Transparent Protection of Aggregate Computations from Byzantine Behaviours via Blockchain

Danilo Pianini
Università di Bologna
Cesena, Italy
danilo.pianini@unibo.it

Giovanni Ciatto
Università di Bologna
Cesena, Italy
giovanni.ciatto@unibo.it

Roberto Casadei
Università di Bologna
Cesena, Italy
roby.casadei@unibo.it

Stefano Mariani
Università di Modena e Reggio Emilia
Reggio Emilia, Italy
stefano.mariani@unimore.it

Mirko Viroli
Università di Bologna
Cesena, Italy
mirko.viroli@unibo.it

Andrea Omicini
Università di Bologna
Cesena, Italy
andrea.omicini@unibo.it

ABSTRACT

Aggregate Computing is a promising paradigm for coordinating large numbers of possibly situated devices, typical of scenarios related to the Internet of Things, smart cities, drone coordination, and mass urban events. Currently, little work has been devoted to study and improve security in aggregate programs, and existing works focus solely on application-level countermeasures. Those security systems work under the assumption that the underlying computational model is respected; however, so-called Byzantine behaviour violates such assumption. In this paper, we discuss how Byzantine behaviours can hinder an aggregate program, and exploit application-level protection for creating bigger disruption. We discuss how the blockchain technology can mitigate these attacks by enforcing behaviours consistent with the expected operational semantics, with no impact on the application logic.

CCS CONCEPTS

• **Information systems → Spatial-temporal systems**; • **Security and privacy → Security services**; • **Computer systems organization → Self-organizing autonomic computing**; *Peer-to-peer architectures*; *Heterogeneous (hybrid) systems*; • **Software and its engineering → Distributed programming languages**;

KEYWORDS

Aggregate Programming, Blockchain, Security, Byzantine fault tolerance

ACM Reference Format:
Danilo Pianini, Giovanni Ciatto, Roberto Casadei, Stefano Mariani, Mirko Viroli, and Andrea Omicini. 2018. Transparent Protection of Aggregate Computations from Byzantine Behaviours via Blockchain. In *International Conference on Smart Objects and Technologies for Social Good (Goodtechs '18), November 28–30, 2018, Bologna, Italy.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3284869.3284870

Goodtechs '18, November 28–30, 2018, Bologna, Italy

ACM ISBN 978-1-4503-6581-9/18/11…$15.00
https://doi.org/10.1145/3284869.3284870

1 INTRODUCTION

Recent decades are characterised by a clear trend towards computationally richer environments and an ever-increasing integration of the physical and digital realms. More and more "smart" devices – capable of computing, sensing, and communicating – are being deployed to our bodies, homes, places, and cities. From such a transition, huge opportunities arise, especially related to a corresponding shift in focus: from what can be provided by an individual device to what can be provided collectively by an ensemble of devices.

In order to exploit the collective dimension of systems, an approach called *Aggregate Programming* has been recently proposed [6]: its key idea is to express the behaviour and coordination logic of a collection of networked, possibly situated devices through a global *aggregate program*, which is to be interpreted continuously at each device against a local context consisting of environmental data and data shared by neighbour devices. Thanks to such continuous re-assessment of local contexts and repeated device-wise execution of the global program, the approach is able to instil capabilities of self-adaptivity/self-organisation in systems, which are crucial for future generation ecosystems, e.g., in smart cities [12].

Aggregate Computing techniques have been leveraged in applications including crowd management [6], swarm robotics, and tactical networks [7]. The suitability of the approach for open scenarios as well as safety-critical applications makes security paramount practice. Indeed, aggregate algorithms – though typically resilient, e.g., to temporary failures and network partitions – can be fragile to focused attacks. Therefore, security mechanisms are needed to relax (or even remove) the fundamental presumption of cooperation that generally underlies aggregate applications.

Motivation. Despite its importance, the problem of security in Aggregate Computing has been so far overlooked in research [25]. A first discussion of the peculiar issues can be found in [11], where trust mechanisms are proposed at the application level to build attack-resistant aggregate computations. That is the only work (to the best of our knowledge) explicitly focussing on the problem, with application-level solutions that are tailored to specific algorithms and require extensive simulation for parameter tuning. Such notable gap in Aggregate Computing research represents a strong motivation for the current work. At the same time, security represents a recurrent crosscutting concern for applications, is generally hard to get right with ad-hoc solutions, is typically *assumed*

by aggregate programs (which are purely cooperative in nature), and is somewhat cumbersome to be dealt with algorithmically in Aggregate Computing [11]; these reasons motivate research about addressing security issues in a transparent fashion by delegating their management to the underlying platform.

Contribution. In this paper, rather than focussing on the application level, we aim at improving the inherent resilience of aggregate programming interpreters by injecting appropriate mechanisms on the platform hosting the aggregate computation. In particular, our contribution is a logical architecture for back-ends of aggregate programming interpreters that leverages Blockchain Technologies (BCTs) [29], so as to mitigate the effects of malicious *Byzantine behaviours* [17]—in which a device behaves inconsistently to the eyes of its neighbours. The blockchain is an emergent paradigm combining cryptography, data/control replication, immutability, and decentralisation in order to keep participants of a distributed system from misbehaving w.r.t. defined protocols. Such a secured back-end is "invisible" to the developer using Aggregate Programming, making protection from the aforementioned attacks entirely *transparent* to the application level.

Paper structure. The remainder of this paper is as follows. Section 2 provides the background, introducing the key peculiar aspects of Aggregate Programming and the Blockchain. Section 3 motivates the paper by discussing the problem of security in Aggregate Computing, highlighting the gap in the state-of-art. Section 4 conveys our contribution, showing how the Blockchain can be adopted to provide transparent protection to aggregate applications by injecting security at the platform-level. Finally, Section 5 wraps up with conclusions and perspectives on future work.

2 BACKGROUND

In this section, we introduce the Aggregate Computing framework, to provide an intuition of how it works and its operational peculiarities before discussing its security concerns in Section 3, as well as the Blockchain technology, which constitutes the basis for the solution outlined in Section 4.

2.1 Aggregate Programming

Aggregate Programming (AP) [6] is a paradigm that directly targets the behaviour of computational ensembles (also known as *aggregates*). In this approach, a logical network of devices is considered as a single computational machine, and programmed as such through declarative abstractions that take into account both the collective and the individual perspective. This stance is particularly useful to program (potentially large) ecosystems of devices that need to react to environmental and context changes—cf., scenarios like the Internet of Things, Cyber-Physical Systems, and Collective Adaptive Systems. In AP, the programmer expresses a collective adaptive behaviour through a single "aggregate program", including both local computation and coordination logic, that is continuously executed by every device of the aggregate. In typical applications, the devices are *situated* (they can inspect and operate the environment through sensors and actuators) and can only interact with a subset of other devices known as their *neighbourhood*. The approach builds

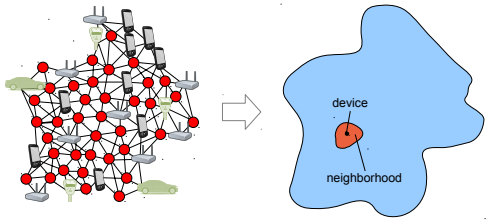

Figure 1: The Aggregate Computing paradigm: the set of devices situated in space and time constitute the lattice upon which the aggregate computation is executed. However, developers can abstract away, and reason in terms of space-time. Image from [4].

on computational fields [24] to compositionally express global behaviours. It is a responsibility of the *aggregate interpreter/compiler* to map such aggregate behaviours to the local behaviours of individual devices—ultimately resulting into a continuous, micro-level interleaving of computations and interactions. A *computational field* (or *field* for short) is a distributed data structure which maps the devices of the system (i.e., the *domain* of the computation) to computational values. The key idea of AP is to functionally describe how input fields get transformed into output fields. Crucially, the field notion is to be interpreted dynamically: input fields may vary across time, and so may output fields as a consequence. In practice, this means that a field program is to be executed in a repeated fashion by the collective.

Under the hood, every device executes in asynchronous computational rounds: the device wakes up, gathers all messages received while sleeping, executes the aggregate program, sends a message (containing all the relevant data to share) to its neighbours, and goes back to sleep. Importantly, the message sent is a multicast: devices working correctly must not send diverse messages to diverse neighbours. Non-conforming devices can be considered attackers, which, left undetected, may lead algorithms to execute undefined behaviour. Deployment-wise, devices executing aggregate programs can be situated devices relying on opportunistic networks (as it is usually the case in wireless sensor networks), but they can also defer any part of the computation (including communication) to a cloud system, with no impact on the aggregate code [26]. In the latter case, the network can be defined via software (e.g. by relying on the GPS position of devices).

A formal explanation of the aggregate paradigm as well as details on the practical languages supporting it [13, 21] is beyond the scope of this work. However, to provide a glimpse of AP, we refer to Figures 1 and 2, where we respectively show how devices "sample" the existing space and time, and how the concept of function application works in an aggregate setting. It is relevant for this work to highlight how AP has applications in medium- to large-scale contexts such as stadiums, arenas, or cities; and is proposed to be used for safety-critical applications with relevant security constrains, such as crowd tracking and steering in smart cities [12], rescue operations over unstable network conditions, or autnomous vehicles [7].

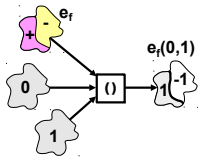

Figure 2: Function application in Aggregate Programming: rather than programming devices, developers target the portion of space-time populated by them. In this case, a field of functions yielding: *(i)* the sum function in a subspace, and *(ii)* the subtraction function in its complement; is applied using as arguments the constant fields `0` and `1`. The resulting field yields `1` in the subspace where the function `+` is applied, and `-1` where `-` is applied instead. Image from [4].

2.2 Blockchain

The term "Blockchain" refers to a set of novel technologies [1, 19, 28, 29] receiving increasing attention (from both the industry and the academia) because of the disruptive contribution they are expected to bring, i.e., within the financial, healthcare, legal, and administrative domains. The Blockchain Technology (BCT) is mostly known thanks to cryptocurrencies (e.g. Bitcoin [19]) and smart contracts (e.g. Ethereum [28]), by far the most famous and hyped applications to date, attracting investors and researchers from across the information technology landscape. In particular, BCT grew in the area of State Machine Replication [22], by introducing a novel way to face well-known problems related to the distributed *consensus* between replicas—e.g. Lamport's Byzantine Generals Problem [17].

The very first BCT reaching world-wide popularity was Bitcoin [22], which proved that the approach is able to make *mutually-untrusted* nodes of an *open* distributed system perceive a common global state, without relying on a central coordinator or a single point of trust. BCTs in fact, conveniently weave cryptography and economic aspects into consensus algorithms, to relax the need for mutually trusted entities within open systems.

The general idea behind the Blockchain is to make a number of mutually untrusted nodes in a peer-to-peer network keep track of the history of updates to a common, system-wide shared state (the "ledger"), by enforcing a protocol which essentially replicates the aforementioned updates history among all nodes. Even if a minority of nodes is misbehaving, partitioned, or compromised, the rest of them is still capable of keeping the whole system consistent and available. Every update to the data stored in a blockchain ("transaction") must be cryptographically signed by the entity proposing it – which is assumed to own a couple of public and private keys representing its identity – and approved by the majority of nodes ("miners"). Once approved, the update is applied to the system and its cryptographic hash is appended by every miner to its local copy of the distributed ledger. These features enforce accountability and compliance, which in turn discourage deliberate Byzantine behaviours since attempts to propose unauthorised updates or to tamper their history can be easily detected.

The users, or clients, of a BCT are distributed processes willing to update the ledger. To do so, they must issue and cryptographically sign a *transaction*, i.e., a message containing the formal description of the update to be performed. The transaction is then propagated throughout the blockchain network by *gossiping* [9], in such a way that most of the miners receive a copy. Once miners reach consensus about the order of such transactions, these are secured into a *block*, which is then published on the whole blockchain network. To have append-only immutability, each block must include the cryptographic hash of its predecessors to be considered valid—so as to form, literally, the "block-chain". Invalid transactions or blocks are simply dropped, thus clients must consider a transaction to be "executed" only after it has been published in a block.

BCTs can be categorised as permission*ed* and permission*less*: the former employ Certification Authorities (CAs) to endow nodes, users, and their organisations with identities and *roles*, whereas the latter are open systems where users can spontaneously create identities by their own to join the system. Permission*ed* BCTs have thus a way to control the *amount* of nodes participating the system, which in turn enables the adoption of high-throughput consensus algorithms [2, 10]. HyperLedger Fabric [1], MultiChain[1], or Corda[2] are well known examples of projects proposing a permissioned BCT. Examples of permission*less* BCTs, instead, are Bitcoin or Ethereum, in which the throughput is lower and computational cost is higher since the *Proof of Work* (PoW) approach [5, 19, 28] is exploited, where an indefinite number of "miners" compete in solving a computational *puzzle*, within a fixed time window, in order to acquire the right to update the system state—earning, as a side effect, a reward in terms of cryptocurrency. There, the BCT clients are also required to pay a small amount of cryptocurrency in order to publish state update on the blockchain system.

Both types of BCTs deliver robustness against Byzantine behaviours, although each is suited for different requirements: permission*ed* BCTs are faster but require some level of trust (in CAs), whereas permission*less* ones work even when nodes have no trust at all in anyone, but are computationally heavier.

3 SECURITY IN AGGREGATE COMPUTING

In this section, we motivate and present the research problem addressed by the paper. We describe the peculiar security issues found in Aggregate Computing and, in particular, we discuss Byzantine behaviours, for which no countermeasures have been yet proposed in literature.

Aggregate Computing is a specialised tool that helps dealing with very large-scale collective systems, possibly without the need of pre-existing infrastructural support. As such, it naturally fits scenarios like smart cities [12], crowd management [6], swarm robotics, and tactical networks [7] for rescue operations—which are often safety-critical, with potential fatal consequences of system failures. Moreover, by taking few assumptions on the concrete composition of an aggregate system, the approach is particularly suited to open scenarios (like public events and ad-hoc networks), which means the participating devices are often *untrusted*. In this perspective, it is key to understand that misbehaviours may have serious repercussions on the system, because of amplifying effects (cf., epidemic dynamics) and the potential of fostering undesired emergent phenomena. As a consequence, aggregate applications

[1] https://www.multichain.com/download/MultiChain-White-Paper.pdf
[2] https://docs.corda.net/

may be accompanied by high risks, given the high likelihood (due to openness) and impact (due to local-to-global effects) of misbehaviours, which may be unacceptable in safety-critical applications. It is therefore desirable to keep untrusted devices from silenty subvert the dynamic of an aggregate system, by providing appropriate means to mitigate the problem.

Despite the obvious relevance of security in aggregate systems, to the best of our knowledge no guarantee is provided by the foundational calculus, nor by the existing practical implementations Protelis [21] and Scafi [13]. The only form of implicit protection is provided by the so called "alignment" [3], a feature of the calculus which ensures that only devices which are executing the same code can share data. However, a malicious participant to the system may always inspect the messages sent by other participants, and forge a poisonous message which will be interpreted as correctly aligned by its neighbours. As a consequence, all sorts of attacks [18] are currently possible against systems developed with Aggregate Programming. In application-level attacks, malicious or malfunctioning devices, respecting the calculus semantics, disrupt the expected behaviour of the aggregate system. Those attacks are particularly subtle when they work by hijacking mechanisms allowing for dynamic code injection, version upgrading, aggregate process prioritization, or process replication, on which a number of applications and algorithms heavily rely [6, 20]. Some countermeasures, however, exist for this kind of security issues [11].

Platform-level attacks are instead impossible to deal with at the application level. By hijacking the platform, attackers do not respect the intended operational semantics of the programming language, and may forge messages to either attempt at a Denial of Service (via malformed messages), or to selectively attack some neighbours (via well-formed messages) tricking them into assuming an unwanted or inconsistent behaviour. The latter kind of attack is particularly subtle, as it may deceive application-level countermeasures into excluding from the aggregate computation devices which are not attackers, but are perceived as such because their behaviour is inconsistent in the eye of their neighbours.

3.1 Focus on Byzantine Behaviours

Three Byzantine behaviours that may disruptively affect aggregate systems are: *i) selective (or Byzantine) attacks* [14], in which malicious devices send different messages to each neighbour rather than performing multi-casts as dictated by the intended semantics of field calculus; *ii) masquerade attacks* [23], in which a malicious device fakes its identity impersonating another; *iii) Sybil attacks* [15], in which the attacker fakes multiple identities in order to get disproportionate weight in consensus algorithms. Attacks of this kind can be incredibly disruptive, and they cannot, in the general case, get intercepted and counteracted at the application level.

Selective attack. A malicious Byzantine attacker may target a single node, sending it poisonous values, but operating normally w.r.t. all other neighbours. The targeted device will begin sending messages with wrong values, lowering its trust level among all its neighbours, and possibly contextually lowering its trust in the attacker. Neighbours shared by the attacker and the target are key for the maneuver to succeed: they will in fact be able to perceive just the erratic behaviour of the target, as the attacker, in their eyes, is operating normally. With this technique, shared neighbours can be tricked into mistrusting the device operating normally. If the the information about trust is shared, to enhance protection, the Byzantine attack may get simpler: the attacker may in fact send, to all its neighbours, information that the target is not to be trusted, even before the attack, thus gaining further advantage. In short, a Byzantine attack can not just highjack an aggregate computation, but also exploit application level security countermeasures maliciously.

Masquerade attack. In a masquerade attack, a device impersonates another by faking its identity. A masquerade attack can exploit the fact that only the last message received by each neighbour is considered when beginning an aggregate computation round, and should be well-timed in order to have maximal penetration. Upon the reception of a target's message, the attacker may quickly modify it, e.g. injecting malicious dynamic code, and send it to neighbours with the id of the target device. All shared neighbours which have not started their computation round between the arrival of the original message and the arrival of the fake one (likely most of them, if the attacker is quick enough) will replace their payload with the attacker's. Trust-based active countermeasures are of little use in this case, as they cannot (as there is no way at the application level) distinguish the masquerade attacker and the original node.

Sybil attack. A Sybil attack would be particularly effective when the aggregate computation is density-sensitive. A whole category of algorithms exist that are not sensitive to device distribution and density [8], and programs that only rely on those are inherently resilient to Sybil attacks. However, this is not the case for many useful applications of Aggregate Programming, and in particular in those cases in which device density is leveraged as a measure to drive the entire system. This is the case, for instance, of crowd tracking and steering systems [6]. A sybil attacker can fake the identity and location of many devices tricking its neighbours – and the whole system as a consequence – into believing that people density in some area is dangerously high. In a crowd steering system this may not just result in people being steered away from certain areas for no reason, but also possibly in dangerous crowds forming around such areas, as well as a loss of trust by users in the steering system: people looking around, seeing few others, and getting alarms and suggestions of leaving the area may decide that the whole steering system is misbehaving and stop trusting suggestions entirely. In Sybil attacks, there is no need to fake messages: as such, trust-based countermeasures at the application level are useless. In their eyes, all devices are behaving correctly, and in fact they are, except that some of them do not exist at all.

4 TOWARDS BLOCKCHAIN-BASED SECURITY IN AGGREGATE COMPUTING

We now analyse the impact of adopting the Blockchain as the communication infrastructure for an aggregate computing system.

Essentially, the system is a multi-tier distributed application, where the blockchain represents the *back-end*, while the aggregate program represents the business-logic laying on top of it. In this setting, *devices* are conceived as clients of a Blockchain subsystem, and a *cluster* of interconnected machines, working as miners ("validators" henceforth), is in charge of running and maintaining the

blockchain back-end. Devices are situated into some *environment* and possibly mobile, while validators are conceptually *disembodied*, since their position in the environment is of no interest from the architectural nor application perspective. The software running on each device consists of an aggregate program interpreter running on top of a *middleware* providing I/O services towards the blockchain subsystem. Such a middleware is in charge of discovering, connecting, and interacting with validators, in order to *i)* translate the interpreter's output messages into blockchain transactions, and *ii)* the information published over the blockchain into input messages to be delivered to the interpreter. Under those assumptions, devices are able to issue (or publish) transactions on the blockchain subsystem and to listen for transactions being published by other devices.

In order to adhere to the operational semantics of Aggregate Programming, devices send messages to their neighbours by issuing multi-cast transactions on the underlying blockchain subsystem. Such transactions are validated by the blockchain subsystem according to some ad-hoc logic, preventing, for instance, devices from issuing more than one message per computational round. Furthermore, in case the blockchain subsystem is aware of devices' topology, a validation rule may be conceived to enforce devices to comply to that topology, for example, by considering invalid those transactions which do not comprehend the right set of receiver devices.

4.1 Transparent Protection

The proposed architecture inherently prevents the three types of attack introduced in Section 3.1: all the existing aggregate code can in principle execute with no change at all, and still profit of the additional security (as well as suffering the limitations that will be discussed in Section 4.2).

Byzantine attacks are impossible by construction: the message is sent as a transaction with a single payload and some receivers. Performing a Byzantine attack would require publishing multiple transactions per computation round with diverse receivers, but such messages would never get validated. Masquerade attacks are made impossible by the blockchain requirement to publish signed transactions: the attacker can no longer impersonate the target device, thus for the attack to succeed it must be able to steal the credentials (private key) of the device it wants to disguise itself as. Sybil attacks are dealt with depending on the kind of blockchain deployed: permission*ed* BCTs prevent clients from creating an arbitrary amount of identities since a CA is assumed to be in charge of issuing credential to clients in a controlled way—thus a Sybil attack requires a compromised CA in the first place; conversely, in permission*less* BCTs clients are allowed to create as many identities as they want to. Nevertheless, publishing a transaction requires an amount of cryptocurrency to be spent, but new identities have no cryptocurrency at all, thus: the attacker spends real-world money or invest considerable computational effort as miner to gain enough cryptocurrency.

4.2 The Blockchain Subsystem: Trade-offs

The first decision that must be taken when choosing a particular BCT for deploying the Aggregate Programming architecture described is whether to adopt a permission*ed* or permission*less* one, since the two variants address two fundamentally different scenarios: permissioned BCTs require a trusted authority, while permissionless ones can operate in zero-trust environments. For the latter, the ability to tolerate such a hostile environment comes at a price: validators must enact the *Proof of Work* consensus approach, making them compete to solve a computational puzzle in order to acquire the right to update the ledger. Such an approach is known to employ a considerable amount of energy and computational resources. Furthermore, it heavily hinders the throughput of applications built on top of it, and requires a rewards scheme to be created to incentivize validators.

In case a permissioned BCT is chosen, a Certification Authority (CA) is needed to endow devices with their identities. This means that a third set of machines, namely the *certifiers*, is in charge of running and maintaining the Certification Authority. The adoption of a CA, in turn, calls for a strategy in distributing identity certificates to devices, or revoking them as soon as a device must be evicted from the system because of a provably malicious behaviour.

In case a permission*less* BCT is selected, instead, no CA is needed, which seems a better choice given the inherently decentralised nature of Aggregate Programming. Nevertheless, such sort of BCTs strongly rely on cryptocurrencies and their intrinsic economical value to deliver security properties. Accordingly, clients must own an amount of cryptocurrency to be spent to issue transactions—that in turn feed the miners' rewards. This, in turn, implies that the applications built on top of permissionless BCTs, there including aggregate programs, cannot ignore cryptocurrencies. Of course, cryptocurrencies may be of no interest to devices or end users, therefore the burden they bring along must be minimised. A strategy should then be defined in order to make devices automatically handle cryptocurrencies on behalf of their users, similarly to what "Bitcoin wallet" applications do. In particular, devices may be endowed with some cryptocurrency as soon as the aggregate software is installed, and such cryptocurrency may be consumed by the ordinary operations such as messaging (which, recall, happens through transactions). In order to attain a sufficient amount of cryptocurrency, the end users interested in using the application may be required, depending on the application, to either pay a fee or subscribe to a plan, or, if the application provider is interested in spreading the application – as it could be the case, for instance, for crowd steering public safety applications –, it may convert some real-world money into cryptocurrency, distributing it to its users.

Throughput is the critical parameter that differentiates permissioned from permissionless BCTs, and is especially important in the context of Aggregate Programming. Permission*less* BCTs introduce an upper bound to the amount of transactions being published per time unit, due to the Proof of Work algorithm exploited, limiting in practice the evolution speed for the aggregate system, which may or not be compatible with the application requirements [27]. It is worth noting that, in turn, this poses an upper bound to *scalability*: a higher number of devices implies more transactions per time unit (assuming devices to compute rounds at similar and approximately constant frequencies), and once such a number gets past the limit imposed by the permissionless blockchain of choice, some transactions will be inevitably lost. Countermeasures may include limiting the round frequency to take into account such bound, or limiting

the number of participants to the system. The former comes with the undesired side effect that the aggregate computation gets slower with the number of participants, while the latter goes against most application scenarios suitable for Aggregate Computing, which usually assume open systems.

5 CONCLUSION AND FUTURE WORK

In this paper, we introduce a system architecture for transparently protecting aggregate programs from Byzantine behaviours by employing the Blockchain. The security layer integrates the trust protocols existing at application level, adding resilience to behaviours non consistent with the operational semantics of aggregate programs. We discuss how an attacker may use Byzantine behaviours to disrupt the system functionality despite the presence of state-of-art application-level protection, and moreover, how it can actually exploit it to exclude legitimate participants from the aggregate computation. We then describe how to mitigate such threats by leveraging the Blockchain technology as the communication infrastructure for aggregate programming systems. We focus on how three type of attacks (Byzantine, masquerade, and Sybil) cannot be counteracted by current application-level, trust-based mechanisms, and how they can instead be tackled by the novel infrastructural layer, with no impact on the business logic—indeed, transparently. Finally, we delve deeper into the trade-offs of the proposed solution, and in particular: the requirement of a certifier in case of permission*ed* blockchain; and the limitations on throughput as well as the requirement for a cryptocurrency-like reward for miners in case of permission*less* blockchain.

Future work includes a prototypical back-end implementation with existing or ad-hoc BCTs, as well as further extension to our proposed architecture letting the blockchain subsystem validate devices' neighbourhoods as well. In both cases, a more detailed discussion about the best-suited BCT among the many available ones is a necessary step. Furthermore, we are evaluating the possibilities offered by combining BCTs and decentralised reputation protocols (such as BackFeed[3]) to take, at least partially, reputation and trust into account directly below the application layer.

REFERENCES

[1] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *13th EuroSys Conference (EuroSys '18)*. ACM, Article 30, 15 pages. https://doi.org/10.1145/3190508.3190538

[2] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. 2015. The Next 700 BFT Protocols. *ACM Transactions on Computer Systems* 32, 4 (Jan. 2015), 1–45. https://doi.org/10.1145/2658994 arXiv:1710.03346

[3] Giorgio Audrito, Ferruccio Damiani, Mirko Viroli, and Roberto Casadei. 2016. Run-Time Management of Computation Domains in Field Calculus, See [16], 192–197. https://doi.org/10.1109/FAS-W.2016.50

[4] Jonathan Bachrach, Jacob Beal, and James McLurkin. 2010. Composable continuous-space programs for robotic swarms. *Neural Computing and Applications* 19, 6 (2010), 825–847. https://doi.org/10.1007/s00521-010-0382-8

[5] Adam Back. 2002. Hashcash - A Denial of Service Counter-Measure. http://www.hashcash.org/hashcash.pdf

[6] Jacob Beal, Danilo Pianini, and Mirko Viroli. 2015. Aggregate Programming for the Internet of Things. *IEEE Computer* 48, 9 (2015), 22–30. https://doi.org/10.1109/MC.2015.261

[7] Jacob Beal, Kyle Usbeck, Joseph Loyall, Mason Rowe, and James Metzler. 2016. Adaptive Task Reallocation for Airborne Sensor Sharing, See [16], 168–173. https://doi.org/10.1109/fas-w.2016.46

[8] Jacob Beal, Mirko Viroli, Danilo Pianini, and Ferruccio Damiani. 2017. Self-Adaptation to Device Distribution in the Internet of Things. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 12, 3, Article 12 (Sept. 2017), 29 pages. https://doi.org/10.1145/3105758

[9] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. 2005. Gossip algorithms: design, analysis and applications. In *IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 3. IEEE, 1653–1664. https://doi.org/10.1109/INFCOM.2005.1498447

[10] Christian Cachin and Marko Vukolić. 2017. Blockchain Consensus Protocols in the Wild. In *31st International Symposium on Distributed Computing (DISC 2017) (Leibniz International Proceedings in Informatics (LIPIcs))*, Andréa W. Richa (Ed.), Vol. 91. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, Article 1, 16 pages. https://doi.org/10.4230/LIPIcs.DISC.2017.1

[11] Roberto Casadei, Alessandro Aldini, and Mirko Viroli. 2018. Towards Attack-Resistant Aggregate Computing Using Trust Mechanisms. *Science of Computer Programming* 167 (2018), 114–137. https://doi.org/10.1016/j.scico.2018.07.006

[12] Roberto Casadei, Giancarlo Fortino, Danilo Pianini, Wilma Russo, Claudio Savaglio, and Mirko Viroli. 2019. Modelling and Simulation of Opportunistic IoT Services with Aggregate Computing. *Future Generation Computer Systems* 91 (Feb. 2019), 252–262. https://doi.org/10.1016/j.future.2018.09.005

[13] Roberto Casadei, Danilo Pianini, and Mirko Viroli. 2016. Simulating Large-scale Aggregate MASs with Alchemist and Scala. In *2016 Federated Conference on Computer Science and Information Systems, FedCSIS 2016, Gdańsk, Poland, September 11-14, 2016. (ACSIS)*, Vol. 8. Polish Information Processing Society, 1495–1504. https://doi.org/10.15439/2016F407

[14] Jin-Hee Cho, Ananthram Swami, and Ray Chen. 2011. A survey on trust management for mobile ad hoc networks. *IEEE Communications Surveys & Tutorials* 13, 4 (2011), 562–583.

[15] John R. Douceur. 2002. The Sybil Attack. In *1st International Workshop on Peer-to-Peer Systems (IPTPS '01)*. Springer, 251–260. http://dl.acm.org/citation.cfm?id=687813 Revised Papers.

[16] Sameh Elnikety, Peter R. Lewis, and Christian Müller-Schloer (Eds.). 2016. *1st International Workshops on Foundations and Applications of Self* Systems (FAS*W 2016)*. IEEE.

[17] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4, 3 (July 1982), 382–401. https://doi.org/10.1145/357172.357176

[18] Félix Gómez Mármol and Gregorio Martínez Pérez. 2009. Security threats scenarios in trust and reputation models for distributed systems. *computers & security* 28, 7 (2009), 545–556.

[19] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. http://bitcoin.org/bitcoin.pdf

[20] Danilo Pianini, Jacob Beal, and Mirko Viroli. 2016. Improving Gossip Dynamics Through Overlapping Replicates. In *Coordination Models and Languages (LNCS)*, Alberto Lluch Lafuente and José Proença (Eds.), Vol. 9686. Springer, 192–207. https://doi.org/10.1007/978-3-319-39519-7_12

[21] Danilo Pianini, Mirko Viroli, and Jacob Beal. 2015. Protelis: practical aggregate programming. In *30th Annual ACM Symposium on Applied Computing (SAC 2015)*. ACM Press, 1846–1853. https://doi.org/10.1145/2695664.2695913

[22] Fred B. Schneider. 1990. Implementing Fault-tolerant Services Using the State Machine Approach: A Tutorial. *Comput. Surveys* 22, 4 (Dec. 1990), 299–319. https://doi.org/10.1145/98163.98167

[23] Robert W. Shirey. 2007. Internet Security Glossary, Version 2. RFC 4949. https://doi.org/10.17487/RFC4949

[24] Mirko Viroli, Giorgio Audrito, Ferruccio Damiani, Danilo Pianini, and Jacob Beal. 2016. A Higher-order Calculus of Computational Fields. *CoRR* abs/1610.08116 (2016), 39. http://arxiv.org/abs/1610.08116

[25] Mirko Viroli, Jacob Beal, Ferruccio Damiani, Giorgio Audrito, Roberto Casadei, and Danilo Pianini. 2018. From Field-Based Coordination to Aggregate Computing. In *Coordination Models and Languages*, Giovanna Di Marzo Serugendo and Michele Loreti (Eds.). LNCS, Vol. 10852. Springer, 252–279. https://doi.org/10.1007/978-3-319-92408-3_12

[26] Mirko Viroli, Roberto Casadei, and Danilo Pianini. 2016. On execution platforms for large-scale aggregate computing. In *2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 1321–1326. https://doi.org/10.1145/2968219.2979129

[27] Marko Vukolić. 2016. The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication. In *Open Problems in Network Security*. LNCS, Vol. 9591. Springer, 112–125. https://doi.org/10.1007/978-3-319-39028-4_9

[28] Gavin Wood. 2014. Ethereum: a secure decentralised generalised transaction ledger. http://ethereum.github.io/yellowpaper/paper.pdf

[29] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. 2017. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. In *2017 IEEE International Congress on Big Data*. IEEE, 557–564. https://doi.org/10.1109/BigDataCongress.2017.85

[3] http://backfeed.cc/