IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

*Digital Object Identifier 10.1109/ACCESS.2017.DOI*

# Analysis of the Matrix Event Graph Replicated Data Type

**FLORIAN JACOB[1], CAROLIN BEER[2], NORBERT HENZE[3], AND HANNES HARTENSTEIN[4], (Member, IEEE)**

[1]KASTEL – Institute of Information Security and Dependability, Karlsruhe Institute of Technology, Karlsruhe, Germany (e-mail: florian.jacob@kit.edu)
[2]KASTEL – Institute of Information Security and Dependability, Karlsruhe Institute of Technology, Karlsruhe, Germany (e-mail: carolin.beer@student.kit.edu)
[3]Institute of Stochastics, Karlsruhe Institute of Technology, Karlsruhe, Germany (e-mail: henze@kit.edu)
[4]KASTEL – Institute of Information Security and Dependability, Karlsruhe Institute of Technology, Karlsruhe, Germany (e-mail: hannes.hartenstein@kit.edu)

Corresponding author: Florian Jacob (e-mail: florian.jacob@kit.edu).

**ABSTRACT** Matrix is a new kind of decentralized, topic-based publish-subscribe middleware for communication and data storage that is getting particularly popular as a basis for secure instant messaging. By comparison with traditional decentralized communication systems, Matrix replaces pure message passing with a replicated data structure. This data structure, which we extract and call the Matrix Event Graph (MEG), depicts the causal history of messages. We show that this MEG represents an interesting and important replicated data type for decentralized applications that are based on causal histories of publish-subscribe events: First, we prove that the MEG is a Conflict-Free Replicated Data Type for causal histories and, thus, provides Strong Eventual Consistency (SEC). With SEC being among the best known achievable trade-offs in the scope of the well-known CAP theorem, the MEG provides a powerful consistency guarantee while being available during network partition. Second, we discuss the implications of byzantine attackers on the data type's properties. We note that the MEG, as it does not strive for consensus or strong consistency, can cope with $n > f$ environments with $n$ participants, of which $f$ are byzantine. Furthermore, we analyze scalability: Using Markov chains, we study the number of forward extremities of the MEG over time and observe an almost optimal evolution. We conjecture that this property is inherent to the underlying spatially inhomogeneous random walk. With the properties shown, a MEG represents a promising element in the set of data structures for decentralized applications, but with distinct trade-offs compared to traditional blockchains and distributed ledger technologies.

**INDEX TERMS** Conflict-Free Replicated Data Type, Decentralized Systems, Distributed Computing, Eventual Consistency, Instant Messaging, Middleware, Publish-Subscribe, Scalability

## I. INTRODUCTION

MATRIX[1] is a public specification of protocols for a middleware that provides communication and data services for decentralized applications. Matrix implements topic-based publish-subscribe services based on a federated architecture. On the one hand, it is particularly popular as a basis for decentralized instant messaging since the Matrix servers of an organization are fully under the control of the organization, but can still federate with servers of other organizations. Prominent examples for private federations are the French government and public administration, and the Federal Defense Forces of Germany. On the other hand, the Matrix public federation has a fast-growing user base, currently with more than 25 million accounts. A prominent example of an organization that uses the public federation is the Mozilla foundation.

Similar to e-mail or XMPP, clients attach themselves to a Matrix server, their so-called homeserver, by which they are represented in the Matrix network. Servers with clients subscribed to a specific topic (called room in Matrix parlance) form a federation to exchange published events that are independent of other topics. Events can be either communication events or state update events on the stored data. In the instant messaging use case, topics are employed for group or one-to-one communication rooms, communication events are used for instant messages, and stored data is used for persistent information like room description or membership.

[1]https://matrix.org/, https:/matrix.org/spec/

In contrast to e-mail or XMPP, Matrix replaces pure message passing with a replicated, per-topic data structure that stores the history of events in causal order. As Matrix servers can thereby synchronize their room's full causal histories, the Matrix approach promises increased system resilience in a decentralized setup: After a network partition, a server has significantly stronger means to recover the complete state of the room, i.e., to avoid loss of events. While this increased level of system resilience has been observed by practitioners, the underlying replicated data type has not yet been analyzed thoroughly.

In this paper, we analyze the replicated data type of the Matrix approach and show that its properties make it a distinct member in the family of blockchain and distributed ledger data structures and of interest for researchers and practitioners alike. We first extract and abstract the replicated data type from the Matrix specification and denote it by Matrix Event Graph (MEG) in the following. A MEG is a Directed, Acyclic Graph (DAG) made up of vertices which represent communication and state update events, and directed edges which stand for potential causal relations between events. Because the graph represents the potential causal order of events, a correct graph is inherently acyclic. Adding new events is the only write operation supported by the MEG, which makes it an append-only data structure. Thus, the MEG can be considered as a fundamental concept for various applications that are based on causal histories, ranging from decentralized crowdsensing databases in Internet of Things scenarios through decentralized collaboration applications to decentralized push notification systems.

Since, for distributed ledger technologies, it has been conjectured that consistency, decentralization, and scalability in their 'strongest forms' cannot be achieved simultaneously [1], [2], our analysis focuses on these aspects and the distinct trade-offs made by a MEG: in brief, we show that the MEG achieves decentralization and scalability, but does not strive for consensus or strong consistency. As the main contribution, we therefore provide an analysis of the degree to which the MEG fulfills consistency, deployability in decentralized scenarios, and scalability:

CONSISTENCY: Since Matrix provides availability and partition tolerance, in accordance with the CAP theorem [3], the MEG necessarily has to sacrifice strong consistency. We show that the MEG provides Strong Eventual Consistency (SEC) by proving that the MEG is a Conflict-Free Replicated Data Type (CRDT) [4] for causal histories. We compare SEC to Eventual Consistency and Strong Consistency in Subsection IV-A.

DECENTRALIZATION: We discuss the implications of byzantine attackers on the specific type of CRDT that the MEG represents. The avoidance of consensus is the primary reason that allows the MEG CRDT to facilitate $n > f$ environments with $n$ total participants, of which $f$ exhibit byzantine faults.

SCALABILITY: The probabilism of uncoordinated, concurrent append updates represents the main challenge for the analysis of the MEG with respect to scalability. We are interested in the width of the MEG, which is represented by the number of forward extremities, i.e. 'vertices without children', over time. We study the width of the MEG using Markov chains. We observe that the MEG does not degenerate and conjecture that this non-degeneracy is inherent to the underlying spatially inhomogeneous random walk.

Thus, as outlined above, the aspects of this analysis are a theoretical model (the MEG) for a real-world system of relevance (Matrix), and proofs of properties of this model based on precise assumptions. To transfer these theoretical results back to a real-world system, we also clarify which assumptions currently hold in the Matrix system and how missing elements can be dealt with.

This paper is structured as follows: We begin with a note on Matrix (Section II) and an overview of how the MEG works and the statement of the problem in Section III. Section IV presents related work and background on replicated data types. Assumptions and architecture are given in Section V. In Section VI, we prove that a MEG is a Conflict-Free Replicated Data Type. In Section VII, we show the MEG can be made byzantine fault tolerant. We also perform a reality check by comparing Matrix and the utilized assumptions of the proofs. Section VIII formalizes the stochastic behavior of the width of the MEG and provides evidence that the width always evolves to a near-optimal value, and does so fast. We summarize limitations and open issues of our analysis in Section IX and conclude the paper in Section X.

## II. A NOTE ON MATRIX

The Matrix project started in September 2014, and it is governed by The Matrix.org Foundation C.I.C. A core team guides the evolution of Matrix by developing the open standard and the reference homeserver [5]. Matrix is the protocol employed by the Element instant messenger, a chat application for human communication in competition to, e.g., Slack, Microsoft Teams, Signal, and WhatsApp. While it features end-to-end encryption similar to Signal, it is mainly advertised for its fully decentralized group chats, which protocol-level contestants like XMPP or IRC do not provide. Today, Matrix is designed for a single dedicated server per user: For each topic, Matrix employs an instance of a resource-intensive full-mesh broadcast protocol, and, in addition, Matrix requires a valid Domain Name System record and matching X.509 certificate per server. Even with end-to-end encryption, sensitive metadata accumulating on the dedicated server is among the reasons why Matrix currently develops into the direction of a peer-to-peer protocol with Matrix servers on every device [6].

However, Matrix is not a chat protocol, but an extensible topic-based publish-subscribe system for the exchange of structured data that can also be used for human-machine and machine-machine communication. Matrix is designed with eventual synchronization of event histories as primary operation. While this matches the use case of instant messaging with reliable history synchronization, it might be

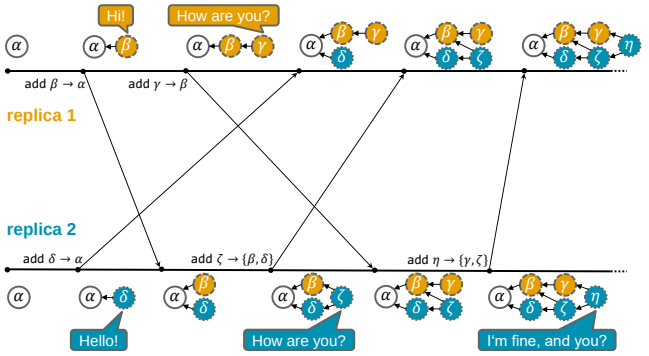


**FIGURE 1.** Basic example of the state evolution of two Matrix Event Graph (MEG) replicas over time, from left to right. Two MEG replicas start with an initial state, i.e. a root vertex $\alpha$. Both replicas concurrently append new vertices to the current state, as indicated by `add` *new vertex* $\to$ *parent vertices*. While the operation is immediately applied to the local state, the arrows between replicas represent the delay until the operation is applied on the remote replica. In the chat use case, every new vertex corresponds to a text message, depicted by chat bubbles. The timeline is presented in detail in Section III.

inefficient or problematic when a history is not needed or allowed. Matrix prioritizes best-effort, opportunistic progress of individual servers over coordination with all servers, and it might therefore be less suited for applications that require consensus or strong real-time guarantees.

## III. MEG: OVERVIEW & PROBLEM STATEMENT

In the following, we give an overview of the MEG concept as a replicated data type for append-only causal histories of events. A sample MEG with two replicas concurrently adding to the graph is illustrated in Fig. 1. For illustration purposes, we typically utilize the instant messaging use case of Matrix. We also typically focus our studies on a single MEG instance, and, therefore, on a single broadcast domain associated with that MEG. However, several independent MEGs can coexist, e.g., one per publish-subscribe topic.

A replicated data type is a data structure that is replicated across the peers of a network (in Matrix: the homeservers for a specific topic) and consists of a) a structure, b) a procedure to add an event, and c) a procedure to update all replicas, i.e. a way to deal with concurrency. Before we describe the general MEG concepts, we first give an example by describing the evolution of MEG replica states, as shown in Fig. 1.

MEG Example Timeline (as given in Fig. 1). Replica 1 starts by adding vertices $\beta$ and $\gamma$, before receiving anything from replica 2. Replica 2 adds vertex $\delta$ concurrently to the addition of $\beta$. In contrast, vertex $\zeta$ is added after $\beta$ is received, and therefore vertex $\zeta$ gets both $\beta$ and $\delta$ as parent. After vertices $\gamma$ and $\zeta$ have been eventually received, convergence is reached momentarily. Due to the concurrent additions, the number of forward extremities has increased to two. Then, replica 2 adds vertex $\eta$ without concurrent additions from replica 1, which reduces the number of forward extremities to one, and again convergence is reached.

General MEG structure. As mentioned before, a MEG is a Directed, Acyclic Graph (DAG). One MEG represents the message history and attributes of a group or 1:1 chat, and it is replicated independently by all participating servers. Upon creation, the DAG consists of only a single vertex, the *root vertex* (cf. $\alpha$ in Fig. 1). Each vertex in the DAG corresponds to an application-defined publish-subscribe event, e.g., to a text message or temperature reading. Edges represent potential causal relationships between events: When a new vertex is added, it is connected to the existing DAG through one or more outgoing edges. These edges point towards vertices that had no incoming edges before, i.e., the newest events in causal history, which we from now on call the *forward extremities* of the DAG. The selection of forward extremities is done according to the current state of the adding replica. This potential causal relationship is known as the *happened before* relationship[2], as defined by Lamport [7]: For $\alpha \leftarrow \beta$, we say that $\alpha$ happened before $\beta$. The edges thereby form a partial order that is consistent with the causal order in which events took place.

In addition to being directed, acyclic, and representing the causal order of events, the MEG is also weakly connected, since all newly added vertices have at least one outgoing edge. The root vertex, being the only vertex without outgoing edges, is therefore the unique minimal element of the partial order represented by the DAG. DAGs with this specific structure are also called *rooted DAG* [8].

Adding a new vertex to the source replica. The replica that creates an event on behalf of a client and appends it as a vertex is called *source replica*. When it adds a vertex, the corresponding event could be causally related to previous events. Thus, all forward extremities should be included as edges: As shown in Fig. 1, replica 2 includes both forward extremities $\beta$ and $\delta$ as parents for the new vertex $\zeta$. In practice, however, there are some issues: Replicas can experience a high number of forward extremities caused by latencies or partitions, and malicious replicas could forge events with a high number of parents. Certain algorithms executed on the MEG do not scale well with the number of parent events, i.e., they can become very resource intensive, especially when old parts of the MEG are referenced as parents [9]. Therefore, the maximum number of parent events is typically restricted to a finite value $d$. If there are more than $d$ forward extremities, a replica selects a subset of size $d$ for the new event. For the potential causal order relation in the MEG still to be consistent with the actual causal order, clients have to inform the replica about actual causal dependencies so that those are included as parents.

Updating all replicas. Beyond appending the new vertex to the local DAG, the source replica also needs to synchronize with the other replicas. The replica sends a DAG update that consists of the new vertex and edges to all replicas using a broadcast protocol. On receiving an update, replicas append the new vertex to their DAG via the new edges selected by the source replica, as soon as all required parent

[2]Note that Lamport defines $\alpha$ *happened before* $\beta$ as $\alpha \to \beta$. In this paper, we actually use the converse relation $\beta \to \alpha$, as common for distributed ledger technologies. It follows that for $\beta \to \alpha$, we say $\alpha$ is the parent of $\beta$.

vertices exist in the local replica. In case the parent vertices are not (yet) available, the update is buffered until they are.

DEALING WITH CONCURRENT UPDATES. When clients at two different replicas concurrently invoke updates, each replica considers their vertex as the single next step in causal history represented by their local DAGs. In case of continuous synchronization failure, e.g., due to a network partition, additional client updates will enlarge the inconsistency between the replicas' DAGs and will lead to two causally independent chains of events, built from the last synchronized event. Both replicas will continue to try to synchronize their state with other replicas. When the partition heals, all replicas will eventually receive all updates. As depicted in Fig. 1, instead of trying to find a linear order of updates and to solve conflicts with rollbacks, the concurrent DAG states are merged by attaching both causally independent chains of events to the last synchronized event, i.e., by forking the DAG. Accepting concurrency in the data type itself by providing only a partial order on events is the core idea of the MEG. It is also the basis for our proof of conflict-freedom in Section VI. A fork in the DAG introduced by concurrency will lead to two causally independent forward extremities. Following the attachment rules for new vertices, a replica that has received and appended both causally independent chains to its DAG selects both as parents for a new vertex. In terms of graphs, this means that the new vertex will join both chains again, which indicates the end of the period of concurrency and causal independence, and reduces the number of forward extremities by one.

**Problem statement.** The way in which concurrency is handled in a MEG, as well as the use of various parameters as outlined above, give rise to the key research questions addressed in this paper:

- Which consistency guarantees can application developers expect from a MEG?
- Under which assumptions do these guarantees hold?
- Can the width of the MEG degenerate?

The preceding explanations describe how the MEG maintains availability under partition, and how it tries to achieve Eventual Consistency, as conjectured by the Matrix developers [10]. In this paper, we provide a proof of Strong Eventual Consistency in Section VI. In Section VII, we relax the adopted assumptions, particularly on the communication primitive. In addition, the overview above showed that if the number of vertex parents is restricted to $d$ and selected randomly, the evolution of the number of forward extremities $u$, i.e., the width of the DAG, is non-trivial in concurrent environments. In Section VIII, we explore whether the width of the DAG converges within a sufficiently small number of iterations for arbitrary start values of the initial number of forward extremities $u$, if $k$ replicas continuously select $d$ parents independently and then synchronize the new vertices. In particular, we investigate how the choice of the number of parent vertices $d$ and $k$ affects the speed of convergence.

**Not in the scope of this paper:** While we make assumptions on and deal with the underlying broadcast primitive, we consider the topic of broadcast communication per se beyond the scope of this paper. Moreover, Matrix employs an access control concept, which we assume to be present in MEGs. The access control aspects were examined in [11] and are not the object of this analysis.

## IV. RELATED WORK & BACKGROUND

Jacob *et al.* investigated quantitative aspects of the public Matrix federation and found scalability problems with the broadcast communication currently employed by Matrix [12]. However, they did not investigate the scalability and other properties of the replicated data structure itself. The access control system of Matrix, which builds on top of the MEG, was recently studied in [11]. Privacy and usability aspects of Matrix, along with a CRDT-based vision on how to improve this situation in federated networks in general, are the topic of [13].

In the field of replicated data types, Shapiro *et al.* introduced the category of Conflict-Free Replicated Data Types (CRDTs), together with a new consistency model provided by this category, namely Strong Eventual Consistency [4]. Following the initial definition, new papers mostly focused on implementations of the data type, like the JSON-CRDT by Kleppmann *et al.* [14], or extended the base concept of CRDTs [15]. The initial CRDT concept was overhauled in cooperation with the original authors in [16]. We will mainly use the new CRDT terminology introduced there.

In contrast to traditional distributed databases that aim for strong consistency or to consistency models for distributed ledger technologies that aim for consensus, SEC does provide neither a global total order nor finality. However, SEC improves over Eventual Consistency, common in Internet-scale distributed databases, as it does not require conflict arbitration or rollbacks [4], [17].

### A. CONSISTENCY MODELS

The inherent trade-off between *Consistency* and *Availability* in the presence of network partitions in distributed systems led to the definition of a variety of consistency models. A well-known consistency model is *Eventual Consistency*, which provides the following guarantees [4]:

- *Eventual Delivery:* An update applied by some correct replica is eventually applied by every correct replica.
- *Termination:* Every invoked method terminates.
- *Convergence:* Correct replicas that applied the same set of updates eventually reach equivalent states.

*Strong Eventual Consistency (SEC)* builds on Eventual Consistency, and strengthens Convergence [4]:

- *Strong Convergence:* Correct replicas that applied the same set of updates have equivalent states.

Whether two states are equivalent depends on the specific application. In our case, the state of two replicas is equivalent if their graphs consist of identical vertices and edges. Note that "the same *set* of updates" means that, while the updates are identical, they might be received or applied in different

order. The key difference between Convergence and Strong Convergence is that with Convergence, replicas may coordinate with other replicas to reach agreement on their state even after having applied updates. Especially, if the ordering of updates matters, this can lead to rollbacks. With Strong Convergence, the agreement has to be immanent and implicit.

### B. CONFLICT-FREE REPLICATED DATA TYPES

Conflict-Free Replicated Data Types (CRDTs) were first formalized in [4]. CRDTs are an abstract data structure that allows for optimistic update execution (cf. [18]), while guaranteeing conflict-freedom upon network synchronization. The system model of CRDTs is based on a *fail-silent* abstraction with a Causal Order Reliable Broadcast communication protocol (see Section V). For objects that implement a CRDT in a system with $n$ replicas, Shapiro *et al.* show that SEC is ensured for up to $n-1$ replica failures [4].

Two conceptually different, but equally expressive types of CRDTs are the *operation-based* and the *state-based* CRDT. Replicas implement functions that can be invoked by clients to access or modify the state. The key difference between operation-based and state-based CRDTs lies in the way of synchronization: In state-based CRDTs, all replicas periodically send their full state to all other replicas, which then merge states. In contrast, operation-based CRDTs only synchronize upon changes. Source replicas transmit state changes resulting from a client invocation as *operations*. In this paper, we focus on operation-based CRDTs and show in Section VI that the MEG is an operation-based CRDT.

Operation-based CRDTs implement functions that can be classified as either `update` or `query` function. A `query` function returns information on the current state of the replica, whereas an `update` function modifies the state. An `update` function consists of two parts: At first, a `generator`[3] part is executed by the source replica. It is side-effect-free and returns an *operation*, i. e., an encapsulation of the state changes. The second part is called `effector`[4], it must be executed at every replica. The source replica transmits the generated operation to all replicas using broadcast. Upon reception of an operation, each replica executes the `effector` part locally and applies the resulting changes to their state [19].

In general, the data structure of a CRDT cannot maintain a specific shape or topology, such as a DAG, as concurrent updates could violate shape or topology invariants. However, specific implementations of CRDTs can overcome this restriction, for example as shown by the *Operation-based Add-only monotonic DAG* described in [20]. Their implementation allows clients to collaboratively edit a DAG, by adding vertices and edges in separate updates. Topology preservation is enforced by rejection of new edges that violate the current partial order of the DAG. In a similar vein, the MEG is designed in a way that preserves its topology as rooted DAG inherently, which we will show in Subsection VI-B.

[3]Originally introduced as *prepare-update*
[4]Originally introduced as *effect-update*

## V. ASSUMPTIONS AND ARCHITECTURE

**Assumptions.** We study a single replica group, consisting of a static and known set of replicas, for a single publish-subscribe topic. Furthermore, we make use of two failure models, both based on the *asynchronous* timing assumption, which means that no upper bounds on computation or network transmission times are given.

The *fail-silent* model [21, p. 63] implies that faulty replicas can crash-stop at any time, while the remaining replicas have no means to reliably distinguish failure from communication or processing delays, i.e., the fault is 'silent'. The *fail-silent-arbitrary* model [21, p. 64] allows for arbitrary, i.e. byzantine, behavior of faulty replicas. This feature includes intentionally malicious behavior and protocol non-adherence. In this model, 'silent' also means that replicas cannot detect whether another replica currently adheres to the protocol or not. We call a replica *correct* if it is non-faulty. Correct replicas strive to achieve consistency on the full MEG state, and interact with each other to reach consistency and provide failure tolerance.

The formal CRDT-proof that we give in Section VI is based on the stricter assumption of a *fail-silent* model. In Section VII, we extend the claims to the *fail-silent-arbitrary* model.

We assume that replicas are connected by direct links with an arbitrarily varying but finite delay, i.e., faulty replicas cannot prevent any two correct replicas from eventually communicating. Furthermore, we make use of two broadcast abstractions in this work. Firstly, we use *Reliable Broadcast*. Informally, this abstraction provides a set of properties which guarantee that, eventually, the same set of messages is received by all correct replicas, even if the sending replica fails [21]. Formally, Reliable Broadcast provides the following properties:

- *Validity:* If a correct replica sends a message $m$, then it eventually receives $m$.
- *No duplication:* Correct processes do not receive messages more than once.
- *No creation:* If a correct replica receives a message $m$ with sender $p$, then $m$ was previously sent by $p$ if $p$ is correct.
- *Agreement:* If a message $m$ is received by some correct replica, $m$ is eventually received by every correct replica.

The other, more powerful, abstraction is called *Causal Order Reliable Broadcast*. It extends the guarantees of Reliable Broadcast by additionally preserving the *causal order* of messages [21]:

- *Causal Delivery:* For any messages $m_1$ and $m_2$ where the broadcast of message $m_1$ *happened before* (cf. [7]), the broadcast of message $m_2$, $m_2$ is only received by replicas that have already received $m_1$.

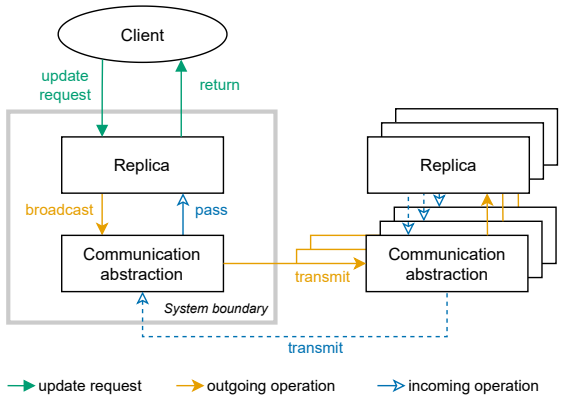


**FIGURE 2.** An update request by a client invokes the generator of an update function at the replica, which creates an update operation. This update operation is then transmitted to all replicas, including the calling replica itself, through the communication abstraction. The communication abstraction enforces guarantees about incoming operations, e.g. on their ordering. The system boundary indicates the components of a single participant.

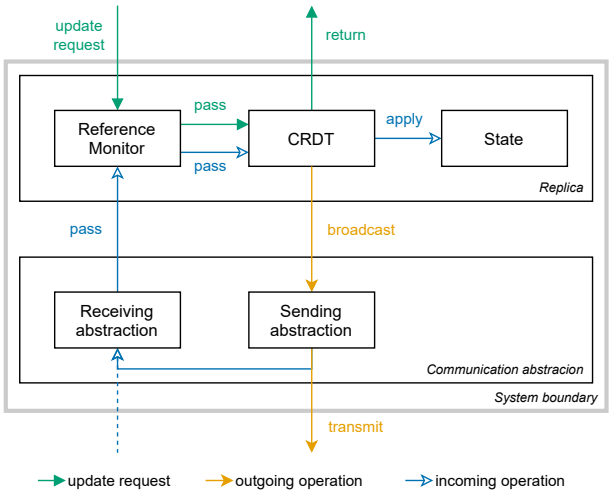


**FIGURE 3.** Inner workings of the source replica and communication abstraction when receiving an update request or an incoming operation. After entering the replica through the Reference Monitor, a valid update request or a valid incoming operation is passed to the CRDT. In case of an update request, the CRDT encodes the state changes as an operation which is then broadcast to all replicas using the communication abstraction. An incoming update operation is processed at the CRDT component, which then applies the operation to the local state of the replica.

The formal CRDT-proof in Section VI is based on the *Causal Order Reliable Broadcast* abstraction. In Section VII, we relax the required primitive to *Reliable Broadcast* — even in byzantine scenarios — while maintaining the CRDT properties.

**Architecture.** As we can see in Fig. 2, each *client* is attached to a single trusted *replica*. The client can request functions of class `query` or `update` at their replica, as defined in Subsection IV-B. As part of executing an `update` function, the source replica distributes operations, i.e., encoded state changes, to all replicas using a broadcast *communication abstraction*.

A more granular architectural view is provided by Fig. 3. Inside a replica, the *Reference Monitor* is the entry point for incoming requests from clients and operations from remote replicas. It serves as a gate keeper to prevent further processing of operations or requests that violate the protocol or, in a byzantine setting, originate from unauthorized or unauthenticated parties. Operations and requests that pass the Reference Monitor are handed to the *CRDT*. The CRDT can read and modify the state of the replica and is thus the core logic module of the replica. In case of a `query` request, it accesses the state and returns the desired value. For `update` requests, the generator of the update function encapsulates state changes into an operation that is passed to the communication abstraction. The CRDT then returns to the client to indicate success. The communication abstraction sends the update operation to all replicas, including the calling replica itself.[5] These update operations then trigger the local update effector, which applies the changes to the state of the replica.

## VI. THE MEG AS CRDT

Building upon the overview given in Section III, we formalize the MEG as an operation-based replicated object. We then show Theorem 1 in accordance with the assumptions used by Shapiro *et al.* for CRDTs (cf. Subsection IV-B) [19].

**Theorem 1.** *Under the assumption of a fail-silent $n > f$ environment with $n$ total and $f$ faulty participants, and a Causal Order Reliable Broadcast primitive, the MEG is an operation-based CRDT and thereby provides Strong Eventual Consistency (SEC).*

### A. FORMALIZATION OF THE MEG

To define the MEG as a CRDT, we adopt the formal definition introduced with the concept of operation-based CRDTs in [4], [19] and use the pseudo code notation by Preguiça [22].

An object is formally defined as a tuple $(S, s^0, q, t, u, P)$: the space of possible per-replica states is denoted by $S$, and $s^0 \in S$ is the `initial` state of every replica. The set of `query` functions is given by $q$. `update` functions are composed of a `generator` step $t$ and an `effector` step $u$. The `effector` $u$ may contain a *delivery precondition* $P$, which must be fulfilled before an operation is processed further. Notably, the delivery precondition $P$ only delays the execution, but it does not abort the `effector` step. When a replica with `state` $s \in S$ executes a step $u$, we denote this as $s \bullet u$, which yields a new `state`. As shorthand for the `state` at replica $i$, we write $s_i \in S$.

We provide a pseudo code implementation of the MEG as an operation-based CRDT in Listing 1. A vertex is a tuple $(e, w)$ that represents an event in the MEG, where $w$ is a unique identifier for the event and $e$ contains the actual event. Edges represent a potential causal relationship between child and parent vertex. The `state` is a DAG, defined through

[5] While, depending on the specific communication abstraction, this is not required in an actual implementation, it is important on a conceptual level to ensure that the guarantees hold.

**Listing 1.** Pseudo code implementation of the Matrix CRDT. The type of the respective functions is indicated by `query` and `update`. The two parts of the `update` function are denoted by `generator` and `effector`. The delivery precondition $P$ is denoted by `pre`. The function `unique` returns a unique identifier.

```
state set S = (V, E) // vertices V are of the form
    (event e, uid w), E represents the edges:
    E ⊆ V × V
  initial ({(e_0, w_0)}, ∅)
  query lookup (uid w) : boolean
    return ∃((e', w') ∈ V) : w' == w
  query hasChild (vertex (e, w)) : boolean
    return ∃((e', w') ∈ V) : ((e', w'), (e, w)) ∈ E
  query getExtremities () : list of vertices
    return L = ⋃_{(e,w)∈V: not hasChild((e,w))} {(e, w)}
  query getState () : set
    return S
  update add
    generator (event e)
      let L = getExtremities()
      let w = unique()
      return add, (e, L, w)
    effector (event e, list of vertices L, uid w)
      pre: ∀(e_p, w_p) ∈ L: lookup(w_p)
      V = V ∪ {(e, w)}
      E = E ∪ ⋃_{(e_p,w_p)∈L} {((e, w), (e_p, w_p))}
```

a set of vertices and a set of edges. In the inital state $s^0$, it consists of a single vertex and no edges. The `query` functions `lookup`, `hasChild`, `getExtremities` and `getState` allow to access the replica state without modification. The `lookup` function checks whether a vertex with a given identifier is part of the current state. Similarly, the `hasChild` function checks for the existence of child vertices for a given vertex. The `getExtremities` function returns the current set of forward extremities, whereas `getState` returns the `state`. The `update` function `add` is used to append new events to the MEG. Its `generator` part $t$ takes the event $e$ as an input argument. Based on the `state` of the source replica at that time, a set $L$ of forward extremities is created. Lastly, a unique identifier $w$ is chosen. The parameters $w$, $e$ and $L$, and a reference to the update function `add` are returned together.

The `effector` $u$ is invoked by the operation that was created in the `generator` step. Once the delivery precondition $P$ is fulfilled, the new vertex $(e, w)$ and the new edges $((e, w), (e_p, w_p))$ for each $(e_p, w_p) \in L$ are added to the `state`, i.e., to the set of vertices and edges, respectively. The delivery precondition $P$ states that all 'parents' in the set $L$ are required to be part of the current state before the new vertex can be added.

### B. PRESERVATION OF THE DAG TOPOLOGY

As mentioned in Subsection IV-B, the preservation of a specific shape, such as a DAG, is not possible in a generic way for CRDTs. We now show that the MEG always preserves the desired data structure of a rooted DAG by design as Lemma 2.

**Lemma 1.** *There is at least one forward extremity at any time after initialization of the MEG.*

*Proof.* Can easily be seen by induction. □

**Lemma 2.** *The MEG maintains the properties of a rooted DAG at all times: (i) single root, (ii) acyclicity, and (iii) weak connectedness.*

*Proof.* By induction.
*Base case:* The initial state $s^0$ contains a single vertex and no edges. This MEG therefore is a rooted DAG.
*Induction step:* Given replicas $i$ with state $s_i = (V_i, E_i)$, where $s_i$ is a rooted DAG, an arbitrary source replica $r$ is selected. As part of the `generator` $t$, the set of forward extremities is determined as $L$, and a unique identifier $w$ is created. By Lemma 1, the set of forward extremities $|L|$ is non-empty. Since generator $t$ is side-effect-free, the MEG remains unchanged.

Consequently, the execution of the `effector` $u$ is triggered at each replica $i$. Effector $u$ awaits the fulfillment of the delivery precondition $P$, which ensures that $s_i$ contains all parents that are referenced by $L$. Finally, applying $u$ yields the new replica states $s_i'$:

$$s_i' = (V_i \cup \{(e, w)\}, E_i \bigcup_{(e_p,w_p)\in L} \{(e, w), (e_p, w_p))\}).$$

Since all new edges are outgoing from the new vertex $(e, w)$, no new cycles can be formed, and existing roots remain roots. No new roots or isolated vertices have been added, as the new vertex has outgoing edges. Because all $s_i$ were assumed to be rooted DAGs, all $s_i'$ must be rooted DAGs. □

### C. PROOF OF CRDT PROPERTIES

We now show Theorem 1, i.e., that MEGs implement an operation-based CRDT and, thus, guarantee SEC. We make use of the Theorem on Commutative Replicated Data Types [4], which states that *"assuming causal delivery of updates and method termination, any op[eration]-based object that satisfies the commutativity property for all concurrent updates, and whose delivery precondition is satisfied by causal delivery, is SEC."*

We need to show commutativity of concurrent updates and causal order reception of operations for noncommutative updates. Commutativity for updates is determined by the commutativity of their operations. Two updates $(t, u)$ and $(t', u')$ commute, if and only if for any reachable state $s \in S$ in which the delivery precondition $P$ is satisfied for both $u$ and $u'$, the following properties hold: (i) $P$ is still satisfied for $u$ in state $s \bullet u'$, and (ii) $s \bullet u \bullet u' \equiv s \bullet u' \bullet u$.

We proceed in the following way: we present three lemmata on the preservation of the delivery precondition, on the commutativity of operations, and on the eventual fulfillment of the delivery precondition. With these lemmata, Theorem 1 is then easily proved.

**Lemma 3.** *Once an update operation satisfies the delivery precondition $P$ for some state $s$, it will continue to satisfy $P$ for any state $s'$ following $s$.*

*Proof.* Consider any update operation $u(e, L, w)$, i.e., the operation $(e, L, w)$ applied via the effector $u$, that satisfies

$P$ in some state $s = (V, E)$. Applying an arbitrary operation $u(e', L', w')$ to $s$ yields a new state $s'$:

$$\begin{aligned} s' &= s \bullet u(e', L', w') \\ &= (V \cup \{(e', w')\}, E \cup \bigcup_{(e_p, w_p) \in L'} \{(e', w'), (e_p, w_p)\}) \end{aligned}$$

A delivery precondition $P$ being satisfied in $s$ implies that it remains satisfied for $s'$:

$$\begin{aligned} &\forall (e_p, w_p) \in L : (e_p, w_p) \in V \\ \Rightarrow &\forall (e_p, w_p) \in L : (e_p, w_p) \in V \cup \{(e', w')\} \end{aligned}$$

□

**Lemma 4.** *Any two operations $u(e_i, L_i, w_i)$ and $u(e_j, L_j, w_j)$ commute with each other.*

*Proof.* We consider any state $s = (V, E)$ and two update operations $u(e_i, L_i, w_i)$, $u(e_j, L_j, w_j)$ that both satisfy delivery precondition $P$ in $s$.

As shown in Lemma 3, after applying one operation, the other operation still satisfies $P$. It remains to show that the resulting states are equivalent, regardless of the order in which the effectors are executed. Since $u$ only performs a union of the edge and vertex sets, by commutativity of the union operator, commutativity of $u$ follows:

$$\begin{aligned} &s \bullet u(e_i, L_i, w_i) \bullet u(e_j, L_j, w_j) \\ \equiv\ &s \bullet u(e_j, L_j, w_j) \bullet u(e_i, L_i, w_i) \end{aligned}$$

□

**Lemma 5.** *Under the assumption of Causal Delivery broadcast, the CRDT delivery precondition $P$ is immediately satisfied on message reception.*

*Proof.* The delivery precondition $P$ ensures that all referenced parents are part of the local `state`. Since `getExtremities` selects all parents from the current `state`, $P$ must be satisfied at the source replica after the `generator` step. Once satisfied, $P$ remains satisfied since vertices are never removed. Therefore, receiving all causally preceding operations is sufficient to satisfy $P$ at every replica. Consequently, having causal order message reception, $P$ is immediately satisfied upon reception. □

*Proof. (Theorem 1)* As we have shown, MEG updates are commutative (Lemma 4), and all essential properties of the MEG are preserved by design (cf. Lemma 2). Since the MEG encodes causal relations as edges in the data structure, the delivery precondition $P$ can ensure that these dependencies are respected without sacrificing commutativity. Thus, Strong Convergence is guaranteed. Eventual Delivery is guaranteed by Lemma 5. Given the implementation in Listing 1, we can see that there are no loops or recursive calls in either of the functions, therefore, they will eventually exit. Knowing that the delivery precondition $P$ is immediately satisfied given causal order message reception, as shown in Lemma 5, we can conclude that Termination holds. Therefore, all properties of an operation-based CRDT are met by the MEG, and Theorem 1 holds. □

## VII. RELAXATION OF ASSUMPTIONS AND REALITY CHECK FOR BYZANTINE SETTINGS

In this section, we evaluate the assumptions of Theorem 1 that were used in the CRDT proof of the MEG in Subsection VI-C, and relax them wherever possible without violating previously shown guarantees. We remove the Causal Order guarantee of the broadcast primitive in Theorem 2 of Subsection VII-A. In Subsection VII-B, we switch from a fail-silent $n > f$ environment with $n$ total and $f$ faulty participants to a fail-silent-arbitrary, i.e. byzantine, $n > f$ environment, which results in Theorem 3. In Subsection VII-C, we compare Matrix as an implementation of the MEG concept in the byzantine setting to the requirements of Theorem 3, and conclude that it currently does not provide SEC without additional assumptions because of its unreliable broadcast protocol.

### A. RELAXATION OF THE BROADCAST ASSUMPTIONS

In Section VI, we assumed a Causal Order Reliable Broadcast abstraction, which is commonly used with CRDTs. However, the causal relationships of events depicted in the MEG hints that the Causal Order delivery property is not necessary in addition to Reliable Broadcast[6] for the MEG to function, and can be removed without violating Strong Convergence for safety as well as Eventual Delivery and Termination for liveness (cf. Section IV for definition). We formulate this observation as Theorem 2:

**Theorem 2.** *Under the assumption of a fail-silent $n > f$ environment with $n$ total and $f$ faulty participants, and a Reliable Broadcast primitive, the MEG is an operation-based CRDT and thereby provides Strong Eventual Consistency.*

*Proof.* For the proof, we revisit every property required for the CRDT proof as in Subsection VI-C, and remove applications of the Causal Order property of the broadcast primitive.

**Strong Convergence.** To provide Strong Convergence, replicas must receive noncommutative update operations in their causal order. Since every update operation commutes with every other, as shown in Lemma 4, Strong Convergence does not require any ordering guarantees by the communication abstraction.

**Eventual Delivery.** In Lemma 5, we used the Causal Delivery property to show that the delivery precondition $P$ is immediately satisfied on reception. For Eventual Delivery, we now need to show that the delivery precondition $P$ is still *eventually* satisfied without Causal Delivery.

[6]The No Duplication property of Reliable Broadcast is also not necessary: because each vertex has a unique identifier $w$, and because outgoing edges cannot be added afterwards, it suffices to make the effector conditional on the presence of the vertex in the replica state to gain idempotent effectors that can cope with multiple receptions of identical operations.

Given an update operation, $P$ is satisfied if all referenced parents are part of the `state` of a replica. If an operation satisfies $P$ at some point in time, it continues to satisfy $P$ thereafter, because the MEG is an append-only data structure. As per Lemma 5, $P$ is satisfied for any given operation after the `generator` step at the source replica finishes. Therefore, all referenced parents must have been previously added to the `state` and therefore be part of some update operation. If an update operation does not satisfy $P$ at some replica due to reordering of operations by the broadcast abstraction, replicas can delay and buffer the update operation until $P$ is satisfied. Owing to the Validity and Agreement properties of the broadcast abstraction (cf. Section V), all missing update operations are eventually received by all correct replicas. As correct replicas apply all operations that they received and that eventually satisfy $P$, all parents must eventually be part of their `state`. Consequently, for correct replicas, $P$ must eventually be satisfied for every update operation.

**Termination.** Since all method executions terminate, and since we have shown that in the new setting, $P$ is eventually satisfied for all operations, the Termination property still holds.

Thus, the MEG only requires Reliable Broadcast, and does not depend on Causal Delivery. □

### B. TOLERATING BYZANTINE FAILURES

We now switch from a *fail-silent* $n > f$ environment with $n$ total and $f$ faulty participants to a *fail-silent-arbitrary*, i.e. byzantine, $n > f$ environment. A fail-silent-arbitrary $n > f$ environment means that a client's trusted replica might be the only non-malicious replica in the system.

It is intuitively reasonable that the MEG is able to cope with such a hostile environment, as it does not strive for consensus: a secure implementation of the Reference Monitor component can identify and filter invalid update operations before they are applied by the effector. Neither the CRDT properties, nor the Reference Monitor, nor the implementation of Validity and Agreement in the broadcast protocol require a quorum or majority vote. Therefore, a majority in the system results in no advantage for the attacker. Based on this intuition, we formulate and prove an analogy of Theorem 2 for a fail-silent-arbitrary $n > f$ environment. However, we first have to clarify assumptions on the attacker and the system.

The main difference in the byzantine environment to previous settings is that attackers can arbitrarily influence those parts of the decentralized system that they control. Considering the system architecture shown in Fig. 3, the attackers control the `generator` part and the sending of the broadcast of their replicas. Through the generator and the broadcast, they can try to influence the receiving part, namely the Reference Monitor and the `effector` of correct replicas, which process attacker-defined input.

From the system architecture, we derive that the attacker is limited to two basic attack capabilities: First, they can violate the broadcast protocol by performing equivocation, i.e., broadcasting different update operations to different replica subsets, or not broadcasting an update operation to all replicas [23]. Second, they can generate and send malicious update operations. This attacker model is based on the threat model [24] for CRDTs by Zhao *et al.*, but simplified as we assume a static, known set of replicas.

Note that "byzantine reliable broadcast", as it is commonly understood [21, p. 120], requires that all correct replicas deliver *the same single message*, even in face of equivocation. This requirement leads to a quorum or other form of majority vote, with correct replicas in two-thirds majority over faulty replicas, i.e. $f < \frac{n}{3}$. However, the MEG is *equivocation-tolerant*, which means that on sender equivocation, correct MEG replicas can cope with receiving *two messages in arbitrary order* from the broadcast abstraction, as long as every correct replica receives both. Due to the MEG's conflict-free nature and commutative operations, this is sufficient to provide SEC, as we will show later in Theorem 3.

**Lemma 6.** *Under the assumption of transferable authentication, there exists a broadcast algorithm that provides Validity and Agreement for equivocation-tolerant algorithms in a fail-silent-arbitrary $n > f$ environment with $n$ total and $f$ faulty participants.*

*Proof.* We show that the classical Eager Reliable Broadcast algorithm [21, p. 80] provides Validity and Agreement, as defined in Section V, for equivocation-tolerant algorithms under the employed assumptions. The algorithm works as follows: A correct sender best-effort-broadcasts the message to all replicas, including itself. Best-effort broadcast works by sending the message to all receivers via reliable unicast, which ensures Agreement as long as the sender is correct. A correct replica immediately delivers the message on reception, which ensures Validity. To ensure Agreement if the sender fails during best-effort broadcast, receiving replicas best-effort-broadcast the message to every replica again, if they have not yet broadcast that message.

We assume that transferable authentication [25] is available, e.g., in the form of digital signatures. Due to transferable authentication, faulty replicas cannot forge messages by other replicas, or manipulate relayed messages. A faulty replica is therefore left with attacking Agreement of the underlying best-effort broadcast via other forms of equivocation [25], either as a sender or a receiver:

If *receiving replicas are faulty* and perform no or a faulty re-broadcast, a correct sender will still contact all correct replicas directly, which ensures Agreement.

A *byzantine sender* can broadcast a message to only a subset of replicas. If the subset contains a correct replica, the replica will deliver the message. Then, it will re-broadcast the message to all replicas, which includes all other correct ones, and will thereby ensure Agreement. If the subset contains no correct replica, the message is not delivered by any correct replica, and Agreement is therefore not violated.

A byzantine sender can also broadcast two different messages to different subsets of replicas. Correct replicas will

immediately deliver any of the received messages. Due to the re-broadcasts, both messages will eventually be delivered by all correct replicas, ensuring Agreement. □

To further describe the attacker capabilities, we have to introduce a distinction between *valid* and *invalid* operations. An operation is *valid* if it cannot be distinguished by the receiving replica from operations that originate from a protocol-abiding `generator`. Valid operations, therefore, include operations from correct replicas as well as from faulty replicas that abide by or deviate from the protocol in indistinguishable ways. An operation is *invalid* if it does not possess authenticity, integrity, or otherwise violates assumptions of the protocol that can be distinguished from valid operations by the receiving replica. While this notion of invalid events can directly be extended to unauthorized events, we consider that out of scope for this paper, and reference publication [11] for a discussion on how a MEG Reference Monitor can ensure authorization based on policy information embedded in the MEG state itself.

As long as invalid operations are applied in the same way by all correct replicas, a CRDT in a state derived from invalid operations would technically still achieve Strong Eventual Consistency, as that would neither violate Eventual Delivery nor Strong Convergence. However, to make the SEC guarantee more meaningful for applications in the fail-silent-arbitrary setting, we define an additional security property that prevents invalid operations from being applied:

- *Protocol Observance*: A correct replica does not apply an invalid operation to its state.

**Lemma 7.** *Under the assumption of a fail-silent-arbitrary $n > f$ environment with $n$ total and $f$ faulty participants, transferable authentication, and a broadcast primitive that provides Validity and Agreement for equivocation-tolerant algorithms, the Reference Monitor component of a MEG can provide Protocol Observance.*

*Proof.* The main task of the Reference Monitor component is to shield the CRDT from invalid operations, i.e., to provide Protocol Observance. To achieve this task, the Reference Monitor is the endpoint for all external interfaces of the replica.

*Authenticity* and *integrity* prevent impersonation and unobtrusive alteration of operations that are not directly received from the source replica. Both properties can be achieved by the assumed transferable authentication.

Faulty replicas can violate the forward extremity selection mechanism and include *non-concurrent forward extremities* as parents. If the parents are received by any correct replica, they will eventually be received by all due to Validity and Agreement, and the Reference Monitor can verify that none of the parents is an ancestor of any others before the effector applies the operation. However, an operation with forged, e.g., random event identifiers for which no operation exists, is indistinguishable from a valid operation for which the parents were not received yet. Therefore, such an operation is valid by definition, and is not part of Protocol Observance. This is a potential denial-of-service attack, and it will be discussed for Theorem 3.

Faulty replicas can generate operations with *non-unique event identifiers*, in order to violate the uniqueness assumption used in the MEG. If applied to the state, such operations would have disastrous consequences for Strong Convergence, as they tamper with the very definition of what *the same* events are. However, (probabilistically) unique identifiers can be ensured in a byzantine environment, by generating identifiers from the event data and metadata using a collision-resistant hash function. In this way, Reference Monitors can verify whether an identifier is valid by recomputing the hash themselves, which is necessary to ensure Strong Convergence in an equivocation-tolerant way.

We conclude that operations that pass the Reference Monitor are valid. Thereby, Protocol Observance is provided. □

We continue to assume a static, known set of replicas in the following proof. Still, we want to note that dynamic replica sets introduce additional attack surfaces in byzantine environments: such attacks may prevent replicas from receiving some or all update operations, which could affect Eventual Delivery. We consider this an important, but separate topic.

For the following discussion of SEC in byzantine environments, we need an additional way to classify operations: An operation generated by a faulty replica is *malicious* if the operation would not have been sent if a correct replica was in the faulty replica's place. While all invalid operations are malicious, valid operations can be malicious as well: For example, faulty replicas can send operations that were not generated by using all forward extremities available to the faulty replica. However, for correct replicas, such an operation is indistinguishable from an operation generated by a correct replica that just has not applied those forward extremities yet.

**Theorem 3.** *Under the assumption of a fail-silent-arbitrary $n > f$ environment with $n$ total and $f$ faulty participants, transferable authentication, and a broadcast primitive that provides Validity and Agreement for equivocation-tolerant algorithms, the MEG is an operation-based CRDT and thereby provides Strong Eventual Consistency.*

*Proof.* We revisit every property required for the CRDT proof as in Subsection VI-C, and we check how byzantine replicas can try to break those properties for correct replicas.

**Strong Convergence.** As we can assert Protocol Observance as given by Lemma 7, Strong Convergence is only concerned with the state resulting from the application of valid operations by correct replicas.

As given by Lemma 6, the broadcast abstraction ensures Agreement in byzantine $n > f$ environments. Nevertheless, by using equivocation, byzantine replicas can influence the order in which their operations are applied. However, this fact does not impair Strong Convergence, as shown in Subsection VII-A.

Byzantine replicas can send malicious but valid operations that are applied to the state. Still, under unique event identifiers, which we can assert due to Lemma 7, any two valid operations commute as of Subsection VII-A, and Strong Convergence holds. The application of a malicious but valid operation can only lead to different outcomes on different replicas if one or multiple replicas deviate from the protocol, e.g. due to an implementation error. In this case, the deviating replicas are faulty and, thus, do not require further consideration for Strong Convergence.

**Eventual Delivery.** Lemma 6 shows that we can assert Validity and Agreement by the broadcast abstraction even in a byzantine $n > f$ environment. By Lemma 7, we can assert that only valid operations reach the effector of a correct replica. We will now show that for a given valid operation, the delivery precondition $P$ is either eventually fulfilled on all correct replicas, or never fulfilled on any correct replica. Be reminded that the MEG's delivery precondition states that all 'parents' referenced by an operation have to be part of the replica state before the operation is applied.

A byzantine replica has two ways to target Eventual Delivery: It can equivocate and provide only a subset of replicas with the update operation, or it can create malicious but valid updates that will never fulfill the delivery precondition.

For any form of *equivocation*, the broadcast abstraction will still ensure Agreement, and Protocol Observance will ensure unique identifiers for non-identical operations. Thereby, every correct replica will receive all valid update operations received by any correct replica. Therefore, the attacker cannot hinder eventual satisfaction of the delivery precondition through equivocation.

An attacker can target Eventual Delivery by the creation of *valid but malicious operations* that include forged identifiers as parents for which no forward extremities exist. In an asynchronous system, those operations are indistinguishable from operations for which the parents have not yet been received, but will be eventually. Therefore, operations that refer to non-existing parents stay forever in the delivery precondition fulfillment buffer. Consequently, the malicious operations will never be delivered at any correct replica. If a single correct replica could apply the operation eventually, it must have received its parents. Then, due to Agreement, all other correct replicas would eventually receive and therefore apply all parent operations as well.

As byzantine replicas cannot lead to an inconsistent fulfillment of the delivery precondition on different correct replicas, Eventual Delivery holds.

**Termination.** As we can assert Protocol Observance as given by Lemma 7, the Termination property in the byzantine setting is about the effector applying malicious but valid operations. Those operations can take two main forms relevant for Termination:

*Not including all forward extremities* does not prevent operation application from terminating. We still want to note that this behavior can slow down the convergence of the MEG width (cf. Section VIII for discussion of MEG width behavior).

As noted with Eventual Delivery, operations that refer to *non-existing parents* stay in the delivery precondition fulfillment buffer forever, as the parents will never be received. To stop an attacker from filling the receiver's buffer, a heuristic approach to detection could be applied by monitoring the buffer sizes, and enforcing a depletion if it grows to unreasonable levels for a specific replica. In case of a correct sender for which the missing parent operations will still be eventually received after depletion, the dropped operations can be re-requested by following the DAG relations as soon as newer operations arrive.

In conclusion, valid but malicious operations can incur load on performance and slow the MEG width shrinkage, but do not threaten Termination for valid, non-malicious operations.

**Conclusion.** While byzantine attackers can try to perform denial-of-service attacks to incur load on performance, neither malicious updates nor equivocation can threaten Strong Convergence, Eventual Delivery, or the Termination property under the employed assumptions. Therefore, Theorem 3 holds. □

### C. REALITY CHECK

In the Matrix system, replicas currently employ best-effort broadcast [26], [27], instead of a reliable broadcast protocol like it was described in Lemma 6. As replicas immediately apply update operations to their local state, best-effort broadcast provides Validity. However, best-effort broadcast does not provide Agreement even in fail-silent systems without byzantine attackers, as a faulty replica could only provide a limited number of correct replicas with the update operation. To mitigate this issue, Matrix uses a backfilling mechanism which allows replicas to specifically request missing operations from other replicas. It is used when a replica receives an update operation for which the parents are not part of the replica state. With this mechanism, Matrix achieves Agreement under the assumption of constant MEG progress, i.e., a never-ending stream of (arbitrary low-frequency) new update operations from each correct replica. However, if the progress comes to a halt, Agreement is violated. Therefore, Matrix only satisfies the requirements of Theorem 3 and thereby provides SEC under the assumption of constant progress.

While Matrix could use the reliable broadcast protocol described in Lemma 6 to provide SEC without constant progress, the scalability of the Matrix best-effort broadcast protocol is already an issue if it is used by the sender alone [12]. For increased scalability, [12] suggests to replace the Matrix broadcast implementation with a gossip-based broadcast protocol that is scalable and robust. However, gossiping introduces a likelihood that correct receivers do not receive a message from a correct sender, and therefore still requires constant progress to ensure Agreement. To increase reliability, replicas could periodically broadcast their current set of forward extremities to all other replicas, which

then could trigger backfilling. This addition would guarantee probabilistic Agreement, and therefore (probabilistic) SEC for the MEG implementation of Matrix.

## VIII. SCALABILITY: WIDTH OF THE MEG OVER TIME

An assessment of the scalability of the MEG replicated data type needs to consider various aspects. While the previous two sections focused on the safety and liveness properties of Strong Eventual Consistency, topics concerning scalability came up repeatedly, which will be collected now. Due to the conflict-free nature of the MEG that allows the application of updates without further coordination, the scalability in terms of the number of replicas is first of all determined by the employed Reliable Broadcast protocol and its implementation of the Validity and Agreement properties. Probabilistic agreement, in particular, offers the opportunity for performance tuning. Second, as an append-only data structure, a strategy is required for compression or garbage collection to avoid a monotonically growing height of the MEG. This second aspect is addressed in the Matrix implementation, but not considered in this paper. Third, the width of the MEG, i.e., the number of forward extremities, needs to be analyzed. The evolution of the width of the MEG over time is the focus of this section. We follow an analytical approach to deliver a precise mathematical definition and treatment of the problem.

In contrast to Sections VI and VII, where we assumed that *all* forward extremities known to a replica are used as parents for new vertices created by the replica, in the following the number of parents of a new vertex is restricted to a finite value $d$. If there are more than $d$ forward extremities, a replica randomly (uniformly distributed) selects a subset of size $d$ of forward extremities as parents for the new vertex. The reason for a fixed value $d$ is based on considerations of performance: correct replicas can experience a high number of forward extremities after a partition, and malicious replicas could deliberately create events with a high number of parents. This is problematic from a performance perspective because checks, particularly of the Reference Monitor, are resource intensive graph algorithms, but needed for every parent [9]. Please note that the proofs of the previous sections are not affected by this relaxation of assumptions.

In this section, we provide evidence that the width of the MEG still converges[7] to the the number $k$ of participating replicas times a small factor when all $k$ replica repeatedly and concurrently add a new vertex.

We model the evolution of the width of the MEG by assuming that vertices are added in rounds, and a round consists of two steps:

- Step 1: each of the $k$ replicas concurrently adds a new vertex. It thereby 'eliminates' the $d$ forward extremities that were chosen as parents, while the new vertex becomes a forward extremity itself.
- Step 2: all replicas synchronize their new extremities and reach a consistent state.

The overall number of eliminated extremities depends on the *overlap* between the chosen parents of different replicas. As we are interested in scaling $k$ while keeping $d$ low, we assume $k > d$. As the number of forward extremities never decreases if new forward extremities have only one parent, we assume $d > 1$. The model also accepts an arbitrarily high number of forward extremities $u_0$ as a starting condition. We analyze the sequence of numbers $u_i$ of forward extremities after round $i$ by a mean value analysis.

Please note that this model maximizes uncoordinated concurrency in Step 1 and, thus, models a worst case scenario: More new vertices per replica, i.e., a higher frequency of updates by clients or prolonged periods of network partition, would eliminate more than $d$ non-overlapping forward extremities, but not create additional ones. Also, if replicas would be aware of the eliminations of other replicas, their forward extremity choices could be done more overlap-free.

### A. STOCHASTIC PROCESS

In order to determine the width of the MEG over time, we are interested in the number of parents selected in Step 1 of the above evolution model. The concurrent updates in Step 1 of each round can be nicely modeled by a stochastic urn problem. The initial number of forward extremities $u_i$ in round $i$ is described by $u_i$ initial red balls, while the number of newly linked parent vertices $d$ is the number of balls taken out during a drawing by a single replica. The update generator executions of the $k$ replicas lead to the conduction of $k$ independent drawings that can be modeled by sequential drawings with the use of black balls: the balls drawn by a replica are replaced by black balls and put back to the urn. Therefore, after $k$ replicas have performed Step 1, the black balls indicate the number of selected parent vertices. After each round, the black balls are replaced by red ones again and the next round starts with the current number $u_{i+1}$ of red balls.

We let the random variable $R_{d,k}(u)$ denote the total number of removed forward extremities, while $u - R_{d,k}(u)$ denotes the number of forward extremities that remain for the subsequent urn experiment. With this urn experiment, we model a stochastic process for the behavior of the number of forward extremities. We derive the expectation and the variance of $R_{d,k}(u)$, and we provide a recursion formula for the distribution of $R_{d,k}(u)$. We discuss the implications on MEGs in Subsection VIII-B.

Let the random variable $U_n$ describe the number of balls in the urn after $n \in \mathbb{N}_0$ rounds. Let $u_0$ be the initial number of balls in the urn, then $U_0 = u_0$ and

$$U_{n+1} = U_n + k - R_{d,k}(U_n).$$

As $(U_n)_{n\in\mathbb{N}_0}$ is a sequence of random variables, it is a stochastic process (cf., e.g., [28]). We are interested in whether convergence can be *expected*, and, if yes, how fast

[7]Convergence in this section refers to the number of forward extremities. In the previous CRDT-related section, convergence is related to propagation of states.

convergence is reached. The process is a spatially inhomogeneous random walk, specifically a time-homogeneous Markov chain (cf., e.g., [29]) with state space $M_U = \mathbb{N}^+$:

$$\begin{aligned}&\forall n \in \mathbb{N}_0 \;\; \forall u_0, \ldots, u_{n+1} \in M_U : \\ &\quad \mathbb{P}(U_{n+1} = u_{n+1} | U_0 = u_0, \ldots, U_n = u_n) \\ &\quad = \mathbb{P}(U_{n+1} | U_n = u_n)\end{aligned}$$

Thus, the process is memoryless with transition matrix $P_{i,j} = \mathbb{P}(U_n = j | U_{n-1} = i) = \mathbb{P}(R_{d,k}(i) = k - (j - i))$ and transition probability $\forall n \in \mathbb{N}_0 \forall i, j \in M_U : \mathbb{P}(U_n = j | U_{n-1} = i) = \mathbb{P}(U_1 = j | U_0 = i)$. Thus, the process is time-homogeneous.

We now provide the expectation (a) and the variance (b) of $R_{d,k}(u)$, and a recursion formula (c) for the distribution of $R_{d,k}(u)$. For the proof, see Appendix .

**Theorem 4.** *For the random variable $R_{d,k}(u)$, we have:*

*a)* $\mathbb{E}(R_{d,k}(u)) = d \cdot \dfrac{1-p^k}{1-p}, \quad k \geq 1,$

*where*

$$p = \frac{u-d}{u} \tag{1}$$

*is the retention probability.*

*b)*

$$\begin{aligned}&\mathbb{V}(R_{d,k}(u)) = \\ &\frac{vud}{1-p}\left(\frac{1-w^{k-1}}{1-w} - p^{k-1} \cdot \frac{1-(w/p)^{k-1}}{1-w/p}\right) \\ &-\frac{vd^2}{(1-p)^2}\left(\frac{1-w^{k-1}}{1-w} - 2p^{k-1}\frac{1-(w/p)^{k-1}}{1-w/p}\right. \\ &\left.+p^{2(k-1)}\frac{1-(w/p^2)^{k-1}}{1-w/p^2}\right),\end{aligned}$$

*where*

$$v = \frac{d(u-d)}{u^2(u-1)}, \quad w = \frac{(u-d)(u-d-1)}{u(u-1)}. \tag{2}$$

*c) If $k \geq 2$ then*

$$\mathbb{P}(R_{d,k}(u) = j) = \sum_{\ell=0}^{d} \frac{\binom{u-(j-\ell)}{\ell}\binom{j-\ell}{d-\ell}}{\binom{u}{d}} \cdot \mathbb{P}(R_{d,k-1}(u) = j - \ell).$$

### B. IMPLICATIONS FOR THE MEG AND REALITY CHECK

In this subsection we make use of the results for the stochastic process model to gain insights into convergence and convergence speed of the width of the MEG, and we check how the real-world Matrix implementation compares to our findings.

The formula for the expectation of $R_{d,k}(u)$ allows for statements on the expected convergence behavior of the MEG in the presence of concurrent updates by different replicas. In addition, the formula for the variance of $R_{d,k}(u)$ shows the deviation from expected convergent behavior. We use these formulas to calculate the expected development and deviation of forward extremities $U_n$ over the number of rounds for varying $k$ but fixed $d$. To plot the calculations in Figure 4, we put different realizations of $U_n$ against the expected value of $U_{n+1}$, via $\mathbb{E}(U_{n+1}) = U_n + k - \mathbb{E}(R_{d,k}(U_n))$. The dashed line is $U_{n+1} = U_n$, its intersection with the colored lines marks their fixed points. In the area below the dashed line, $\mathbb{E}(U_{n+1}) < U_n$, the urn contents are expected to decrease, in accordance with the plotted standard deviation. The change from linear to constant curves (for decreasing $U_n$, i.e. from right to left) shows the shift from likely overlap-free choices to overlapping choices, which decrease the urn contents to a lesser extent. It shows that for any plotted realization of $U_n$, we either expect a decreasing urn value (below the dashed line), or a transition to the fixed point. Therefore, the plotted configurations show convergence. In addition, the variance is very low. We observe that the convergence of the width of the graph appears to be almost optimal, i.e., the fixed point is near the number of replicas $k$.

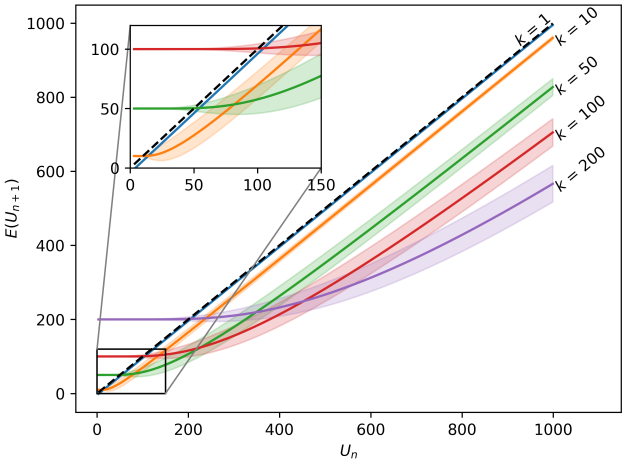


**FIGURE 4.** Expectation for the next urn content $\mathbb{E}(U_{n+1})$ for different realizations of $U_n$, $d = 5$, and varying $k$. Points below the dashed line of $U_n = \mathbb{E}(U_{n+1})$ mean that the urn content is expected to decrease, points above mean that an increase is expected. For visibility, the plotted standard deviation is increased by the factor 5. Please note that when the curves are followed from right to left, they change from a linear slope to a constant value close to $k$.

Synapse, the reference implementation of a Matrix replica, recently released a feature to force the depletion of forward extremities by sending empty 'dummy' events using the same parent selection rules as regular events[8] with $d = 10$, as soon as there are more than 10 forward extremities present [30]. This fact allows to take advantage of the convergence in periods of missing update operations, and it brings reality closer to our model.

To gain insights into the influence of the number $d$ of parents of a new vertex, we use the expectation of $U_n$ via $\mathbb{E}(U_{n+1}) = \mathbb{E}(U_n) + k - \mathbb{E}(R_{d,k}(U_n))$, and calculate the number of rounds $n$ until $\mathbb{E}(U_n) - \mathbb{E}(U_{n+1}) < 1$. Fig. 5 shows that the number of rounds until convergence is reached directly depends on the choice of $d$. However, there are di-

[8]Note that Synapse actually takes 5 random forward extremities and 5 of the newest forward extremities, which are not independent between replicas.

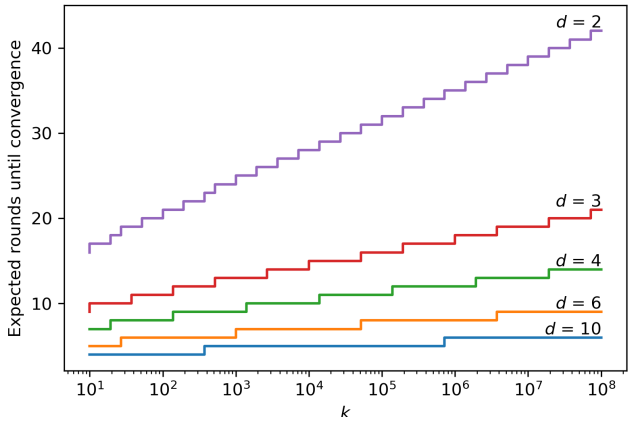


**FIGURE 5.** Expected number of rounds until convergence for varying $d$ and $k$, starting at $u_0 = 100 \cdot k$. While convergence speed increases with $d$, the returns in the number of rounds to reach convergence diminish.

minishing returns: the highest gain in time until convergence is between $d = 2$ and $d = 3$, while there is much less difference between $d = 6$ and $d = 10$. When $u \gg k \cdot d$, the speed of convergence is nearly $k \cdot (1 - d)$, and therefore the number of rounds until convergence is nearly proportional to $\frac{1}{1-d}$. Synapse employs $d = 5$ with $k \lessapprox 10^3$, which we can confirm as a good compromise in convergence speed and performance using our formulas in Figs. 4 and 5.

With small $u$, bad choices, i.e. overlapping choices for parents, are made, but because $u$ is small, they do not permanently impair convergence. With large $u$, the probability for overlapping choices becomes smaller and smaller, and convergence speed is linear.

### C. CONJECTURE

We conjecture that the above stochastic process represents a positive recurrent Markov chain and, since the process is aperiodic and irreducible, it has a stationary distribution, i.e., a fixed point of the transition function in which the probabilities for the next state do not change with state transitions.

The properties of being aperiodic and irreducible can be easily seen as follows. If we assume $u_0 \in [0, k-1]$, then $u_1 \geq k$, as no more than $u_0$ balls can be drawn, but $k$ balls are added. Therefore, states $[0, k-1]$ are transient and can be removed from the chain. The remaining states are irreducible and aperiodic: As the next state increment in one round is in $[k - k \cdot d, k - d]$, every other state can be reached in a finite number of iterations. However, it is unclear whether the remaining states are transient, i.e., visited only once, or positively recurrent, i.e., have a finite expected time until they are visited again. This question represents an open problem and is left for future work.

In practice, this means that if the conjecture holds, the MEG possesses a self-stabilization property [31] in the sense that if transient faults lead to a high number of forward extremities (a high $u$), a correct system converges to a stable state with a number of forward extremities above but near $k$ in a finite number of rounds, and remains stable as if the faults had never occurred.

## IX. LIMITATIONS AND OPEN ISSUES

The theorems of the previous sections as well as their interpretation in the context of the Matrix implementation show that the MEG represents a distinct and interesting replicated data type of practical relevance. Therefore, the open issues we identified in the previous sections represent challenges for further research, and they are summarized in this section.

First of all, for an MEG to represent a CRDT, a reliable broadcast primitive is required. When an MEG is used for a topic in a publish-subscribe system, there is no need for a 'one-size-fits-all' solution. Instead, an appropriate broadcast primitive could be selected on a per-topic basis. When probabilistic broadcast is used, tuning the parameters to meet the demands of the application is required. Since there are a plethora of existing reliable broadcast and multicast approaches, the selection process and/or auto-tuning appears the more important problem than devising new approaches.

Second, as an append-only structure, the MEG needs strategies for compression and/or garbage collection to ensure scalability. Somewhat related is the issue of resistance to denial-of-service attacks: how should one handle the situation when a replica tries to add an excessive number of new vertices? We want to emphasize again that such an 'attack' would not compromise the fundamental properties of the replicated data structure. However, it could keep a replica busy or buffers full. Related to the garbage collection strategy, we note that in Matrix, replicas are allowed to garbage-collect old, irrelevant events to conserve space in accordance with individual policies. New replicas are not required to sync the full MEG state, from the dawn of a topic's creation, but can store multiple discontiguous components of the full MEG state instead. Using the backfilling mechanism described in Subsection VII-C, replicas can retrieve events on demand, as long as any other live replica still has them in its state. In addition, replicas could only store specific types of publish-subscribe messages that their users are subscribed to, or even employ a way of sharding so that just a subset of the participating replicas is responsible to maintain specific graph partitions. Therefore, the extension of the proofs to non-uniform replicas with different synchronization policies is another relevant open issue.

Third, we presented evidence that a high number of forward extremities will quickly collapse to only a few forward extremities. However, while we were able to show this behavior for expected values, the proof of positive recurrence of the underlying Markov chain represents an open problem.

In terms of limitations, we showed the MEG properties in form of manual proofs. However, especially in the world of CRDTs, machine-checked proofs and formal verification are well-established techniques [32]. While we currently look at formal verification methods in the context of MEG-based access control, we deem formal verification of the properties

required for SEC as well as specific correctness properties of the MEG an important step to strengthen our results.

MEG replicas detect their own errors and the errors of remote replicas based on the local state of the causal structure, but the correction of local or remote errors requires communication with other correct replicas. While we assume a static replica group and direct eventual communication between all correct replicas, in practice, churn is present and an indirect communication is preferable to arbitrarily long waiting times for direct communication. Even with churn and indirect communication, we conjecture that, as long as correct replicas form a connected component in the network, performance can suffer, but fault tolerance and consistency properties of the MEG are still valid.

## X. CONCLUSION

We extracted and abstracted the replicated data type employed by the Matrix specification, and we proved that this Matrix Event Graph (MEG) represents a Conflict-Free Replicated Data Type. Therefore, the MEG provides Strong Eventual Consistency, a fact that in particular implies that all correct replicas that applied the same set of updates are in an equivalent state – immediately and without additional coordination. This proof explains why the Matrix system, in practice, shows good resilience and scalability in the number of replicas. Hence, it makes the MEG an attractive candidate as a basis for other decentralized applications.

In addition, we analyzed the challenges for systems with byzantine actors and demonstrated that the properties of the MEG facilitate a byzantine-tolerant design, especially due to equivocation-tolerance. We showed that in practice, those properties are not guaranteed by the Matrix reference implementation yet and provided ideas for possible solutions. Design and analysis of an underlying broadcast protocol with the identified properties remain topics for future research.

Furthermore, we formalized and studied the evolution of the width of the graph as a spatially inhomogeneous random walk. Our observations let us conjecture that the width of the graph always converges independently of the specific system parameters, and does so fast. If the conjecture can be confirmed, the convergence translates to a self-stabilizing property that allows a real-world system to cope with transient faults in a scalable way.

In summary, we believe that the MEG represents a highly relevant data structure for real-world scenarios that require decentralization, scalability, and Strong Eventual Consistency. When compared to traditional blockchains and distributed ledger technologies, the MEG strengthens political independence of participants, as it does not require global coordination. However, the commutativity requirement for updates is the price to pay for its equivocation-tolerance. We hope that our results advance the understanding as well as the appropriate real-world deployment of those systems, and can serve as a basis for further research into the Matrix Event Graph replicated data type.

## APPENDIX. PROOF OF PROPERTIES OF THE TOTAL NUMBER OF REMOVED FORWARD EXTREMITIES $R$

For a series of drawings $R_{d,k}(u)$, we write $Z_k$ for the number of red balls that show up in the $k$th drawing, so that $R_{d,k}(u) = Z_1 + \ldots + Z_k$.

a) In what follows, let $k \geq 2$. Under the condition $R_{d,k-1}(u) = r$, the urn contains $u - r$ red and $r$ black balls. Thus, the conditional distribution of $Z_k$ given $R_{k-1} = r$ is the hypergeometric distribution $\mathrm{Hyp}(d, u-r, r)$, which implies

$$\mathbb{E}(Z_k | R_{d,k-1}(u) = r) = d \cdot \frac{u-r}{u}.$$

Since $R_{d,k}(u) = R_{d,k-1}(u) + Z_k$, we have $\mathbb{E}(R_{d,k}(u)) = \mathbb{E}(R_{d,k-1}(u)) + \mathbb{E}(Z_k)$. Moreover,

$$\begin{aligned}\mathbb{E}(Z_k) = \mathbb{E}\big[\mathbb{E}(Z_k | R_{d,k-1}(u))\big] &= \mathbb{E}\left[d \cdot \frac{u - Z_{k-1}}{u}\right] \\ &= d - \frac{d}{u} \cdot \mathbb{E}(R_{d,k-1}).\end{aligned}$$

It follows that

$$\begin{aligned}\mathbb{E}(R_{d,k}(u)) &= \mathbb{E}(R_{d,k-1}(u)) + d - \frac{d}{u} \cdot \mathbb{E}(R_{d,k-1}(u)) \\ &= d + p\,\mathbb{E}(R_{d,k-1}(u)).\end{aligned}$$

Together with $\mathbb{E}(R_{d,1}(u)) = d$, we now obtain by induction over $k$

$$\mathbb{E}(R_{d,k}(u)) = d \sum_{j=0}^{k-1} p^j = d \cdot \frac{1-p^k}{1-p},$$

as was to be shown. Notice that

$$\lim_{k\to\infty} \mathbb{E}(R_{d,k}(u)) = \frac{d}{1-p} = u.$$

This result is not surprising, since in the long run each of the red balls will have shown up.

b) The proof uses the general fact that, for random variables $X$ and $Y$, the variance of $X$ can be calculated according to the formula $\mathbb{V}(X) = \mathbb{E}[\mathbb{V}(X|Y)] + \mathbb{V}(\mathbb{E}[X|Y])$, i.e., the variance of $X$ is the sum of the expectation of the conditional variance of $X$ given $Y$ and the variance of the conditional expectation of $X$ given $Y$. In our case, we put $X = R_{d,k}(u)$ and $Y = Z_{k-1}$, where $k \geq 2$, and obtain

$$\begin{aligned}\mathbb{V}(R_{d,k}(u)) &= \mathbb{V}(R_{d,k-1}(u) + Z_k) \\ &= \mathbb{E}\big[\mathbb{V}(R_{d,k-1}(u) + Z_k | R_{d,k-1}(u))\big] \\ &\quad + \mathbb{V}\left(\mathbb{E}[R_{d,k-1}(u) + Z_k | R_{d,k-1}(u)]\right).\end{aligned} \tag{3}$$

Since $\mathbb{V}(R_{d,k-1}(u) + Z_k | R_{d,k-1}(u)) = \mathbb{V}(Z_k | R_{d,k-1}(u))$ and the conditional distribution of $Z_k$ given $R_{d,k-1}(u)$ is the

hypergeometric distribution $\mathrm{Hyp}(d, u-R_{d,k-1}(u), R_{d,k-1}(u))$, it follows that

$$\begin{aligned}\mathbb{V}(Z_k|R_{d,k-1}(u)) = & \\ & d\cdot\frac{u-R_{d,k-1}(u)}{u}\cdot\left(1-\frac{u-R_{d,k-1}(u)}{u}\right)\\ & \cdot\left(1-\frac{d-1}{u-1}\right)\\ = & \frac{d}{u^2}\left(1-\frac{d-1}{u-1}\right)\\ & \cdot(u-R_{d,k-1}(u))R_{d,k-1}(u).\end{aligned} \tag{4}$$

Moreover, we have

$$\begin{aligned}\mathbb{E}[R_{d,k-1}(u)+Z_k|R_{d,k-1}(u)] = & \\ R_{d,k-1}(u)+\mathbb{E}[Z_k|R_{d,k-1}(u)] & \\ = Z_{k-1}+d\cdot\frac{u-R_{d,k-1}(u)}{u} & \\ = d+\left(1-\frac{d}{u}\right)R_{d,k-1}(u). &\end{aligned}$$

Therefore, the second summand figuring in (3) equals

$$\left(1-\frac{d}{u}\right)^2\mathbb{V}(R_{d,k-1}(u)).$$

Since $\mathbb{V}(R_{d,k-1}(u)) = \mathbb{E}\left[R^2_{d,k-1}(u)\right] - (\mathbb{E}R_{d,k-1}(u))^2$, Equation (4) yields

$$\begin{aligned}\mathbb{E}[\mathbb{V}(Z_k|R_{d,k-1}(u))] = & \\ \frac{d}{u^2}\left(1-\frac{d-1}{u-1}\right)\cdot(u\mathbb{E}(R_{d,k-1}(u)) & \\ -\mathbb{V}(R_{d,k-1}(u))-(\mathbb{E}R_{d,k-1}(u))^2\big). &\end{aligned}$$

We thus obtain the recursion formula

$$\begin{aligned}\mathbb{V}(R_{d,k}(u)) = & v\cdot\mathbb{V}(R_{d,k-1}(u))\\ & +\frac{d}{u^2}\cdot\left(1-\frac{d-1}{u-1}\right)\\ & \cdot\left(u\mathbb{E}(R_{d,k-1}(u))-(\mathbb{E}R_{d,k-1}(u))^2\right)\end{aligned}$$

with $v$ given in (2), from which the result follows by straightforward calculations. Notice that $\mathbb{V}(R_{d,1}(u)) = 0$, ($R_{d,1}(u)$ is the constant $d$), and that $\lim_{k\to\infty}\mathbb{V}(R_{d,k}(u)) = 0$. The latter convergence is clear from the fact that, in the long run, all red balls will have been drawn.

c) The result follows from the fact that the event $\{R_{d,k} = j\}$ is the union of the pairwise disjoint events $\{R_{d,k-1}(u) = j-\ell, Z_k = \ell\}$, $\ell = 0, 1, \ldots, d$, and the fact that the conditional distribution of $R_{d,k}(u)(= R_{d,k-1}(u) + Z_k)$ given $R_{d,k-1}(u) = j-\ell$ is the hypergeometric distribution $\mathrm{Hyp}(d, u-(j-\ell), j-\ell)$.

## ACKNOWLEDGMENT

We thank the Matrix developers for their ingenious system design. Hannes Hartenstein acknowledges funding of the Helmholtz Association (HGF) through the Competence Center for Applied Security Technology (KASTEL). Florian Jacob thanks Alexander Marsteller for many hours of differential equation analysis and mathematical brainstorming. We thank our peer reviewers for their valuable comments and improvements, and all who gave us feedback on preliminary drafts of this paper.

## References

[1] K. Zhang and H. Jacobsen, "Towards dependable, scalable, and pervasive distributed ledgers with blockchains," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 1337–1346.

[2] M. Raikwar, D. Gligoroski, and G. Velinov, "Trends in development of databases and blockchain," *arXiv preprint arXiv:2003.05687*, 2020.

[3] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *SIGACT News*, vol. 33, no. 2, pp. 51–59, Jun. 2002, ISSN: 0163-5700. DOI: 10.1145/564585.564601.

[4] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," in *Symposium on Self-Stabilizing Systems*, Springer, 2011, pp. 386–400.

[5] (2020). "Matrix.org: Frequently asked questions," The Matrix.org Foundation C.I.C., [Online]. Available: https://matrix.org/docs/guides/faq.html (visited on 01/27/2021).

[6] M. Hodgson, "The path to peer-to-peer matrix," FOSDEM 2020. [Online]. Available: https://archive.fosdem.org/2020/schedule/event/dip_p2p_matrix/.

[7] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978, ISSN: 15577317. DOI: 10.1145/359545.359563.

[8] J. C. Platt, N. Cristianini, and J. Shawe-Taylor, "Large margin DAGs for multiclass classification," in *Advances in Neural Information Processing Systems 12*, MIT Press, 2000, pp. 547–553.

[9] M. Hodgson. (2017). "Forward extremities accumulate and lead to poor performance," [Online]. Available: https://github.com/matrix-org/synapse/issues/1760 (visited on 05/31/2020).

[10] (2019). "Matrix specification: Architecture," The Matrix.org Foundation C.I.C., [Online]. Available: https://matrix.org/docs/spec/#architecture.

[11] F. Jacob, L. Becker, J. Grashöfer, and H. Hartenstein, "Matrix decomposition: Analysis of an access control approach on transaction-based DAGs without finality," in *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies*, ser. SACMAT, 2020, pp. 81–92. DOI: 10.1145/3381991.3395399.

[12] F. Jacob, J. Grashöfer, and H. Hartenstein, "A glimpse of the Matrix: Scalability issues of a new message-oriented data synchronization middleware," in *Proc. ACM 20th Int. Middleware Conference Demos and Posters*, 2019, pp. 5–6.

[13] A. Auvolat, "Making federated networks more distributed," in *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, 2019, pp. 383–384.

[14] M. Kleppmann and A. R. Beresford, "A conflict-free replicated JSON datatype," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 10, pp. 2733–2746, 2017.

[15] K. De Porre, C. Scholliers, F. Myter, W. De Meuter, C. De Troyer, and E. G. Boix, "A generic replicated data type for strong eventual consistency," *Proceedings of the 6th Workshop on Principles and Practice of Consistency for Distributed Data, PaPoC 2019*, 2019. DOI: 10.1145/3301419.3323974.

[16] N. Preguiça, C. Baquero, and M. Shapiro, "Conflict-free replicated data types CRDTs," *Encyclopedia of Big Data Technologies*, pp. 491–500, 2019. DOI: 10.1007/978-3-319-77525-8_185.

[17] W. Vogels, "Eventually consistent," *Commun. ACM*, vol. 52, no. 1, pp. 40–44, Jan. 2009, ISSN: 0001-0782. DOI: 10.1145/1435417.1435432.

[18] Y. Saito and M. Shapiro, "Optimistic replication," *ACM Computing Surveys*, vol. 37, no. 1, pp. 42–81, 2005, ISSN: 03600300. DOI: 10.1145/1057977.1057980.

[19] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," Tech. Rep., 2011. [Online]. Available: https://hal.inria.fr/inria-00609399v1.

[20] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "A comprehensive study of convergent and commutative replicated data types," Tech. Rep. RR-7506, 2011. [Online]. Available: https://hal.inria.fr/inria-00555588.

[21] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to Reliable and Secure Distributed Programming*, 2nd. Springer Publishing Company, Incorporated, 2011, ISBN: 9783642152597. DOI: 10.1007/978-3-642-15260-3.

[22] N. Preguiça, "Conflict-free replicated data types: An overview," *arXiv preprint arXiv:1806.10254*, 2018. [Online]. Available: http://arxiv.org/abs/1806.10254.

[23] M. F. Madsen and S. Debois, "On the subject of non-equivocation: Defining non-equivocation in synchronous agreement systems," in *Proceedings of the 39th Symposium on Principles of Distributed Computing*, ser. PODC '20, Virtual Event, Italy: Association for Computing Machinery, 2020, pp. 159–168, ISBN: 9781450375825. DOI: 10.1145/3382734.3405731.

[24] W. Zhao, M. Babi, Y. William, X. Luo, Y. Zhu, Y. Jack, L. Chaomin, and M. Yang, "Byzantine fault tolerance for collaborative editing with commutative operations," *IEEE International Conference on Electro Information Technology*, pp. 246–251, 2016, ISSN: 21540373. DOI: 10.1109/EIT.2016.7535248.

[25] A. Clement, F. Junqueira, A. Kate, and R. Rodrigues, "On the (limited) power of non-equivocation," in *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, ser. PODC '12, Madeira, Portugal: Association for Computing Machinery, 2012, pp. 301–308, ISBN: 9781450314503. DOI: 10.1145/2332432.2332490.

[26] O. Wilkinson. (2020). "Catch-up after federation outage," [Online]. Available: https://github.com/matrix-org/synapse/pull/8272 (visited on 10/30/2020).

[27] R. van der Hoff. (2017). "Homeservers don't catch up with missed traffic until someone sends another event," [Online]. Available: https://github.com/matrix-org/synapse/issues/2528 (visited on 04/02/2020).

[28] R. G. Gallager, *Discrete stochastic processes*. Springer Science & Business Media, 2012, vol. 321.

[29] M. Mitzenmacher and E. Upfal, *Probability and computing: randomization and probabilistic techniques in algorithms and data analysis*. Cambridge University Press, 2017.

[30] E. Johnston. (Jun. 19, 2019). "Synapse: Add experimental option to reduce extremities," [Online]. Available: https://github.com/matrix-org/synapse/pull/5480 (visited on 06/01/2020).

[31] B. Awerbuch, B. Patt-Shamir, G. Varghese, and S. Dolev, "Self-stabilization by local checking and global reset," in *Distributed Algorithms*, G. Tel and P. Vitányi, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 326–339, ISBN: 978-3-540-48799-9.

[32] V. B. F. Gomes, M. Kleppmann, D. P. Mulligan, and A. R. Beresford, "Verifying strong eventual consistency in distributed systems," *Proc. ACM Program. Lang.*, vol. 1, no. OOPSLA, Oct. 2017. DOI: 10.1145/3133933.

FLORIAN JACOB graduated in informatics at the Karlsruhe Institute of Technology in 2019, and has since worked as a staff member in the Decentralized Systems and Network Services research group. His research interests are in the investigation of decentralized, federated communication networks like Matrix, focussing on scalability and security. Within this context, he also deals with end-to-end encryption, usable security, distributed data structures, and access control mechanisms.

CAROLIN BEER received a B. Sc. degree in industrial engineering and management from Karlsruhe Institute of Technology (KIT), Germany, in 2018 and is currently pursuing a M. Sc. in computer science at the same university.

She worked as Student Research Assistant in the *Decentralized Systems and Network Services* and the *Smart Grids and Energy Markets* research groups at KIT. In addition, she gained industry experience as Intern with Google, IBM, and Amazon. Her research interests are in the area of distributed and decentralized systems, particularly concerning fault-tolerance, consistency models, and scalability. Ms. Beer holds a scholarship of the Friedrich Ebert Foundation for her bachelor's and master's studies.

NORBERT HENZE received the doctoral degree in mathematics and the Dr.Sc. degree from the University of Hannover. He held visiting positions at the University of Göttingen and the University of Gießen. Since 1991 he has been a professor at the Karlsruhe Institute of Technology. His main research interests are in goodness-of-fit tests, nearest neighbor methods, and geometric extreme value theory.

HANNES HARTENSTEIN (M'97) received a diploma in mathematics and a doctoral degree in computer science from Albert-Ludwigs-Universität, Freiburg, Germany. He was a Senior Research Staff Member with NEC Europe. He was appointed to the chair of "Decentralized Systems and Network Services" (DSN) in October 2003 and has directed the DSN research group at the Karlsruhe Institute of Technology since then. His research interests include decentralized and distributed systems, information security, and information technology management. He is a Principal Investigator in the Competence Center for Applied Security Technology KASTEL.