On Extend-Only Directed Posets and Derived Byzantine-Tolerant Replicated Data Types

Florian Jacob
florian.jacob@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

Hannes Hartenstein
hannes.hartenstein@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

Abstract

We uncover the extend-only directed posets (EDP) structure of recently discussed Byzantine-tolerant conflict-free replicated data types (CRDT). We also show how a key-value map model can be derived from the EDP formulation, provide sketches for proving strong eventual consistency, and give an outlook on a systemic access control CRDT.

CCS Concepts: • **Security and privacy → Distributed systems security**; *Access control*; • **Software and its engineering → Consistency**.

Keywords: Conflict-Free Replicated Data Types, Strong Eventual Consistency, Byzantine Fault Model, Matrix Event Graph

1 Introduction

Recently, we noticed [6] that the conflict-free replicated data type (CRDT) of a system for decentralized messaging (Matrix, [11]) retains its CRDT property also in environments with Byzantine nodes, i.e., in environments with nodes that arbitrarily deviate from the expected protocol behavior. Kleppmann [7] was able to show that an arbitrary crash-fault tolerant CRDT can be transformed into a CRDT that tolerates an arbitrary number of Byzantine nodes. These proposals for the Byzantine case have in common that events are managed in a graph-like shared object: events are independently generated at the various replicas (available under partition), but associated to previous events known to the replica.

The shared objects essentially show the following dynamic behavior. All replicas start with an initial state of the same genesis event. To append a new event, replicas attach this event to all events currently known to them that "happened-before" and have no "descendants", i.e., the "most recent" events, without coordinating with other replicas. Replicas then synchronize, i.e., exchange updates, to reach a consistent state again. When replicas append events that happened concurrently, branches occur, which leads to a tree-like structure. When replicas learn of branches, they will join them again on their next event, thereby eliminating branches. Events consist of both payload as well as hashes of previous events, which ensures integrity and strong eventual consistency in the face of Byzantine equivocation [7, 4]. An example evolution over time of such an object is illustrated in Fig. 1.

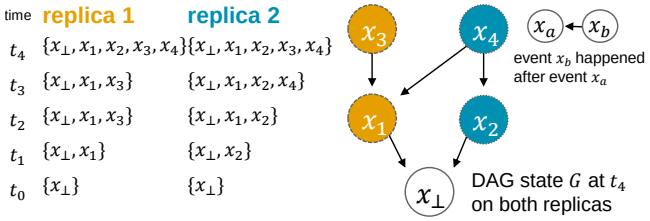

Figure 1. Example graph state G at a point in time t_4 as seen by both replicas and containing events from both replicas and the pre-shared genesis event $x_\perp$. First, both replicas send events x_1 and x_2 in concurrently. When replica 2 sends event x_4, it has learned about x_1 from replica 1 and therefore puts both x_1 and x_2 as happened-after parents. Replica 1 has not learned of x_2 yet, and thereby appends x_3 only to x_1.

While in previous work, the corresponding objects have been described as directed acyclic graphs (DAGs), in this work-in-progress paper we propose a more fundamental formalization via *directed partially-ordered sets* (directed posets), called Extend-only Directed Posets CRDT (EDP CRDT), that shows the following advantages:

- The set-theoretic formalization captures the essence of Byzantine-tolerant CRDTs, revealing *a)* a state-based CRDT that does not require crypto elements, and *b)* a hash-based operation-based construction as optimization for efficiency. We believe that the EDP CRDT facilitates formal verification in future work. Furthermore, the EDP CRDT formulation makes use of standard mathematical notions and fits the theory of CRDTs that is largely formulated in terms of set theory [10].
- The EDP CRDT results in a class of CRDTs that can be used to build derived types as well as combinations that can then more easily be shown to have the CRDT property in Byzantine environment. In this paper, we give an example for a map-based data type and an outlook towards a CRDT for systemic access control, inspired by the Matrix system.

Related work made use of sets instead of DAGs in Byzantine environments, however, did not treat and generalize them as CRDTs [3]. In the following sections, we will define EDP CRDTs and indicate why they represent a solid basis for the class of Byzantine-tolerant CRDTs discussed above.

2 Extend-only Directed Posets (EDPs)

2.1 Basics and Notations

The state space of the EDP replicated data type is based on relational structures, for which we will now introduce mathematical basics and notation.

A relational structure $S = (X, R)$ is a tuple which consists of a ground set X and a relation $R \subseteq X \times X$. To facilitate notation, we sometimes write $S.X$ and $S.R$ to denote the set X and the relation R of $S = (X, R)$. A partially-ordered set, also called a poset, is a relational structure that is reflexive ($\forall a \in X\colon (a, a) \in R$), transitive ($\forall a, b, c \in X\colon (c, b) \in R \wedge (b, a) \in R \Rightarrow (c, a) \in R$), and antisymmetric ($\forall a, b \in X\colon (a, b) \in R \wedge (b, a) \in R) \Rightarrow a = b$). While a $\leq$-like R is usual in mathematics, we use a $\geq$-like R in these definitions for consistency with the rest of the paper. If S is a poset, then R is called a partial order relation, or partial ordering. If S is also strongly connected ($\forall a, b \in X\colon (a, b) \in R \vee (b, a) \in R$), S is called a linearly-ordered set.

A reflexive and transitive relational structure S is called a downward-directed set if for any two elements, the set contains a lower bound, i.e., $\forall a, b \in X\colon \exists x_{lb} \in X$ s.t. $(a, x_{lb}) \in R \wedge (b, x_{lb}) \in R$. A finite, downward-directed poset is directed towards its unique least element $x_\perp$, also denoted as $S^\perp$, i.e., $x_\perp \in X$ and $\forall x \in X\colon (x, x_\perp) \in R$. Conversely, a finite upward-directed poset is directed towards its top element $S^\top = x_\top \in X$, i.e., $\forall x \in X\colon (x_\top, x) \in R$. A both downward- and upward-directed finite poset is said to be a poset bounded by $x_\perp$ and $x_\top$. The set of maximal elements of S is $\max(S) = \{m \in X | \forall x \in X\colon (x, m) \in R \Rightarrow (m, x) \in R\}$. Conversely, the set of minimal elements of S is $\min(S) = \{m \in X | \forall x \in X\colon (m, x) \in R \Rightarrow (x, m) \in R\}$.

A relational structure $S' = (X', R')$ is called an extension of another relational structure S if $X \subseteq X'$ and $R = R'|_X$, where the restriction $R|_A$ is defined as usual as $R|_A = R \cap (A \times A)$. We call S' an upward extension of a downward-directed poset S if additionally $S'^\perp = S^\perp$. The downward closure $y^{\downarrow S}$ of an element $y \in S.X$ is defined as $y^{\downarrow S} = \{c \in X | (y, c) \in R\}$. The downward closure $Y^{\downarrow S}$ of a subset $Y \subseteq X$ is generalized from the single-element downward closure as $Y^{\downarrow S} = \bigcup_{y \in Y} y^{\downarrow S}$. The upward closure of elements and subsets of X is defined correspondingly, $y^{\uparrow S} = \{c \in X | (c, y) \in R\}$. The set of maximal lower bounds of an element $y \in X$ is the set of maximal elements of the downward closure of y without y itself, $\text{mlb}(y) = \max(y^{\downarrow S} \setminus \{y\})$.

2.2 Specification

To build an append-only CRDT (as sketched in the introduction) that tolerates an arbitrary number of Byzantine nodes, the key idea is to replace an event x with its downward closure, i.e., adding all the relations to previous events. Thereby, a byzantine node cannot create inconsistent relations anymore — it can only create spam additions. The EDP RDT will be defined, therefore, as 'higher-order' directed posets,

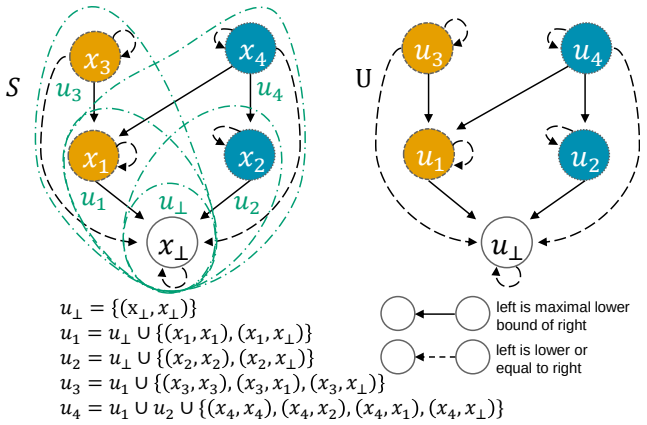

Figure 2. A BDP state S and equivalent EDP state U based on Fig. 1. An upward extension $u_i \in U$ is the set of its enclosed relations in S, to represent the formation history of $x_i \in S$.

i.e., will represent directed posets of directed posets. The resulting EDP RDT can then easily be shown to represent a state-based CRDT, for which an operation-based version can be constructed by using cryptographic hash functions as order-guaranteeing mechanisms. For clarity, we denote as Basic Directed Posets (BDP) the directed posets that are used as 'base layer' to construct the EDP RDT.

To fix a BDP, one selects a universe $\mathbb{X}$ of valid elements and a universe $\mathbb{R} \subseteq \mathcal{P}(\mathbb{X}^2)$ of valid relations. BDP states are finite, downward-directed posets $S = (X, R)$, $X \subseteq \mathbb{X}$, $R \in \mathbb{R}$, with a common bottom element $S^\perp$ (to be interpreted as the initial state of all correct replicas). As example, one can think of the universe $\mathbb{X}$ as application-layer messages, and the universe $\mathbb{R} \subseteq \mathcal{P}(\mathbb{X}^2)$ as causal orders, and a specific bootstrapping application-layer message as bottom element.

To define extend-only operations, we only want to allow single-element upward extensions of a BDP state S. We want finality as required validity criterion for extensions: Adding a new element to the BDP and adding the relations of the new element needs to be a single, atomic upward extension, i.e., cannot be changed later. The required finality property can be expressed as follows: when element x is the single element that has been added to gain state S, for all upward extensions S' of S needs to hold[1]: $x^{\downarrow S'} = x^{\downarrow S}$ and $R'|_{x^{\downarrow S'}} = R|_{x^{\downarrow S}}$.

To support this notion of finality, the obvious (of course, highly inefficient) approach is to bind a single-element upward extension, which extends $S = (X, R)$ with an element $x \in \mathbb{X}$, to all single-element upward extensions for elements y with $x \geq y$. This way, the history of extend-only operations is fixed and serves as identity-forming information for the new single-element upward extension. To be able to express sets of 'formation histories' of BDP elements, one has

[1] In contrast, the upward closure of an element and their relations might be never final: $x^{\uparrow S'} \supseteq x^{\uparrow S}$ and $R'|_{x^{\uparrow S'}} \supseteq R|_{x^{\uparrow S}}$. A new upper element can always be in transit or purported to have been in transit.

to move to sets of posets. Thus, for the EDP definition, we move 'one level up' and map those BDP posets to elements of the EDP as done in the following two steps. An illustration of the BDP to EDP state mapping based on the state in Fig. 1 is found in Fig. 2.

Step 1. Let the directed poset $S' = (X', R')$ denote an upward extension with single element $x \in \mathbb{X}$ of a directed poset $S = (X, R)$ of the BDP. The downward closure $x^{\downarrow S'}$ together with R' restricted to $x^{\downarrow S'}$ is a sub-poset $(x^{\downarrow S'}, R'|_{x^{\downarrow S'}}) \subseteq S'$ bounded by $x_\perp$ and x. Let $u_x \in \mathbb{R}$ denote the corresponding relation $u_x := R'|_{x^{\downarrow S'}}$ of such a BDP upward extension with x, and $X(u_x) := \{y \in \mathbb{X} | (y, y) \in u_x\} = x^{\downarrow S'}$ the set of reflexive pairs in u_x. We can derive from u_x the upward extension S' of S with x as $S' = (X \cup X(u_x), R \cup u_x)$. Thereby, as shorthand notation, we call u_x an upward extension as well. An upward extension $u_x \in \mathbb{R}$ is valid if $(X(u_x), u_x)$ forms a BDP sub-poset of S' bounded by $x_\perp$ and x.

Step 2. We can now move 'one level up' and define the EDP by using upward extensions u_x as elements and subset relations to form posets of those upward extensions. An EDP state $U \in \mathcal{P}(\mathbb{R})$ is the set of single-element upward extensions $U = \{u_\perp, u_1, \ldots\}$, i.e., the formation history of BDP state S. The initial EDP state is $U = \{u_\perp = \{(x_\perp, x_\perp)\}\}$, which is the upward extension of the empty set with the bottom element. An EDP state $U \in \mathcal{P}(\mathbb{R})$ is valid if $(U, \supseteq |_U)$ is a $u_\perp$-directed poset and $\forall u \in U\colon\ \mathrm{mlb}(u) \subsetneq U \wedge |X(u)| = |\{X(\bigcup \mathrm{mlb}(u))| + 1$, i.e., every upward extension in U also has its maximal lower bounds in U, and extends the BDP state with exactly one element that is not present in any of its maximal lower bounds. Fig. 3 provides an illustration of applying an upward extension, derived from the graph of Fig. 1. An EDP state U can be transformed back to a directed poset $S(U)$ on the underlying BDP via $S(U) = (X(\bigcup U), \bigcup U)$.

The above definition of the EDP leads to a *state-based CRDT*: the state space $\mathcal{P}(\mathbb{R})$ and set union $U_1 \cup U_2$ as join operation form a join-semilattice. As shown in [4], state-based CRDTs in the crash-fault model with a join operation that cannot fail in the Byzantine model can easily be proven to ensure SEC when faced with an arbitrary number of Byzantine replicas. Thus, while the modeling as directed posets might introduce some formalism, the above formulation of the EDP CRDT uncovers the simple set theoretic fundamentals, i.e., the state-based structure, of Byzantine-tolerant CRDTs.

From a practical point of view, a state-based EDP CRDT is highly inefficient: replicas continuously gossip their full state U, which will only increase in size. We now work up our way from the state-based formulation to an efficient operation-based EDP CRDT formulation that makes use of cryptographic hash functions for integrity protection. The construction follows [6, 7], its purpose here is to clarify the relationship between operation-based and state-based formalizations. We need two optimizations for which we give an intuition now, and a formalization next. First, an upward

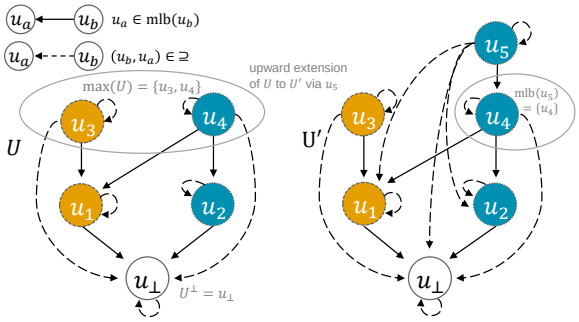

Figure 3. Example of a replica state U of an EDP, and state U' that results from an upward extension of U with u_5.

extension u_y represents a sub-poset of S' bounded by $x_\perp$ and y, thus, sending only the relations of that sub-poset instead of the state U' is sufficient. For the *operation-based formulation*, we then compress the subset, reducing worst-case update size from "depth" to "width" of U': We define an operation o consisting of y and the set of hashed maximal lower bounds of u_y in U'. As $\mathrm{mlb}(u_y) \subseteq U$, u_y can be reconstructed based on operation o and existing state U.

To chain y to its maximal lower bound hashes, a cryptographically secure hash function $h(a, b)$ is used that returns the hash of its chained arguments. The set of hashes of a set of maximal lower bounds $U'_{\mathrm{mlb}} := \mathrm{mlb}(u_y)$ is recursively defined as $H(U'_{\mathrm{mlb}}) := \{h(\max(X(u)), H(\mathrm{mlb}(u)))|u \in U'_{\mathrm{mlb}}\}$. To define the hash of the set of hashes in the second argument of h, one can concatenate its linearization gained by lexicographical sorting. An operation $o := (y, H_{\mathrm{mlb}})$ is then defined as compression of upward extensions u via $o(u) := (\max(X(u)), H(\mathrm{mlb}(u)))$. The hashed maximal lower bound set recursively protects the integrity of the directed sub-poset down to the bottom element $u_\perp$ against Byzantine nodes. To reconstruct upward extension u from operation o based on knowledge of current state U, we need the set of relations of all of $o.y$'s predecessors $P := (\bigcup U'_{\mathrm{mlb}} \subseteq U | H(U'_{\mathrm{mlb}}) = o.H_{mlb})$. Then, the upward extension is the union of the reflexive relation of y, the relation of y to its predecessor elements $X(P)$, and its transitive relation, i.e., its predecessor relations P. We formalize this reconstruction as $u(y, P) := \{y\}^2 \cup \{y\} \times X(P) \cup P$. An operation o can be locally applied as soon as all maximal lower bound upward extensions are part of the replica state, i.e., $o.H_{\mathrm{mlb}} \subseteq H(U)$. An operation o is applied to state U by $U' = U \cup u(o.y, P)$, inheriting commutativity from set union, which makes for an operation-based CRDT with state space $\mathcal{P}(\mathbb{R})$. The size of the update is now bounded by the maximum size of maximal lower bound set of any elements in the downward closure, instead of the overall state size.

The operation-based EDP is given in Algorithm 1. Encountering an unsatisfied assertion, the algorithm stops processing the function and returns with an error. Encountering

Algorithm 1 Op-Based EDP CRDT for BDP ($X \subseteq \mathbb{X}, R \in \mathbb{R}$)

Require: universe of valid BDP elements $\mathbb{X}$
Require: universe of valid BDP relations $\mathbb{R} \subseteq \mathcal{P}(\mathbb{X}^2)$
  state set of upward extensions $U \in \mathcal{P}(\mathbb{R})$
  initial $U \leftarrow \{u_\perp = \{(x_\perp, x_\perp)\}\}$
  query *bot* () : $u_\perp = U^\perp \in U$
  query *max* () : $U_{\max} = \max(U) \subseteq U$
  query *mlb* ($u \in U$) : $u_{\text{mlb}} = \text{mlb}(u) \subsetneq U$
  query S () : directed poset $S = (X \subseteq \mathbb{X}, R \in \mathbb{R})$
    $S \leftarrow (X(\bigcup U\}, \bigcup U)$

  update *extend* ($u \in \mathbb{R}$)
    generate ($u \in \mathbb{R}$)
      assert $\emptyset \neq u \notin U$
      assert $\emptyset \neq \text{mlb}(u) \subsetneq U$
      $o \leftarrow (\max(X(u)), H(\text{mlb}(u)))$
    effect ($o = (y \in \mathbb{X}, H_{\text{mlb}})$)
      await $H_{\text{mlb}} \subseteq H(U)$ ▷ await effect of updates $\subsetneq u$
      assert $H_{\text{mlb}} \neq \emptyset$
      $P \leftarrow \bigcup(U'_{\text{mlb}} \subseteq U | H(U'_{\text{mlb}}) = H_{mlb})$
      $U \leftarrow U \cup (\{y\}^2 \cup \{y\} \times X(P) \cup P)$

an unsatisfied await, the algorithm interrupts to await its potential future satisfaction, without blocking subsequent function calls. The `extend` function is used to extend the current set state U with a new upward extension $u \in \mathbb{R}$. The side-effect-free `generate` function generates the update operation o and broadcast it to all replicas, including itself. The received broadcast is then processed by the `effect` function, which awaits that all maximal lower bounds are eventually present in the current set state before proceeding to apply the operation. Byzantine tolerance of the op-based EDP CRDT is sketched in Section 2.4 and requires only some form of resilient broadcast as outlined in the next subsection.

2.3 Resilient Broadcast of Operation-Based Updates

The operation-based EDP formulation does not need the strong guarantees of a crash-/Byzantine-fault reliable broadcast. As explained in [7], mere eventual delivery of updates is sufficient for such Byzantine Sybil-resistant CRDTs, as long as correct replicas form a connected component. Due to the shared bottom element and relation directedness, broadcasting the set of maximal elements is sufficient, as missing elements can be iteratively queried from other replicas. An optimized broadcasting approach in this spirit is found in [7].

When the set of maximal upward extensions $\hat{U} = \max(U)$ of the replicas' current state U (or their space-efficient operation equivalents) is gossiped regularly, the broadcast is a state-based CRDT itself: The state space forms a join-semilattice with join of $\hat{U}_1$ and $\hat{U}_2$ being $\max(\hat{U}_1 \cup \hat{U}_2)$. While update size is close to EDP size in worst case, if all replicas are correct, update size converges quickly to approximately the number of involved replicas [6]. Byzantine replicas can send a large set of made-up extensions, but as the resulting state-based broadcast is a Byzantine Sybil-resistant CRDT [4], they can only attack performance but not correctness.

2.4 Byzantine Strong Eventual Consistency

To show Byzantine strong eventual consistency of the op-based EDP RDT, we use the strong eventual consistency (SEC) notion as defined in [8] consisting of three properties: a) **Self-update**, i.e., iff a correct replica generates an update, it applies that update to its own state; b) **Eventual update**, i.e., for any update applied by a correct replica, all correct replicas will eventually apply that update; and c) **Strong Convergence**, i.e., any two correct replicas that have applied the same set of updates are in the same state.

Lemma 1. *Let $U \in \mathcal{P}(\mathbb{R})$ be a directed poset corresponding to $S = (X \subseteq \mathbb{X}, R \in \mathbb{R})$, and U' (S' resp.) the resulting state after applying the update $o = (y, H_{\text{mlb}})$ for upward extension u. Then U' (S' resp.) are a partially-ordered extensions of U (S resp.) directed towards the same element $U^\perp$ ($S^\perp$ resp.).*

Proof Sketch. Invalid upward extensions are discarded via assertions, and then $U' = U$. Valid upward extensions only add a single element y to S. Due to P being the union of $u_\perp$-directed sub-posets and $\{y\} \times X(P) \subsetneq u$, $U'^\perp = U^\perp$ and also $S'^\perp = S^\perp$, i.e., $\perp$-directedness is kept. Due to the same argument, the partial-order properties also still hold for both U' and S'. □

Theorem 1. *Under the assumption of authenticated, integrity-protected channels and a connected component of correct replicas, the EDP is a Conflict-free Replicated Data Type even in face of an arbitrary number of Byzantine replicas.*

Proof Sketch. **Self-update:** With a broadcast as described in Section 2.3, the broadcasting replica receives the update without waiting for any acknowledgment. The `generate` function creates an update for which the await precondition in `effect` is immediately satisfied, which means that the `effect` function directly updates the replica's own state. Byzantine nodes have no attack vector to interfere with this process. **Eventual update:** Using a broadcast as described in Section 2.3, as soon as one correct replica receives an update, eventually every correct replica will receive the update due to the connected component of correct replicas. Due to authenticity and integrity, Byzantine nodes have no attack vector to interfere with eventual delivery of updates from correct replicas. Through the hash-chained maximal lower bounds, replicas can verify the integrity and completeness of the directed sub-poset of the downward closure up until the bottom element on every new update, without a Byzantine node being able to interfere. As soon as one correct replica can satisfy the await precondition for applying an update, i.e., has received the necessary downward closure for the element to add, it will eventually share the downward closure

with all correct replicas, for which the delivery precondition is then also fulfilled eventually so that they can proceed with the effect function as well. **Strong convergence:** Due to Lemma 1 and the commutativity of set union, all valid updates commute. Due to the `effect` function's assertions, only valid updates are applied. Due to an operation consisting of both its payload as well as the hashes of its maximal lower bounds, the integrity of the directed poset of the downward closure of an element is integrity-protected through hash chaining. Thereby, a Byzantine node that tries to attack consistency via equivocation with two operations (y_a, H^a_{mlb}) and (y_b, H^b_{mlb}) where either $y_a = y_b$ or $H^a_{\text{mlb}} = H^b_{\text{mlb}}$ is not successful, as all correct replicas will reject neither update as already received, but treat them as separate updates. As the validity checks are deterministic and the same on all replicas, Byzantine replicas cannot get an update applied on only a proper subset of correct replicas. □

2.5 Causal Extend-Only Directed Poset (CEDPs)

We denote the subtype of Extend-only Directed Posets with temporal events as set items together with their causal relation, in form of their happened-after-equal relation, as Causal Extend-only Directed Posets (CEDPs). As an EDP subtype, Theorem 1 also applies. Hash chaining can represent any upward-extend-only partial ordering, but inherently proves that the image happened-after the preimage, and thereby the happened-after relation in a Byzantine-tolerant way.

CEDPs are the set-based version, i.e., the reflexive-transitive closure, of causal Event DAG approaches like the Matrix Event Graph (MEG) [11]. A specific proof that the MEG is a Byzantine-tolerant CRDT is found in [6]. In [8], an Event DAG approach is used as transport layer to make arbitrary CRDTs Byzantine-tolerant. Their arguments for Byzantine tolerance and broadcast requirements also apply to CEDPs.

While the causal set approach to discrete, logical time is well-known in quantum physics [1], in distributed systems, the happened-before relation as defined by Lamport [9] is common. The happened-after-equal relation is the converse of the reflexive closure of the happened-before relation, and thereby the relations are interchangeable. The happened-after-equal relation however represents the direction of hash chaining and an ordering-based notion of equality. In Physics, an event is defined as a point in space-time. An operation $o = (x, H_{\text{mlb}})$ can be read as "event x happened at discrete logical time coordinate H_{mlb} and discrete space coordinate of the (implicit) replica identifier". The happened-after order is a superset of the causal order, i.e., the order in which events causally depend on each other. Either the application provides new events and their maximal lower bounds in causal order, or we use the happened-after order via $\max(U)$ as maximal lower bounds, which also covers the actual causal order.

3 Replicated Data Types Derived from EDPs

3.1 EDP-based Maps (EPMs)

3.1.1 EPM Specification. Based on the EDP replicated data type, we derive a map replicated data type we call EPM. Essentially, the universe $\mathbb{X}$ now represents key-value pairs, and as before the relation $\mathbb{R}$ defines how update operations are ordered. The EPM has a 'largest-element-wins' semantics with respect to an ordering of update operations based on $\mathbb{R}$.

Formally, we define $\mathbb{M} \subseteq \mathbb{X}$ as the set of valid key-value pairs of the form $(k \mapsto v)$, and a map $M \subseteq \mathbb{M}$ as 'injective' subset of all valid key-value pairs, i.e., a given key only has one unique associated value. We define the square bracket operator to query keys for map M as $(k \mapsto v) \in M \Leftrightarrow M[k] = v$ and a map update operator $\uplus$ for single-element upward extensions u that keeps injectiveness of M:

$$\forall u \in \mathbb{R}, x = \max(X(u)) :$$

$$M \uplus u := \begin{cases} M \setminus \{k \mapsto \_\} \cup \{k \mapsto v\} & \text{if } x = (k \mapsto v) \in \mathbb{M} \\ M & \text{if } x \in \mathbb{X} \setminus \mathbb{M} \end{cases}$$

The functions of the EPM replicated data type are given in Algorithm 2. The core of the algorithm is the *linearize* function, which linearizes a set $T \subseteq U$ partially-ordered by $\subseteq$ to a sequence T_n. We define a relation $R_\parallel$ of all pairs of upward extensions $(u_1, u_2) \in T^2$ that are 'parallel', i.e., cannot be compared using $\subseteq$, and are also minimal in the sense that no elements smaller than u_1 exist that cannot be compared to u_2, and vice-versa for u_2. Using a cryptographically secure hash function h, we define a strict linear order relation $R_h = \{(u_1, u_2) \in \mathbb{R}^2 | h(u_1) < h(u_2)\}$. The relation $R_\parallel \cap R_h$ then contains the necessary tie-breakings for the partial ordering $\subseteq$ to gain the linear ordering $R_l = (\subseteq \cup (R_\parallel \cap R_h))^+$. The hash function allows resolving ties without Byzantine nodes being able to compromise the outcome.

Algorithm 2 Operation-Based EPM Replicated Data Type

state set of upward extensions $U \in \mathcal{P}(\mathbb{R})$
initial $U \leftarrow \{u_\perp = \{(x_\perp, x_\perp)\}\}$

update *put* $((k \mapsto v) \in \mathbb{M})$
  $y \leftarrow (k \mapsto v)$
  $extend(u(y, \bigcup \max(U)))$ ▷ function of Algorithm 1
function *linearize* $(T \subseteq U) : T_n \in \mathbb{R}^n$
  $R_\parallel \leftarrow \{(u_1, u_2) \in T^2 | u_1 \not\subseteq u_2 \wedge u_2 \not\subseteq u_1$
    $\wedge \forall u \subsetneq u_1 : u \subseteq u_2 \wedge \forall u \subsetneq u_2 : u \subseteq u_1\}$
  $R_l \leftarrow (\subseteq \cup (R_\parallel \cap R_h))^+$ ▷ order $R_\parallel$ via R_h,
    ▷ $^+$ denotes the reflexive-transitive closure
  $T_n \leftarrow enumerate(T, R_l)$ ▷ set $T \rightarrow$ sequence T_n
query *get* $(T \subseteq U) : M \in \mathbb{M}$
  $T_n \leftarrow linearize(T^{\downarrow U})$
  $M \leftarrow T_0 \uplus T_1 \uplus \ldots \uplus T_n$

Note that Matrix also provides maps based on the CEDP similar to Section 3.1 to assign additional attributes to replicas and replicated objects, and uses them to provide access control, which we will discuss in Section 3.2.

3.1.2 Byzantine Eventual Consistency Verification.

Theorem 2. *Under the assumption of authenticated, integrity-protected channels and a connected component of correct replicas, an EPM is a Conflict-free Replicated Data Type even in face of an arbitrary number of Byzantine replicas.*

Proof Sketch. By reduction to the Byzantine strong eventual consistency of the underlying EDP. In a correct replica, the key-value pair is put in context through the set of maximal elements $\max(U)$ as maximal lower bounds of $(k \mapsto v)$, which satisfies the await-precondition and keeps the **Self-Update property**. Nothing has changed in the broadcasting and application of updates, and the `linearize` function does not interact with other replicas but takes all updates applied in the EDP into account, which keeps **Eventual Delivery**. The `get` and `linearize` functions as well as the map update operator $\uplus$ are deterministic and ignore invalid updates, so given the same downward-directed poset $(U, \supseteq)$, i.e. the same state of the EDP, they return the same map M, maintaining **Strong convergence**. □

3.2 Outlook on Systemic Access Control

Access control is usually enforced by a centralized entity in a strongly consistent way. In a distributed, weakly-consistent setting, we have to embrace that time is only a partial ordering, events and administrative changes happen concurrently, and there is no such thing as consensus on a total order of events or on which policies are in in effect "now" [12].

Previous works on access control for CRDTs mainly focus on filesystem-like cryptographically-enforceable access control in closed groups while our outlook is inspired by the granular authorizations and administrative permissions of Matrix [11] and similar systems [2].

The idea is to provide *systemic* access control, i.e., storing attributes needed for policies as well as policies itself in the types, thereby allowing integration of access decisions in the CRDT functions and gaining a decentralized enforcement. Such a systemic access control can be constructed by a composition of a CEDP – to gain a concept of logical time through the happened-after relation – and multiple EPMs.

The main challenge is to treat concurrent administrative changes securely. To deal with concurrent, conflicting changes, the key idea is that a concurrent update is never *rejected* if it was authorized for its downward closure, it just might be ignored on linearization if another change wins. A prototype model of such a composed data type is defined and discussed in Appendix A, together with a proof sketch of the CRDT property.

4 Conclusion & Future Work

We presented the Extend-only Directed Posets (EDPs) as a generalization of ideas around causal event DAG CRDTs in Byzantine environments and as a class of Byzantine-tolerant CRDTs of its own right. The state-based formalization shows the essence of these CRDTs, while the operation-based formalization makes the optimization for efficiency explicit. Based on the EDP definition, other Byzantine-tolerant CRDTs can be derived and combined with ease as exemplified by the map-based replicated data type and indicated by the use case for systemic decentralized access control.

This work-in-progress provides a step towards formal verification of strong eventual consistency in Byzantine environments of both EDP-based system designs as well as corresponding implementations (like Matrix), including security properties of decentralized access control.

References

[1] Luca Bombelli, Joohan Lee, David Meyer, and Rafael D Sorkin. 1987. Space-time as a causal set. *Physical review letters*, 59, 5, 521.

[2] Herb Caudill. 2023. Local first auth: decentralized authentication and authorization for team collaboration, using a secure chain of cryptological signatures. https://github.com/local-first-web/auth.

[3] Vicent Cholvi, Antonio Fernández Anta, Chryssis Georgiou, Nicolas Nicolaou, Michel Raynal, and Antonio Russo. 2021. Byzantine-tolerant distributed grow-only sets: specification and applications. (2021). DOI: 10.48550/ARXIV.2103.08936.

[4] Florian Jacob, Saskia Bayreuther, and Hannes Hartenstein. 2022. On CRDTs in Byzantine environments : conflict freedom, equivocation tolerance, and the Matrix replicated data type. In *Sicherheit 2022*. Gesellschaft für Informatik. DOI: 10.18420/sicherheit2022_07.

[5] Florian Jacob, Luca Becker, Jan Grashöfer, and Hannes Hartenstein. 2020. Matrix decomposition – analysis of an access control approach on transaction-based DAGs without finality. In 25th ACM Symposium on Access Control Models and Technologies. SACMAT 2020. DOI: 10.1145/3381991.3395399.

[6] Florian Jacob, Carolin Beer, Norbert Henze, and Hannes Hartenstein. 2021. Analysis of the Matrix event graph replicated data type. *IEEE access*, 9, 28317–28333. DOI: 10.1109/ACCESS.2021.3058576.

[7] Martin Kleppmann. 2022. Making CRDTs Byzantine fault tolerant. In *Proceedings of the 9th Workshop on Principles and Practice of Consistency for Distributed Data*, 8–15.

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

[9] Leslie Lamport. 2019. Time, clocks, and the ordering of events in a distributed system. In *Concurrency: the Works of Leslie Lamport*.

[10] Nuno Preguiça, Carlos Baquero, and Marc Shapiro. 2018. Conflict-free replicated data types (crdts). *arXiv preprint arXiv:1805.06358*.

[11] The Matrix.org Foundation C.I.C. 2022. Matrix specification v1.5. Tech. rep. https://spec.matrix.org/v1.5/.

[12] Mathias Weber, Annette Bieniusa, and Arnd Poetzsch-Heffter. 2016. Access control for weakly consistent replicated information systems. In *International Workshop on Security and Trust Management*. Springer, 82–97.

[13] Matthew Weidner, Heather Miller, and Christopher Meiklejohn. 2020. Composing and decomposing op-based CRDTs with semidirect products. *Proc. ACM Program. Lang.*, 4, ICFP. DOI: 10.1145/3408976.

A Integrating Systemic Access Control in a CEDP/EPM Composition

In this appendix, we compose a CEDP and one or more EPMs to provide access control for the corresponding replicated data types. Attributes needed for policy evaluation as well as the policies themselves are stored in the replicated data types and serve as basis for access decision and enforcement. Furthermore, also administrative access control tasks like changing rights and policies are governed by the attributes and policies within the replicated data types. Therefore, one does not need to resort to an external system for administering access control. We refer to this approach as *systemic* access control and call the composed data type *Access-Controlled Extend-only Directed Poset and Maps* (ACEDPM). The approach is inspired by the way the Matrix messaging platform handles access control.

Access control is defined in terms of subjects that perform different actions on objects. In our messaging example, a user subject performs actions like reading and writing chat messages, or querying and changing group membership, on objects like other users or a chat group. An event is the act of a subject performing an action on an object. We focus on consistency, and as read actions cannot harm consistency, we restrict ourselves to write actions in the following. We store (write) events as set elements in the data type. To gain a concept of logical time, we store the happened-after-equal relation among stored events, i.e., a CEDP. One-off events like chat messages are stored directly in the CEDP, while change events of attributes like group membership are stored via one or more EPMs on top of the CEDP.

Whoever initiates the CEDP and EPM replicated objects also defines "the rules" of the composed ACEDPM object, i.e., the initial attributes and policies as foundation on which all further regular and administrative events have to be authorized. Classical access control thinking assumes a centralized, strongly-consistent access control enforcement based on data and policy changes ordered by linear, real-world time. The challenge here is that CEDP and EPM only provide strong eventual consistency (SEC) as trade-off for availability under partition, and we want to integrate access control while keeping that trade-off. In decentralized, weakly-consistent access control, different replicas do not necessarily make the same decision at one point of real-world time, as data is not necessarily consistent and changes are ordered by partial, logical time. However, if the composed ACEDPM is still a Byzantine-tolerant CRDT, i.e., can still provide SEC, its state and access control is still secure in the sense that Byzantine replicas cannot force correct replicas to diverge in their state and access decisions permanently. The challenge is to treat concurrent administrative events securely. The ACEDPM keeps all concurrent events and linearizes them. We explain the approach in Appendix A.1, and show a proof sketch for SEC in Appendix A.2.

A.1 ACEDPM Specification

An ACEDPM deals with concurrent events in a CRDT-usual way: it accepts *all* valid events, here in a partial order, and then breaks ties among concurrent events, here using an access-control-based linear ordering among participants. We assume a peer-to-peer system architecture where one replica corresponds to a single user and is under the user's full control. An ACEDPM object stores events that consist of an action *act* of subject *sbj* (the sending replica) and the action's content *cnt*. All events have the ACEDPM replicated object as implicit object, and may have an explicit object *obj*, e.g., another user. We represent events as tuples $x = (act, sbj, cnt)$, or as $x = (act, sbj, obj \mapsto cnt)$ if an explicit object is present. A few examples: Alice writing a text message is $(\texttt{chat}, \texttt{Alice}, \texttt{"Hi!"})$. If Alice changes group membership to include Bob in the group, the event is $(\texttt{membership}, \texttt{Alice}, \texttt{Bob} \mapsto IN)$. If Alice sets her own access level to 2342, we get $(\texttt{level}, \texttt{Alice}, \texttt{Alice} \mapsto 2342)$.

In the following, we use two EPMs on top of the CEDP causal event set to integrate access control: one map M for a group membership attribute and one map L for an access level attribute. The first map M is accessed via a *getM* variant of *get* query function from Algorithm 2. Possible membership values are for example $\{OUT, IN, INVITE, BAN\}$. For authorization, we will require a subject to be IN, but e.g. Matrix also constrains membership state transitions. The second map L is used for an access level attribute of subjects and actions, accessed via *getL*. Levels are integers associated with subjects and actions, and allow a subject to store events for all actions with less or equal level. In addition, levels impose an access-control-based linear ordering on subjects and objects, as well as their respectively sent events. For more details on a possible access control system, please cf. Level- and Attribute-based Access Control as defined in [5].

For systemic access control, we change the EDP and EPM models to use an authorization function $authorized(u, T)$ for update $u \in \mathbb{R}$ and point in logical time $T \in \mathcal{P}(\mathbb{R})$ before applying updates, and an access-control-based priority relation R_a to break ties. We first explain why and where changes have to be made in the models, and then show algorithms for $authorized(u, T)$ and R_a. In the *effect* function of the EDP (c.f. Algorithm 1), we have to assert $authorized(u, \mathrm{mlb}(u))$ to ignore received updates that are not authorized by their maximal lower bounds. In the *get* function of the EPM (c.f. Algorithm 2), a linearized update T_i also must only be applied if $authorized(T_i, \bigcup_{j=0}^{j<i} T_j)$, to ignore stored updates that are not authorized by updates preceding the linearization. Finally, the priority relation R_a has to be used instead of the hash-based ordering R_h in the *linearize* function of the EPM, as R_h orders updates deterministic but arbitrarily. The strict linear ordering R_a instead prioritizes revocations, and all actions according to their subject's level, before falling back to R_h.

Algorithm 3 Level- and Attribute-Based Authorization

function *authorized* $(u \in \mathbb{R}, T \in \mathcal{P}(\mathbb{R})) : a \in 0, 1$
▷ whether u is authorized at point in logical time T
$x \leftarrow \max(X(u)), M \leftarrow getM(T), L \leftarrow getL(T)$
$group_a \leftarrow M[x.sbj] = IN$
$action_a \leftarrow L[x.act] \leq L[x.sbj]$
$object_a \leftarrow x.sbj = x.obj \vee L[x.obj] < L[x.sbj]$
$level\_cap \leftarrow x.act = \texttt{level} \Rightarrow x.cnt \leq L[x.sbj]$
▷ For "set level" actions, cap new level by subject level
$a \leftarrow group_a \wedge action_a \wedge object_a \wedge level\_cap$

The *authorized* function shown in Algorithm 3 decides whether an upward extension u is authorized by the attributes at the logical point in time T. Event x is authorized if the subject is a) member of the group, b) authorized for the action, c) authorized for acting on the (optional) object, and d) does not exceed their access level. The subject needs a level strictly greater than the object to prevent conflicting permission revocations to create inconsistencies: Same-level subjects cannot revoke each others' permissions.

The access-control-based priority relation R_a shown in Algorithm 4 is a strict linear ordering among all upward extensions $\in \mathbb{R}$. In first order, R_a prioritizes revocation actions over other actions that cannot reduce permissions, to ensure that revocations are applied before other concurrent upward extensions. In second order, for actions that are either both revocations or both other actions, it prioritizes actions according to their subject's level, to ensure that actions of subjects with higher level are applied before. In third order, for actions that are from subjects of equal level, R_a falls back to deterministic but arbitrary hash-based ordering.

By prioritizing revocations in the linearization of map updates, revocations gain "concurrent+causal for-each" semantics, a concept Weidner identified[2] based on [13]: a revocation acts against causally succeeding as well as concurrent map updates. When using the linearization of all elements as query function, revocations gain these desired semantics for all ACEDPM updates, i.e., for both CEDP and EPMs.

A.2 Byzantine Strong Eventual Consistency of ACEDPM

One might wonder whether concurrent revocations against each other could break eventual delivery or strong convergence. However, SEC is ensured because a concurrent update is never *rejected* if it was authorized by its downward closure, it just might be ignored in the linearization phase in case the other revocation wins. Therefore, all correct replica states will eventually contain both revocations and their downward closure, regardless of reception and application order. Linearization is only based on the relation of revocations to each other and their respective downward closure, whereby all correct replicas will eventually reach the same decision.

[2] https://mattweidner.com/2022/02/10/collaborative-data-design.html

Algorithm 4 Access-Control-Based Priority Relation R_a

$R_a \leftarrow \{(u_1, u_2) \in \mathbb{R}^2 | \; prior_a(u_1, u_2)\}$
function $prior_a\ (u_1, u_2 \in \mathbb{R}) : b \in \{0, 1\}$
▷ whether u_1 is prior to u_2 regarding access control
$x_1 = \max(X(u_1)), x_2 = \max(X(u_2))$
$L_1 \leftarrow getL(\text{mlb}(u_1)), L_2 \leftarrow getL(\text{mlb}(u_2))$
$b \leftarrow rvc(u_1) \wedge \neg\, rvc(u_2)$ ▷ prioritize revocations
if $rvc(u_1) = rvc(u_2)$ **then**
$b \leftarrow L_1[x_1.sbj] < L_2[x_2.sbj])$
▷ if both / neither is revocation, order by level
if $L_1[x_1.sbj] = L_2[x_2.sbj]$ **then**
$b \leftarrow h(u_1) < h(u_2)$
▷ if equal subject level, order by hash

function $rvc\ (u \in \mathbb{R}) : b \in 0, 1$ ▷ whether u is revocation
$x \leftarrow \max(X(u))$
$M_{pre} \leftarrow getM(\text{mlb}(u)), M_{post} \leftarrow M_{pre} \uplus u$
$L_{pre} \leftarrow getL(\text{mlb}(u)), \quad L_{post} \leftarrow M_{pre} \uplus u$
$b \leftarrow (M_{pre}[x.obj] = IN \wedge M_{post}[x.obj] \neq IN)$
$\vee L_{pre}[x.obj] > L_{post}[x.obj]$

Theorem 3. *Under the assumption of authenticated, integrity-protected channels and a connected component of correct replicas, the ACEDPM is a Conflict-free Replicated Data Type even in face of an arbitrary number of Byzantine replicas.*

Proof Sketch. Based on SEC of the underlying CEDP/EPM. **Self-update:** A correct replica only generates updates that pass its local, current authorization checks. Thereby, authorization does not hinder the replica from immediate self-update. **Eventual update:** An update generated by a correct replica is authorized based on its maximal lower bounds. Receiving correct replicas check whether an update is authorized only based on its maximal lower bounds. Thereby, they only filter out invalid updates that cannot have been sent by a correct replica. But they apply all updates authorized by their maximal lower bounds, even if a concurrent update has already revoked the permissions in the EPM state based on the receiving replica state's maximal elements. Resolving conflicts is moved towards linearization, and not solved through not applying updates eventually. **Strong convergence:** For the CEDP, the authorization is checked before applying an update, whereby authorization cannot make two replicas differ that have applied the same updates. For the EPMs, there is an authorization check inside the linearization that ignores updates if they are not authorized in linearization order. But, linearization is deterministic and independent from the order in which updates were applied. Therefore, replicas in the same CEDP state come to the same linearization, and thereby also to the same EPM states. □