Do Byzantine-Tolerant CRDTs Matter?

On Conflict-Freedom, Equivocation Tolerance, and the Matrix Replicated Data Type

Anonymous Authors[1]

Abstract: Conflict-free replicated data types (CRDTs) refer to replicated data types that allow updates to be applied to different replicas independently and concurrently — without the need for a remote conflict resolution. Thus, whenever CRDTs can be applied, they provide a building block for scalability and performance of fault-tolerant systems. Currently, CRDTs are typically used in a crash fault setting for global-scale, partition-tolerant, highly-available databases or collaborative applications like shopping lists and collaborative text editors. In this paper, we explore the notion of CRDTs in a setting with Byzantine processes. This exploration is inspired by the replicated data type used in the popular Matrix messaging system: as recently shown, the underlying Matrix Event Graph replicated data type represents a CRDT with optimal Byzantine fault tolerance. However, it is currently unclear to which class of applications this Byzantine-tolerant CRDT can be generalized. We, therefore, first discuss in which setting the combination of Byzantine faults and CRDTs might be 'well-defined' and how the notion of conflicts relates to equivocation.

We, then, show that a subclass of CRDTs is equivocation-tolerant, i.e., without equivocation detection, prevention or remediation, they still fulfill strong eventual consistency (SEC), and can thereby cope with any number of Byzantine faults. We also conjecture that there is only one operation-based CRDT design supporting non-commutative operations — namely the one used in the Matrix messaging system — that fulfills SEC in Byzantine environments with any number of faults. The corresponding CRDT does obviously neither provide consensus nor total order. We close the paper with thoughts on chances and limits of this potentially unique data type.

Keywords: Dependable Distributed Protocols; Conflict-Free Replicated Data Types; Equivocation Tolerance; Byzantine Fault Model; Matrix Event Graph

1 Introduction

Conflict-free Replicated Data Types (CRDTs) are data structures that are replicated among multiple processes. To keep all replicas up to date about local changes to the CRDT, replicas repeatedly broadcast updates to all replicas. As the name suggests, CRDTs provide powerful properties: in particular, updates can be applied without further coordination of replicas, and recovery from network partitions can be done with ease. Therefore, CRDTs are popular for global-scale, partition-tolerant, highly-available databases or peer-to-peer

[1] Example Institute, Example Faculty, Example University, 12345 Example City, Example Country firstname.lastname@example.org

collaborative applications like shopping lists and collaborative text editors, in a crash fault setting. As a potential 'conflict' depends on the underlying fault model, we are interested in a generalized understanding of CRDTs also in a Byzantine environment where faulty replicas can equivocate. In particular, we are interested in the characterization of Byzantine-tolerant CRDTs as well as in the relevance of these CRDTs. This interest is currently shared by various researchers [KH20, Au21] and also inspired by the success of the Matrix messaging system [Th21] that is based on the Matrix Event Graph replicated data type, a CRDT in a Byzantine environment [Ja21].

What does a Byzantine environment mean for CRDTs? Recent works on CRDTs in Byzantine environments have followed different paths. The work stretches from classical assumptions of an honest two-thirds majority [Zh16] introducing coordination mechanisms (e.g., coordinated Byzantine-tolerant causal-order broadcast [Au21]) to coordination-free, Sybil-resistant CRDTs using broadcast based on the happened-before relation as directed, acyclic graphs [KH20, Ja21].

In this paper, we take up the latter approach that does neither depend on additional coordination mechanisms nor on a particularly strengthened broadcast. We relate the notion of equivocation to crash-tolerant CRDTs and show under which conditions a subclass of crash-tolerant CRDTs is equivocation-tolerant in Byzantine environments. We show that, due to equivocation tolerance, all state-based and a subclass of operation-based CRDTs in the crash fault model tolerate any number of Byzantine faults while depending on rather mild assumptions on the communication layer. Further, we conjecture that the Matrix Event Graph is the only operation-based CRDT with non-commutative operations that provides the Strong Eventual Consistency (SEC) guarantee for any number of Byzantine faults.

Do Byzantine-tolerant CRDTs matter? As "safety does not guard against faulty clients" [Ca99, Section 3], even if Byzantine faults can be tolerated on a CRDT level, the application built on top of a Byzantine-tolerant CRDT has to be able to handle the resulting service guarantees, namely SEC. In this paper, we like to start the discussion on the chances and limits of Byzantine-tolerant CRDTs deployments. Exemplary for Byzantine-tolerant CRDTs, the Matrix Event Graph looks like a graph-based groww-only logging mechanism of the causal history of published messages. As CRDTs provide neither consensus nor total order, depending on the use case, one might need some form of access control and/or combination with additional decision mechanisms as well.

The structure of the paper is as follows: In Sect. 2, we motivate the case for Byzantine-tolerant CRDTs by the Matrix messaging use case and start the discussion on the use of Byzantine-tolerant CRDTs in other applications contexts. In Sect. 3 and 4, we provide terminology, assumptions, and the technical characterization of Byzantine-tolerant CRDTs as well as a conjecture. In Sect. 5, we take up the question on the relevance of Byzantine-tolerant CRDTs based on the technical characterization and address open issues.

2 Byzantine-Tolerant CRDTs and their Application Context

2.1 Matrix Event Graph

The Matrix Event Graph (MEG) is the replicated data type underlying the Matrix messaging system. As of today, Matrix is primarily used for decentralized instant messaging, e.g., by the French public sector and the German military forces. Matrix is also the upcoming standard in the German healthcare sector [Ho21b]. As of October 2021, the public Matrix federation consists of more than 35 million users and 70 thousand servers [Ho21a].

The MEG represents a CRDT that actually is conflict-free not only in the crash fault model, but also in the byzantine fault model [Ja21]. Therefore, the MEG serves as our prime example for Byzantine-tolerant CRDTs in this paper. Below we will outline why Byzantine behavior, in particular equivocation, does not 'hurt' the MEG. However, due to space restrictions, we only present a very short introduction to this data type and its use.

Essentially, a MEG is a graph-based grow-only logging mechanism of the causal history of published messages [Th21] that can be considered a *HashDAG (Hash-based Directed Acyclic Graph)*, in analogy to a hash chain in blockchains. On publishing a new message (event), the sending replica appends the new message to all currently known messages without children, i.e., the graph represents the happened-before relation, also called the (potential) causal order. Fig. 1 shows an example causal history of a replica. The sending replica then broadcasts an update operation consisting of the new message and *content identifiers* (hashes) of the operations which appended the causal parent messages to the graph. In the MEG, cryptographic hashes are used as content identifiers to ensure the integrity against Byzantine replicas of the relation between operation and identifier, as well as the happened-before relation: when message β is able to reference the hash of message α, α must have happened before β. Therefore, the MEG can be considered a kind of HashDAG.

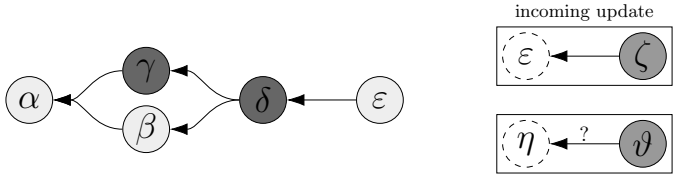

Fig. 1: MEG of one replicas with messages from itself (○) and from other replica (●). It receives two updates from a third replica (●): The update with message ζ can be applied immediately, since its causal parent ϵ is already known. The update with ϑ cannot be applied (yet) since η is not part of replica's MEG.

As depicted in Fig. 1, the happened-before relation allows receiving replicas to detect faults: If its parents are not yet known, a new message is not appended to the graph. Instead, other replicas can be queried for the missing operations. Through the hash-based content identifiers, replicas can verify the integrity of replies even from Byzantine replicas, and will eventually receive the operation if any correct replica received it.

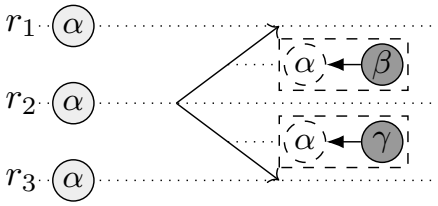

Fig. 2: An example for an equivocation performed by replica r_2: r_2 created two updates that both reference the same message α. r_2 sends update with message β to r_1 and update with message γ to r_3, where it should have sent only one to both.

Due to its construction, the MEG can tolerate Byzantine behavior of senders. Messages with made-up parents will never be added to the log of a correct replica. As shown in Fig. 2, a Byzantine replica (r_2) can perform equivocation: It sends message β to r_1 and message γ to r_2 where it should have sent only one to both. As different updates have different content identifiers, as soon as correct replicas r_1 and r_3 communicate with each other, they can reliably detect that their graphs differ by comparing content identifiers of received updates. Because the DAG only implies a partial order, equivocated updates can be treated and merged as two causally independent, concurrent updates, and consistency is restored.

2.2 Application Context

A Byzantine-tolerant CRDT ensures on the CRDT layer, i.e., the layer below the application, that Byzantine replicas which deviate from the CRDT protocol do not violate the properties of the CRDT (the properties are given in Sect. 3). In Sect. 4 we present the technical characterization of Byzantine-tolerant CRDTs. A somewhat different question is which applications would work reasonably well with the guarantees provided by Byzantine-tolerant CRDTs — and whether Byzantine behavior can be dealt with on application layer.

In classical CRDT use cases in the crash fault model, all parties have the permission to perform any CRDT operations, including deletion. Thus, on the technical CRDT layer, deleting is not regarded as a Byzantine act. On the application layer, however, deletion might or must be considered Byzantine behavior. To avoid this issue, when dealing with CRDTs in Byzantine environments, one would typically focus on grow-only CRDTs. A grow-only CRDT only supports ‘append’ operations, and can thereby treat a byzantine replica that deleted local state just like a correct replica that has not yet received the state without harm.

Beyond grow-only CRDTs, access control can be used to restrict operations for specific parties. However, this requires a verifiable caller of individual operations. We take up this discussion in Sect. 5 and will show that the differentiation into state-based and operations-based CRDTs is crucial when addressing the questions of “do byzantine-tolerant CRDTs matter?”. For now, we like to emphasize that CRDTs inherently forfeit consensus and total order in favor of availability and partition tolerance.

3 Terminology and Assumptions

We assume an asynchronous network with a static set of n processes participating in the CRDT. The processes are connected with authenticated channels and all correct processes form a connected component in the communication graph. Every update that is sent over a channel can be arbitrarily delayed, but is received eventually on the other side of the channel. Typically, CRDTs assume that processes can only fail by crashing. However, in Sect. 4, we assume that processes can also fail arbitrarily, that is, behave *Byzantine*.

A *Conflict-Free Replicated Datatype* is a data structure that is maintained by the processes, also called *replicas*. Replicas execute update operations to change the state of the CRDT. Replicas broadcast information on the current state of their local copy of the CRDT. Without coordination or conflict resolution between replicas, a CRDT ensures *Strong Eventual Consistency* (SEC). Intuitively, SEC ensures that correct replicas eventually converge to a consistent state, regardless of the order in which updates were received. SEC consists of the following properties:

Eventual Delivery: If an update is applied at some correct replica, it is eventually applied at every correct replica.
Termination: Every operation that is executed by a replica eventually terminates.
Strong Convergence: Correct replicas that applied the same set of updates maintain the same state.

CRDTs are either *state-based* or *operation-based*. With state-based CRDTs, to fulfill SEC, replicas repeatedly broadcast the current state of their local copy of the CRDT to the other replicas, which can merge the received state with their local state. All states of the CRDT form a *semilattice*. A semilattice is a partially ordered set, and every possible CRDT state is one element of the set. Every pair of states has a least upper bound, which is also called the *join* of two states. An update is *valid* if it is part of the semilattice. If a replica r receives a valid state from another replica p, p's state can be directly applied by merging r's state with p's state using the join operation of the semilattice.

Operation-based CRDTs differ from state-based CRDTs in that replicas do not send their whole new state as updates but only the operation that lead to the new state. To fulfill SEC, the updates must either be delivered in causal order or be commutative. Usually, the causal order is enforced with an underlying causal order broadcast. We consider a different approach to enforcing the causal order: By including the happened-before relation via hash-based content identifiers as part of the updates themselves, the CRDT can apply and verify the causal order in a byzantine fault tolerant way (cf. Sect. 4.2).

Replicas only apply *valid updates*. A valid update is an update that is protocol-conforming when viewed on its own. An invalid update violates locally verifiable protocol properties and cannot be applied to the CRDT. A *conflict* occurs when two concurrent updates that may be individually *valid* but, when viewed together, violate some invariant [Sh11, Footnote 1].

In crash fault environments, conflicts refer to involuntary violations of invariants, e.g., violation of the uniqueness of update identities due to a hash collision or concurrent writes. In contrast, Byzantine replicas may try to violate any invariant, even ones that would never be violated by protocol-conforming replicas. Invariants that only exist on the application layer are considered out of scope and are discussed in Sect. 5. in the crash fault model.

In the Byzantine fault model, *Equivocation* is the act of a Byzantine replica to send different valid updates to different recipients, where it should have sent the same update [Ch07], also depicted in Fig. 2. In contrast to an invalid update, which is detectable when viewed on its own, equivocated updates can only be detected globally or with both equivocated updates. Equivocated updates always originate from a Byzantine replica, and do not exist in the crash fault model. Please note that in the scope of this paper, equivocated updates are not necessarily conflicting, i.e., with respect to the invariants of the technical CRDT layer. The messages might still conflict on the application layer, consider for example the classical "Attack!" and "Retreat!" conflict [LSP82].

Omission occurs if a crash- or Byzantine-faulty replica sends an update to only a strict subset of all protocol-intended recipients. In crash fault environments, omission can occur involuntarily when a replica crashes during broadcasting an update. In Byzantine environments, omissions can also happen intentionally.

We say that a system is *equivocation-tolerant* if it neither needs to detect, prevent, nor remedy equivocation to ensure its provided guarantees beyond what is needed to cope with omission.

4 CRDTs and Equivocation Tolerance

The notion of CRDTs was originally defined to guarantee SEC in a crash fault environment [Sh11]. In this section, we analyze under which conditions CRDTs still provide their SEC guarantee in Byzantine environments. As a Byzantine process might equivocate, we have to analyze the ability of a CRDT to handle equivocated updates as non-conflicting in Byzantine settings. Equivocation itself mainly threatens the Strong Convergence property: either the notion of which updates are the same, or the application order of updates can be equivocated. Thus, conflict freedom in a Byzantine setting requires (implies) the property of equivocation tolerance: any valid update needs to be applicable without an extra non-local equivocation detection, prevention, or remediation. When equivocation tolerance can be assumed, a replicated data type that is conflict-free in the crash fault model and transferred to a Byzantine environment has to deal additionally only with omission faults. However, as we will see below, omission faults will not threaten the property of Termination. Thus, equivocation tolerance represents a key feature of a CRDT in a Byzantine setting. Below, we show that all state-based CRDTs and some specific operation-based CRDTs are equivocation-tolerant. As equivocation tolerance essentially means that equivocations cannot harm the replicated data type, and Byzantine processes can basically only equivocate

(on the level of the replicated data type) when not following the protocol, we show that equivocation tolerance also implies fault tolerance for any number of Byzantine faults.

4.1 State-based CRDTs and Equivocation Tolerance

Proposition 4.1. *All state-based CRDTs in the crash fault model also trivially provide equivocation tolerance in Byzantine environments.*

Sketch of proof. State-based CRDTs are based on a defined join-semilattice of all valid states. Replicas of a state-based CRDT in the crash fault model only send their current state without metadata, which means that an update is valid if and only if it is part of the semilattice, which is locally verifiable. Due to the commutativity of the join relation and the partial order of the semilattice, any two valid updates cannot conflict with each other, as both can be merged in an arbitrary order with the same result. Thus, in case a replica wants to equivocate, all the replica can do is to send different valid updates that will not conflict, by definition of the state-based CRDT in the crash fault model. Accordingly, an equivocation consisting of d differing updates can be treated as d independent updates for which omission has occurred.

Corollary 4.2. *State-based CRDTs in the crash fault model also ensure Strong Eventual Consistency for all correct replicas in an environment with n replicas of which an arbitrary number f exhibit Byzantine faults, i.e., have fault tolerance $n > f$.*

Sketch of proof. Due to Proposition 4.1, we can treat equivocation as omission, i.e., an update was not sent to all other replicas. State-based CRDTs gossip their current state regularly and unsolicitedly to all other replicas. With the given system model, every update that is sent to another replica is received eventually by this replica. As the replicas' current state indirectly contains all updates they have received and merged before, updates not sent directly to a specific replica will eventually reach that replica indirectly via correct replicas. Thus, also Eventual Delivery and Termination are not violated.

4.2 Operation-based CRDTs and Equivocation Tolerance

While state-based CRDTs need no further assumptions to work in Byzantine environments with any number of faults, only certain operation-based crash-tolerant CRDTs can do so, and they need a number of additional assumptions.

Operation-based CRDTs require that non-commutative operations are applied in causal order. To avoid conflicts due to equivocation on non-commutative operation ordering, we consider the 'non-standard' way to enforce the causal order found in the Matrix Event

Graph: The datatype semantics inherently records the causal order (*inherent ordering*). By extending an update with its happened-before relation via content identifier using its causal parents, the receiving replica can ensure to apply non-commutative updates in causal order. If implemented via hashing, Byzantine attackers cannot tamper with the happened-before relation, as hashes verifiably prove that an operation referenced by another operation happened before the other operation.

To avoid other conflicts due to equivocation, content identifiers also provides an *inherent identity* to a valid update operation: the verifiable unique identifier of an operation is the hash of its content. Any two operations that are the same as defined by the protocol need to be idempotent, i.e., lead to the same outcome when applied. Thereby, identical operations are not applied twice when received twice.

Proposition 4.3. *Operation-based CRDTs in the crash fault model require inherent identity of operations and inherent ordering of operations to be equivocation-tolerant in any Byzantine environment.*

Sketch of proof. Operation-based CRDTs rely on operation metadata, especially operation identifiers and happened-before relations. The uniqueness of operation identifiers and correctness of happened-before relations are invariants whose violation by Byzantine replicas would violate Strong Convergence, and thereby SEC. Without inherent identity, a Byzantine replica can equivocate by sending two different updates with the same identifier to different replicas. Without inherent ordering, a Byzantine replica can equivocate by sending two versions of two non-commutative updates with converse causal order. In both cases, the conflicting updates would be applied and lead to an inconsistent state, without a coordination-free way for the receiving replicas to detect or remedy. With inherent identity and inherent operation, one of any pair of conflicting updates that would violate identifier uniqueness / happened-before correctness is invalid, i.e., can be locally detected and rejected by correct replicas without coordination. It follows that both inherent identity as well as inherent ordering is required for operation-based CRDTs to be equivocation-tolerant.

Without periodic gossiping of state-based CRDTs, operation-based CRDTs need to be able to handle omissions to ensure Eventual Delivery. If a happened-before relation is included in update operations, then *Omission Handling* can rely on the relation to detect missing operations. Using hash-based content identifiers, those missing operations can be re-requested verifiably from other replicas. Alternatively, CRDTs can periodically gossip the set of all received operations. The gossiping approach can be formalized and made more efficient through the happened-before relation and hash chaining [KH20]. We note that this approach essentially uses a state-based set CRDT to synchronize all operations, benefiting from the Byzantine tolerance of all state-based CRDTs shown in Corollary 4.2.

Corollary 4.4. *Omission-handling, equivocation-tolerant operation-based CRDTs ensure Strong Eventual Consistency for all correct replicas in an environment with n replicas of which an arbitrary number f exhibit Byzantine faults, i.e., have fault tolerance $n > f$.*

Sketch of Proof. Proposition 4.3 allows to treat equivocation as omission. For CRDTs that use an omission handling mechanism (e.g., one of the approaches explained above), Byzantine replicas cannot prevent that updates they sent to at least one correct replica are eventually delivered to all correct replicas, i.e., they cannot harm the Eventual Delivery property through omission.

4.3 Uniqueness Conjecture

In environments with Byzantine majorities, a CRDT with non-commutative operations has to record the happened-before relation in the data type to ensure the causal order independently of the broadcast order. The only way known to us to ensure the causal order in a locally verifiable, Byzantine-tolerant way is to use hash chaining. The happened-before relation being a partial order inherently leads to a directed, acyclic graph (DAG) of all operations. To efficiently ensure Eventual Delivery, one can employ the happened-before relationship recorded in the graph by re-requesting missing parent operations and only gossiping childless operations. This line of thought leads us to the conjecture:

Conjecture 4.1. *The only operation-based CRDT with non-commutative operations that provides SEC for any number of Byzantine faults, i.e., has fault tolerance $n > f$, is a HashDAG as described in [KH20, Ja21] and implemented in the Matrix Event Graph [Th21].*

5 Discussion and Conclusion

We analyzed the reasons why and under which circumstances a subclass of CRDTs is not only crash-tolerant, but also Byzantine-tolerant. For the analysis we made use of the notion of equivocation tolerance and its relation to conflict freedom. We showed that even when faced with an arbitrary number of Byzantine faults, this subclass can keep the characteristic traits of CRDTs, like efficiency, low coordination effort, and Strong Eventual Consistency.

We now like to take up the question in the title of this paper and the aspects addressed in Sect. 2 by looking to the current practical relevance of Byzantine-tolerant CRDTs as well as to potential combinations with access control and/or further coordination mechanisms.

First of all, the Matrix Event Graph with its HashDAG construction proves the practical relevance of Byzantine-tolerant CRDTs for the use case of instant messaging. But is there a relevance for other use cases than the Matrix one? With state-based CRDTs, the replicas propagate their current state in the system, making them easy to deploy in Byzantine environments, as we showed in Corollary 4.2, based on their unconditional equivocation tolerance. However, the identity of updates gets lost since only states are propagated in the system. It is not possible to reconstruct the update that led to a new state, which makes it impossible to prove which replica performed which CRDT updates and whether it was allowed to do so. Hence, access control on the different access and update operations of

state-based CRDTs, giving different permissions to different participating replicas, cannot be enforced. Therefore, state-based CRDTs might be suitable for decentralized systems in the spirit of the Newsgroup system where any user can write new articles and reply to old ones. The existing Newsgroup system is susceptible to equivocation, as users are trusted to not assign the same identifier to different articles. If based on an equivocation-tolerant CRDT, SEC could be guaranteed. In contrast to state-based CRDTs, as we showed in Corollary 4.4, operation-based CRDTs require additional properties to provide SEC in Byzantine environments. However, with operation-based CRDTs, the original caller of an update operation can be determined and verified with authentication mechanisms like digital signatures. Therefore, operation-based CRDTs provide the necessary prerequisites for access control, which makes them easier to deploy in more demanding systems.

In general, CRDTs inherently forfeit consensus and coordination in favor of availability and partition tolerance. Without any form of Sybil countermeasures like controlled membership, the space of solvable application problem is the class of invariant-convergent problems, i.e., invariants for which local, uncoordinated replica decisions are sufficient to preserve the invariants globally [KH20]. While this takes cryptocurrencies out of question, the typical CRDT use cases for which Strong Eventual Consistency suffices, e.g., collaborative applications like shopping lists, text editors or whiteboards, are also invariant-convergent and, therefore, uses cases for Byzantine-tolerant CRDTs.

When stronger guarantees are required, e.g., strong consistency, Byzantine-tolerant CRDTs can obviously only serve as part of a solution and need to be combined with other, stronger mechanisms. While the combination 'technically' inherits the stronger assumptions, from a practical deployment perspective, a hybrid mode of operations might make sense: an application can make use of the Byzantine-tolerant CRDT to collect data only requiring corresponding weak assumptions on the system. Whenever stronger assumptions like synchronous operation are fulfilled at certain phases in time, more demanding tasks like consensus can be performed. In other words, a Byzantine-tolerant CRDT can be used in a reactive system that in general simply does robust instant messaging but whenever the system is in favorable conditions, it builds consensus. Thus, the resulting hybrid system would fall into the category of partially synchronous system [DLS88], reaching agreement on previously aggregated updates.

To conclude, Byzantine-tolerant CRDTs definitely play a significant role as a dependable distributed protocol for robustly collecting data as it is required in an (instant) messaging system. A role further than this one depends on the validity of the conjecture we have stated in this paper as well as on the exploration of the extensibility of this replicated data type. We hope that the technical characterization of Byzantine-tolerant CRDTs we presented in this paper provides a basis for such a future exploration.

Bibliography

[Au21] Auvolat, Alex; Frey, Davide; Raynal, Michel; Taïani, François: Byzantine-tolerant causal broadcast. Theoretical Computer Science, 2021.

[Ca99] Castro, Miguel; Liskov, Barbara et al.: Practical byzantine fault tolerance. In: OSDI. volume 99, pp. 173–186, 1999.

[Ch07] Chun, Byung-Gon; Maniatis, Petros; Shenker, Scott; Kubiatowicz, John: Attested append-only memory: Making adversaries stick to their word. ACM SIGOPS Operating Systems Review, 41(6):189–204, 2007.

[DLS88] Dwork, Cynthia; Lynch, Nancy; Stockmeyer, Larry: Consensus in the presence of partial synchrony. Journal of the ACM (JACM), 35(2):288–323, 1988.

[Ho21a] Hodgson, Matthew: , Element raises $30M to boost Matrix. https://matrix.org/blog/2021/07/27/element-raises-30-m-to-boost-matrix, 2021. The Matrix.org Foundation C.I.C.

[Ho21b] Hodgson, Matthew: , Germany's national healthcare system adopts Matrix! https://matrix.org/blog/2021/07/21/germanys-national-healthcare-system-adopts-matrix, 2021. The Matrix.org Foundation C.I.C.

[Ja21] Jacob, Florian; Beer, Carolin; Henze, Norbert; Hartenstein, Hannes: Analysis of the matrix event graph replicated data type. IEEE Access, 9:28317–28333, 2021.

[KH20] Kleppmann, Martin; Howard, Heidi: Byzantine eventual consistency and the fundamental limits of peer-to-peer databases. arXiv preprint arXiv:2012.00472, 2020.

[LSP82] Lamport, Leslie; Shostak, Robert; Pease, Marshall: The byzantine generals problem. ACM Transactions on Programming Languages and Systems, 4(3):382–401, 1982.

[Sh11] Shapiro, Marc; Preguiça, Nuno; Baquero, Carlos; Zawirski, Marek: Conflict-free replicated data types. In: Symposium on Self-Stabilizing Systems. Springer, pp. 386–400, 2011.

[Th21] The Matrix.org Foundation C.I.C.: Matrix specification v1. Technical report, 2021. https://matrix.org/docs/spec/.

[Zh16] Zhao, Wenbing: Optimistic byzantine fault tolerance. International Journal of Parallel, Emergent and Distributed Systems, 31(3):254–267, 2016.