid value by simply choose the median of that vector. The former problem is referred to as *vector consensus*.It is sufficient but not necessary for median validity. So we turn to the *weak interactive consistency* [20], which ends up with a primary proposing a vector, and every follower verifies and then *accepts* or *rejects* it, but not agrees on it. The only two requirements are:

- If the primary is correct and the agreement instance is a good one (see Section III-C: Network), every correct process will accept the proposal.
- Even if the primary is faulty, it cannot lead a correct follower to accept an invalid value.

With this idea to implement the propose module, a faulty primary can only prevent the system to make a progress. This can be solved via the primary rotating in the view-change.

Weak interactive consistency is shown to be solved either with or without digital signatures [20]. The original algorithms were implemented in a round-based model with timing assumptions in each round. In our work the algorithms are designed as time-free as possible, so that they are more practical in asynchronous situation (similar concern is considered in [21]). The algorithms are slightly modified to get rid of the round notation and time assumptions.

The signature-based algorithm is intuitive: every process signs its value with signature and sends to the primary. 1) The primary waits until it collects $(n - t)$ values, which should always happen if the instance is good. Then the primary concatenate the original signed message as the certificate, and propose the median of them together with the certificate. 2) Since a faulty primary cannot forge the signatures, correct followers can verify it and will only accept a valid value. The two arguments above ensure the correctness of the algorithm.

Algorithm 1 Signature-free proposal and verification (single instance)

```
1  Initialization:
2      $v_p \leftarrow$ initial value of $p$
3      $V_p[:] = \perp$
4      $M_p[:][:] = \perp$
5  Task T1:
6      broadcast $\langle VALUE, p, v_p \rangle$    /* $\langle$MSG-TYPE, sender-id, ...$\rangle$ */
7      if $p = primary$ do
8          wait until $n - t$ values in $V_p$ are not $\perp$
9          broadcast $\langle PROPOSE, p, \texttt{median}(V_p), V_p \rangle$
10     upon $\langle PROPOSE, primary, med, V_{primary} \rangle$ is delivered do
11         if $|V_p| = n - t$ and $med = \texttt{median}(V_p)$ do
12             wait until $\forall i : V_{primary}[i] = \perp$ or
                   $|\{j : V_{primary}[i] = M[j][i]\}| \geqslant t$
13             accept $med$
14 Task T2:    /* repeatedly execute to handle $VALUE$ and $ECHO$ */
15     while true do
16         upon $\langle VALUE, i, v_i \rangle$ is delivered do
17             if $p \neq primary$
18                 broadcast $\langle ECHO, p, i, v_i \rangle$
19             else do
20                 send $\langle ECHO, p, i, v_i \rangle$ to itself
21         upon $\langle ECHO, j, i, v_i \rangle$ is delivered do
22             $M_p[j][i] = v_i$
23             if $p = primary$ and $|\{(j, v_i) : M_p[j][i] = v_i\}| \geqslant 2t + 1$ do
24                 $V_p[i] = v_i$
```

As for a signature-free version, one more message round is needed, and it requires $n > 3t$. The algorithm is shown in Algorithm 1 and Figure 2. The value $M[j][i]$ means that "process j confirms it has received $M[j][i]$ from process i". And we have the similar arguments for the correctness:

Lemma 3. *If the primary is correct and the algorithm is running in a good instance and $n > 3t$, every correct process will eventually accept the proposal of primary.*

Proof. Line 8 and 12 are the only places to block the algorithm. If it is in a good instance and all messages are eventually delivered, all the $(n-t)$ correct values must be echoed by the $(n - t)$ correct processes, making line 8 moving on. $(n - t)$ echos contain at least $(n - 2t - 1)$ from correct processes (except the primary), which is at least t given $n > 3t$. Any correct process sending echo to the primary should also send to everyone else, so line 12 will also continue. This confirms the termination of the algorithm. □

Lemma 4. *A correct process will only accept a median valid value, if it accepts any value in the end.*

Proof. Line 11-12 ensures that every non-$\perp$ value in $V_{primary}$ should also be echoed by another t processes. Among those t processes plus the primary, at least one is correct. So every $V_{primary}[i] \neq \perp$ is confirmed by at least one correct process. Since correct processes do not lie, $V_{primary}[i]$ must be initially sent by i. The conclusion follows from Lemma 2. □

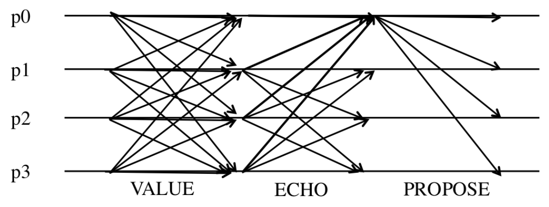

Figure 2: Signature-free value exchange. p_0 is the primary.

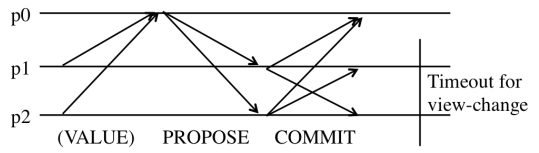

Figure 3: Normal case communication pattern with a trusted subsystem. p_0 is the primary. The $VALUE$ message round is not required in a follow-command agreement.

C. An extension: using a trusted subsystem

We use the trusted subsystem (Section III-D) to perform the encryption/decryption operation and to prevent a faulty process sending contradictory messages to different recipients (equivocation). A monotonic counter inside the subsystem is applied, similar to CheapBFT [19] or MinBFT [22]. This affects both the value exchange and agreement phase.

The most significant benefit is that the lower bound of total processes reduces from $(3t+1)$ to $(2t+1)$ to tolerate t Byzantine processes (Section III-D). So we can build a minimal group of three rather than four provided $t=1$. This is almost the optimal case we can achieve, because if there are only two while one of them might be faulty, an agreement is pointless since each one mistrusts the other.

The agreement protocol can be simplified too. A follow-command agreement needs only two rounds communication: propose-commit, compared to the three rounds communication preprepare (propose)-prepare-commit of [5]. As for the consensus scenario, we can stick to the signature-based version because all message authentications are performed inside the trusted subsystem, which can fulfil the duty of a digital signature as explained in Section III-D. The communication pattern is illustrated in Figure 3. Because of the space limit, we omit the detailed description of the protocol.

However there is one issue of the median validity in the consensus scenario. Recall Definition 1 requires $(n \geqslant 3t+1)$. To prevent the index out of bound, formula 1 is modified as $Sorted\_Correct[\min(0, c-t)] \leqslant v \leqslant Sorted\_Correct[\max(c+t, n-f-1)]$.

Lemma 2 had the same requirement. Otherwise, consider an example of temperature measuring. The real value is about 5 °C. One correct process sends a value 5.1 °C, another faulty one sends −30 °C and the third one with 4.9 °C is slow and thus excluded from the agreement protocol. But no one can infer a good value from 5.1 and -30. Taking the average does not help because of an extreme outlier as in this example. To overcome this issue, extra information is required, e. g. a maximum permissible error δ provided by the sensor specification. Accordingly the vector A in lemma 2 should be modified as: a vector contains $(n-t)$ initial values, and the difference between any two values is within 2δ.

Table I: Measured delay in ms

Delay (ms)	Follow-command	Consensus signature-based	Consensus signature-free
No trusted module	113	786	118
With trusted module	60	119 (HMAC)	-

D. An issue about the time

Most embedded mobile system applications are time-critical. For example the sensor reading requires a certain degree of “freshness” of the value. It raises a new question: how does a process distinguish messages of the current period from the ones from the past? One intuitive solution is using timestamped messages. By comparing the timestamp with its own clock, the recipient knows whether the message belongs to the current agreement instance or to an old one. This requires a synchronized clock, e. g., via GPS, which is reasonable at least in vehicular communication [23]. However this assumption might not always hold in a more general, heterogeneous system. When a time service or synchronized clock is not available, a malicious process may even intercept and retransmit messages with a delay (replay attack). How to address this issue is left as future work.

V. Evaluation

We build our testbed with four RaspberryPi 3s, connected via ad-hoc wireless mode with static IP addresses. Letting three of them ping the fourth one simultaneously, the average round trip time is measured as 7.99 ms. We use DS18B20 1-wire digital thermometers as sensors. The sensor reading time is 827 ms in average, which is a main bottleneck, so it is not included in the agreement delay below.

Message authentication and digital signature use HMAC and RSA-1024 on SHA256 hash function respectively. As a preliminary prototype, we simulate the trusted subsystem with a separated process. In future work we will exploit the TrustZone technology equipped on RaspberryPi. Furthermore, we inject 2 ms delay on every function call (validate/verify every message) in the trusted subsystem. This is a quite conservative estimate reported by [19]. All experiments are conducted under the fault-free case, except that in the sensor consensus we heat up one of the thermometers to produce a wrong value.

The results are shown in Table I [4]. It is not surprising that the signature-based algorithm without a trusted subsystem is much slower, especially on such systems with limited computational resources. But there is no obvious difference between the follow-command and the signature-free consensus scenario. When using a trusted susbsystem, the delay drops to 50% in the follow-command scenario. This is due to the facts that it requires only two rounds of communication instead of three. A more remarkable improvement in the signature-based consensus, because symmetric encryption (HMAC) instead of RSA is used. The delay decreases to be as good as a signature-free version without trusted subsystem. The reason

[4]Since the trusted subsystem already acts as signature and cannot be bypassed, its signature-free version does not make any sense.

might be that the message certification/verification is the bottleneck during value exchange. Although signature-based algorithm is one round less, every message composing the $PROPOSE$ certificate still needs to be verified one-by-one.

VI. Conclusion

In this work we have discussed the Byzantine fault tolerant agreement on mobile systems, where a group of autonomous entities needs to take a joint action or reach a joint view. We studied two scenarios: In the first one, a special primary proposes and the followers accept it. The second one is a sensor value consensus. Each process reads its own value from the sensor, and the group agrees on a value close enough to every correct process. We also discussed the use of a trusted subsystem to constrain the ability of a Byzantine process and to reduce the overhead, and most importantly reduce the group size from $(3t+1)$ to $(2t+1)$, resulting in a minimal group of three processors instead of four. The experiment runs on RaspberryPis connected via ad-hoc wireless network. The results show that a trusted subsystem reduces the delay to 50% in the follow-command scenario. As for the sensor value consensus, the performance is comparable to the signature-free version in non-trusted-subsystem case. However it significantly reduces the message complexity. The future work will explore dynamic group member management, rather than a static, closed group.

References

[1] Leslie Lamport et al. “The Byzantine Generals Problem”. In: *ACM Trans. Program. Lang. Syst.* 4.3 (July 1982), pp. 382–401.

[2] Lin Xiao et al. “A scheme for robust distributed sensor fusion based on average consensus”. In: *Proceedings of the 4th international symposium on Information processing in sensor networks*. IEEE Press. 2005, p. 9.

[3] Francesco Acciani et al. “Achieving robust average consensus over wireless networks”. In: *Control Conference (ECC), 2016 European*. IEEE. 2016, pp. 555–560.

[4] Leslie Lamport et al. “Paxos made simple”. In: *ACM Sigact News* 32.4 (2001), pp. 18–25.

[5] Miguel Castro and Barbara Liskov. “Practical Byzantine fault tolerance and proactive recovery”. In: *ACM Transactions on Computer Systems* 20.4 (2002), pp. 398–461.

[6] Hans P Reiser and António Casimiro. “Optimizing Byzantine Consensus for Fault-Tolerant Embedded Systems with Ad-Hoc and Infrastructure Networks”. In: *Proc. of the 4th Int. Workshop on Dependable Embedded Systems (WDES’07)*. 2007.

[7] Andrei Mocanu and Costin Bădică. “Bringing paxos consensus in multi-agent systems”. In: *Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14)*. ACM. 2014, p. 51.

[8] António Casimiro et al. “The karyon project: Predictable and safe coordination in cooperative vehicular systems”. In: *Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on*. IEEE. 2013, pp. 1–12.

[9] Andreas Köpke. “Engineering a communication protocol stack to support consensus in sensor networks”. PhD thesis. Technischen Universität Berlin, 2012.

[10] Henrique Moniz et al. “Turquois: Byzantine consensus in wireless ad hoc networks”. In: *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*. IEEE. 2010, pp. 537–546.

[11] Keith Marzullo. “Tolerating failures of continuous-valued sensors”. In: *ACM Transactions on Computer Systems* 8.4 (1990), pp. 284–304.

[12] Gil Neiger. “Distributed consensus revisited”. In: *Information Processing Letters* 49.4 (1994), pp. 195–201.

[13] David Stolz and Roger Wattenhofer. “Byzantine Agreement with Median Validity”. In: *19th International Conference on Priniciples of Distributed Systems (OPODIS), Rennes, France*. 2015.

[14] Danny Dolev et al. “Reaching Approximate Agreement in the Presence of Faults”. In: *J. ACM* 33.3 (May 1986), pp. 499–516.

[15] M. Wagner et al. “Context as a service - Requirements, design and middleware support”. In: *2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. Mar. 2011, pp. 220–225.

[16] Michael J Fischer et al. “Impossibility of distributed consensus with one faulty process”. In: *Journal of the ACM (JACM)* 32.2 (1985), pp. 374–382.

[17] Frank McKeen et al. “Innovative instructions and software model for isolated execution.” In: *HASP@ ISCA*. 2013, p. 10.

[18] ARM. *ARM Security Technology - Building a Secure System using TrustZone Technology*. Tech. rep. PRD29-GENC-009492C. ARM Technical White Paper, 2009.

[19] Rüdiger Kapitza et al. “CheapBFT: Resource-efficient Byzantine Fault Tolerance”. In: *Proceedings of the EuroSys 2012 Conference*. Ed. by European Chapter of ACM SIGOPS. Switzerland, 2012, pp. 295–308.

[20] Zarko Milosevic et al. “Unifying Byzantine consensus algorithms with weak interactive consistency”. In: *International Conference On Principles Of Distributed Systems*. Springer. 2009, pp. 300–314.

[21] Miguel Correia et al. “From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures”. In: *The Computer Journal* 49.1 (2006), pp. 82–96.

[22] Giuliana Santos Veronese et al. “Efficient byzantine fault-tolerance”. In: *Computers, IEEE Transactions on* 62.1 (2013), pp. 16–30.

[23] Panos Papadimitratos et al. “Vehicular communication systems: Enabling technologies, applications, and future outlook on intelligent transportation”. In: *IEEE Communications Magazine* 47.11 (2009).