Canton Network:
A Network of Networks for Smart Contract Applications

Digital Asset
updated January 2024

Abstract

Current smart contract networks suffer two constraints that significantly limit meaningful adoption by traditional financial institutions and other enterprises. First, they require every application to inherit the governance properties and the fully transparent privacy model of the underlying network. Second, applications compete for transaction throughput. In this whitepaper, we present the Canton Network, a smart contract network of networks that overcomes these limitations and enables each application provider to define their application's privacy, scaling, permissions, and governance while being part of a broader decentralized public permissioned network.

Introduction

Motivation

Several smart contract blockchain networks exist, but they all impose problematic constraints on assets and applications built on top of them. Specifically, on these networks, (1) all assets and applications share all data permanently and publicly with all users, (2) interaction with assets and applications is all-or-nothing; application operators cannot easily control how different users interact with their applications, (3) applications compete for global network resources; application operators cannot independently scale or choose on which infrastructure to deploy. Furthermore, fees are interconnected and unpredictable, with increased usage in one application raising costs for all users.

In contrast, most of the world's assets are heterogeneous, and are governed by unique rules for how users and businesses transact with them. Operators of applications that interact with these assets need control over the privacy, scale, service availability, and infrastructure cost of their applications. As such, the limitations of existing blockchain networks prevent the onboarding of the bulk of the world's assets and processes into an interconnected network, reducing public blockchains' ability to build substantial network effects outside of crypto-native assets.

For illustration purposes, we contrast existing public blockchain networks with the most successful public network, the Internet. The Internet is a heterogeneous network hosting independently operated applications; e.g., Wikipedia is fully public and coexists with gated banking portals on the same public Internet. High-volume, low-value services co-exist on the same network as low-volume, high-value services. Each application provider has the sovereignty to control its application's unique permissions, fees, scale, service levels, and more.

The Internet has unlimited horizontal scalability and grows with each application contributed to the network. As a service's traffic increases, the application provider adds resources; the growth of one service does not reduce the resources available to others. The heterogeneity of the applications found on the Internet helped it reach billions of users. Some users want to go to Wikipedia, and some want to access their banking portal; both go to the same place - the same Internet.

The lack of support for application heterogeneity in public blockchains has led to two significant negative outcomes. First, due to these networks' privacy limitations, only assets and data which can be part of the permanent public record are brought into public blockchains; we see experimentation with cryptocurrencies and non-fungible tokens (NFTs), but we don't see enterprises and governments bring traditional assets and records to public blockchains. Second, to overcome the contention on shared global resources, the bulk of application logic for blockchain applications is built 'off-chain' under the control of centralized application providers, meaning key functionality is operated off the shared network, negating the independent verifiability users expect of public blockchains.

Limitations of existing blockchain networks

Concretely, in Ethereum, and similar smart-contract networks, (1) data is fully transparent to anyone who can connect to the ledger, (2) there are strict, vertical limits on transaction capacity on layer 1, (3) layer 2s, rollups, and similar scaling channels lack transactional composability (4) issuers of assets forfeit control of that asset to a pool of pseudonymous validators. From a regulatory perspective, the data transparency and loss of control over assets make these networks unsuitable for use by financial institutions.

When smart contract applications hit transaction throughput limits, the results are catastrophic for the network and providers of smart contract applications on the network. For example, in 2017, Axiom Zen launched the wildly successful application CryptoKitties on the Ethereum network, exceeding 12% of all network transactions and causing massive network congestion[1]. As a result, other applications on Ethereum at this time experienced very high fees and latencies. Following this, the company behind Axiom Zen built and commercialized a new blockchain[2], moving away from the network upon which it built its success, and fragmenting the market.

The scaling limitations of existing blockchains are not inherent to the synchronization of application data and state; we propose a design that avoids these limitations. Existing public blockchains force all applications through a single ordering service, even where this isn't necessary. But this bottleneck is not required; for example, the order of text messages in one messaging application should be independent of the order of a social network feed of another application. These two applications should have independent ordering mechanisms for their state. Likewise, a network of smart-contract applications should allow for a similar localization of

[1] CryptoKitties is causing ethereum network congestion - Quartz
[2] CryptoKitties scratch Ethereum, find new life on Flow blockchain - Decrypt

transaction ordering. However, ordering across these applications must also be possible, as necessary, to be an interoperable smart contract network[3]. That requires a shared protocol to synchronize transactions composed across multiple applications. Current attempts to allow independent scaling with synchronization across applications, typically known as layer 2 protocols, rollups, and cross-chain bridges, add significant complexity and security problems[4], as evidenced by numerous recent hacks[5]. In contrast, the Canton Network enables applications across multiple subnets to natively interoperate between them without requiring a layer 2 protocol or asset bridge.

Our Contributions

In this paper, we introduce the Canton Network, a network of networks for smart contract applications with heterogeneity and scalability properties similar to the Internet, giving application providers control over their applications.

Like existing blockchain networks, the Canton Network provides real-time synchronization of sensitive data across participants. It has the privacy of a private blockchain on a public network; applications on the Canton Network see a single public ledger. The Canton Network has an expressive smart contract language called Daml, which has programmable privacy built into every asset or piece of data. The Canton protocol allows each application to scale independently, increasing availability and keeping fees low.

Thus, the Canton Network fills a major gap in the public ledger space: it has smart contracts on a single virtual ledger, similar to Ethereum, Solana, Tezos, and many more, and it has built-in privacy with selective transparency, similar to the bitcoin lightning network and Zcash. As of early 2023, financial institutions transact over $50 billion daily on limited-access subnets of the Canton Network.

Overview

In the rest of this paper, we describe the implementation details of the Canton Network at a high level. First, we describe the underlying technologies: Daml's data model, the Daml smart-contract language, and the Canton protocol. We then describe the Canton Network, a public network of permissioned subnets built using Daml and Canton.

Daml

Daml is an open-source[6] smart-contract language and framework designed to make it easy to develop, operate, and maintain multi-party applications in a way that preserves privacy and data consistency. More concretely:

---

[3] This is known as "partial ordering" - an ordering that doesn't specify the exact order of every pair of events, but only defines the order between certain items that depend on each other
[4] Ethereum Foundation Research Team AMA - Pt 7: 07 January, 2022
[5] E.g., Ronin ($615m), Binance ($570m), Wormhole ($320m), Nomad ($200m)
[6] https://github.com/digital-asset/daml

1. Daml provides concepts to capture rules that govern real-world business transactions. This helps programmers focus on business logic only while avoiding common security pitfalls.[7]
2. Daml allows to specify access and authorization policies within the smart contract code, making it easy to keep them in sync. Data is confidential by default, and access policies are easily defined so that the smart contract programmer can understand and maintain them effortlessly.
3. Daml supports application interoperability by enabling the composition of workflows into more complex ones, including when the workflows are already deployed across different applications on different networks. A party can unilaterally extend functionality by composing existing workflows into more complex ones. This ability for any party to extend functionality fosters organic growth of ledger usage and helps manage complexity. Daml enables the composition of workflows across a network of applications while maintaining the confidentiality and authorization requirements of each application.
4. Daml supports interoperability with other systems through integration tooling, including auto-generation of bindings for common programming languages, bridges to other blockchains, and common standard and domain-specific libraries.

Contracts

Daml defines a contract as a codified agreement on a workflow between multiple parties on the network; these parties are called contract signatories. In addition, other parties may observe the contract; these are called contract observers. A party can be an individual entity signing with a private key or a consortium signing with a flexible multi-signature confirmation policy; as such, assets can be issued, and contracts can be signed by central parties, or consortiums.

Transactions

A contract is created as part of a transaction, making the contract active. A subsequent transaction may archive the contract, rendering it archived[8]. To ensure consistency among network participants while maintaining each contract's privacy, we need transactions to exhibit two properties. First, we need a mechanism by which the different parties agree on the order of transactions to avoid diverging views. Second, various parties may be entitled to see distinct parts of the transaction based on the privacy definitions of the specific contracts; we call this sub-transaction privacy. Transactions must enable parties to have a partial view of the transaction, which they can verify, also known as a sub-transaction.

The state of active contracts is known as the Active Contract Set (ACS)[9] and is derived from a transaction graph. Every transaction in the graph may archive and create contracts[10], referencing all contracts upon which it depends. New transactions are atomic changes appended to the end of the transaction graph. The transaction may consist of multiple

[7] E.g., Reentrancy bugs in Ethereum's smart-contract language, Solidity
[8] This is known as the Unspent Transaction Output (UTXO) model
[9] The ACS is equivalent to bitcoin's UTXO set
[10] Transactions may include actions other than create and archive, omitted here for clarity. For a complete list of actions see Daml SDK documentation - Actions and Transactions

sub-transactions, with different parties accessing different sub-transactions. As such, different parties observe a different subset of the global ACS.

This model is similar to the UTXO transaction model used in bitcoin and other public blockchains, with two notable differences:

1. No party sees the full transaction graph of the entire network; instead, each party sees a subset of the graph, also known as that party's view. This partitioning of the global transaction graph contrasts to other UTXO blockchains such as bitcoin and Cardano, in which every party can see the entire graph.
2. A transaction does not always archive referenced contracts. Whether a transaction archives an input UTXO or not depends on the application logic, and is defined in Daml using the keywords `consuming` and `nonconsuming`. This option to keep a referenced contract active is in contrast to bitcoin and others, in which referencing a UTXO always archives it.

Transactions are structured as trees; this enables workflows to compose: the trees of existing workflows become the subtrees of combined workflows. Each party can validate its subtree and ignore the rest of the transaction.

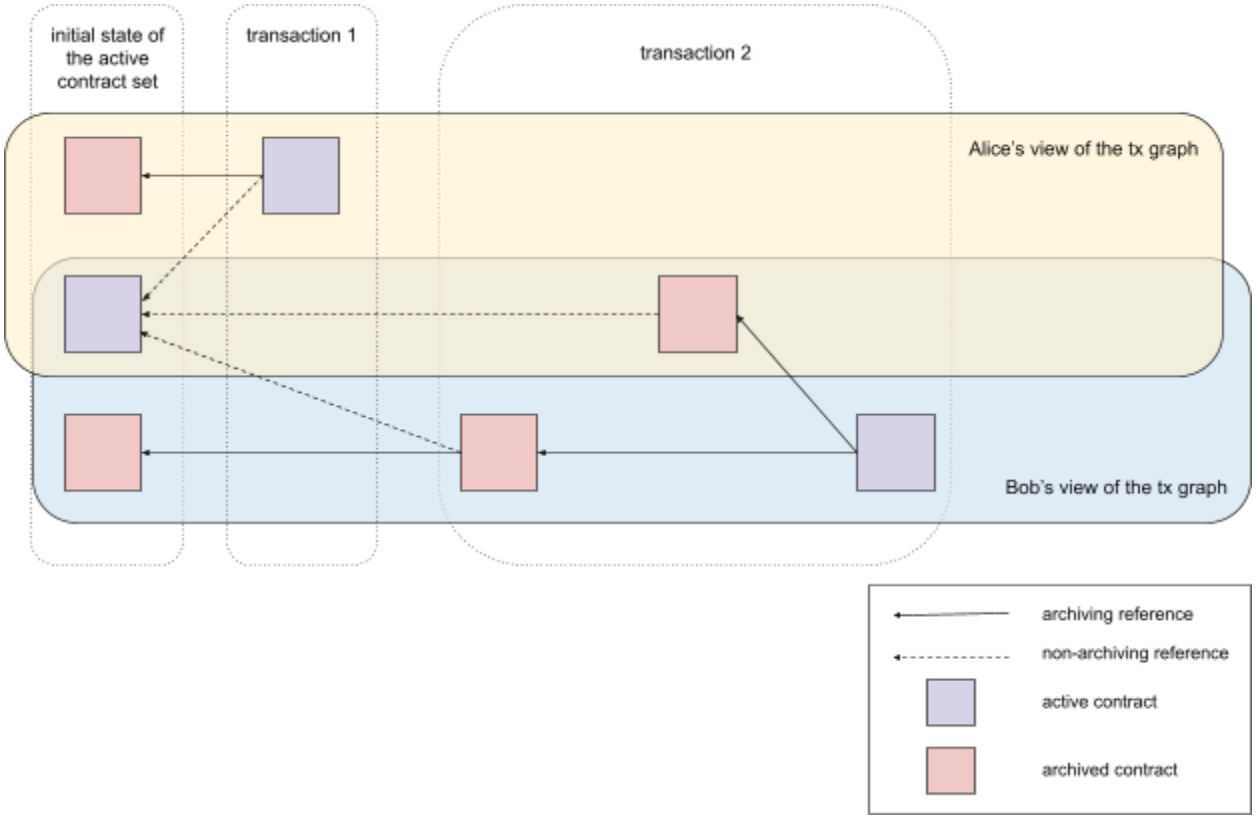

Figure 1: Example transaction graph with sub-transaction privacy. Alice and Bob each have only a partial view of the full transaction graph. Initially there are three active contracts, each party sees only two of them. Transactions 1 and 2, submitted by Alice and Bob respectively, evolve the Active Contract Set (ACS), archiving two of the initial contracts, creating two new active contracts, and

creating-and-archiving two transitory contracts. Following the two transactions, there are three active contracts, denoted in purple. Each party has access to only two of the three.

Said another way, the main difference between Canton's ledger model[11] and that of other blockchains is that, in Canton, each party sees only a subset of the ACS and a subgraph of the global transaction graph, also known as the party's view. This party-specific view is always a valid ledger[12] that can be verified locally by the party's node; a party need not trust any other party for verification. Upon receiving a transaction or sub-transaction, a party's node will verify three things: that the transaction is consistent with the party's view, that the transaction conforms with the logic written in the smart contracts, and that the transaction is properly authorized.

Beyond governing access control, we further utilize this partitioning of the ledger for parallel processing. Since transactions explicitly declare their dependencies, separate infrastructures can process independent transactions in parallel; this allows the Canton Network to scale horizontally by adding capacity as network demand fluctuates.

This model poses two challenges:

1. We must ensure that different parties' views of the global ACS are consistent; in other words, for every contract, the various parties who see it must always agree on whether it is active or archived. We will show, in the Consensus section, how the Canton protocol achieves this.
2. We need an application development model that makes it easy to work with these restrictive privacy controls. We will show how the Daml SDK achieves this.

Smart-contract language

Daml is a modern functional language featuring a static type system that can rule out many undesired behaviors and correctness errors at compilation time.

Developers define the data schema, workflow semantics, and transaction execution in contract templates. These are equivalent to object-oriented programming language class definitions and SQL database schemas. A template defines:

1. Arguments - data the contract stores
2. Choices - actions that parties can take on the contract. Choices are equivalent to methods in object-oriented programming classes or stored procedures in databases.
3. Authorization - signatories are parties who must authorize creating or archiving the contract; observers are other parties who can view the contract; controllers are parties who can take specific actions on the contract by exercising contract choices. A party can delegate its authority to another party to make particular choices. A party that delegates its authority sees whenever a transaction uses its authority.
4. Constraints - predicates that must hold for every contract of the template, denoted by the `ensure` keyword.

[11] For more details on the ledger model, see Daml SDK documentation - Ledger Structure
[12] For a formal definition of valid ledgers, see Daml SDK documentation - Valid Ledgers

Example:

```
template Iou
  with
    issuer : Party
    owner : Party
    currency : Text
    amount : Decimal
  where
    ensure amount > 0.0
    signatory issuer, owner

    choice Transfer : ContractId Iou
      with newOwner : Party
      controller owner, newOwner
      do
        create this with owner = newOwner
```

By default, exercising a choice archives the contract. In the case of the `Transfer` choice above, it also creates a new contract with a new owner. As an improvement on bitcoin's UTXO model and Cardano's eUTXO model, the developer can specify a choice as `nonconsuming`. Non-consuming choices do not archive the UTXO, thus reducing contention[13].

Daml's model of explicitly defining authorization enables manual intervention by a contract's stakeholders to rectify unexpected situations. Templates make it explicit where, how, and by whom intervention can happen during the execution, without requiring a priori knowledge of the exact type of intervention and without relaxing any security guarantees. Signatories can jointly agree to archive, upgrade, or create new contract instances as long as there is unanimous consent. If any of the signatory parties are consortiums, their consent is governed by the underlying consensus protocol of that party/consortium and may, for example, require a ⅔ supermajority instead of unanimous consent. Observers are parties entitled to be notified of, and can independently validate, any such changes but whose authorization is not required. All actions on contracts - their creation, archival, and calls to choices - are events in transaction trees and form a complete and non-repudiable audit log of all changes. This ability to change contracts post hoc, with the appropriate authorizations, enables application providers to upgrade data, processes, and operating procedures, due to unforeseen events. For example, to deal with regulatory or judicial decisions which require retroactive changes to business transactions.

Daml organizes templates in modules and packages. Packages can depend on other packages, including across applications which may be deployed to multiple networks. This ability to depend on packages across applications on different Canton subnets enables an open architecture, where parties can combine workflows with other parties like building blocks.

[13] The ability to reference a contract without archiving it is useful when referencing relatively static data such as, for example, daily interest rates, or the existence of a trading agreement between counterparties

For an in-depth review of Daml's design, see Daml's white paper[14]. For an overview of Daml tooling, see https://docs.daml.com.

Ledger model

Daml enables parties to exchange value (in the form of smart contracts) in a way that is unique among currently available technologies. A smart contract update is nothing more than an authorized and validated update to entries on a ledger. The fundamental challenge when trying to accomplish this update without a trusted, central intermediary is ensuring that the ledger entries reflecting the smart contracts are accurate and can be proven to a third party. A common approach to address this challenge is to decentralize the ledger among the parties on the network by requiring every party to hold and update a copy of the ledger for the entire network. A consensus mechanism is used to ensure accurate replication of the ledger to all parties. But this leads to networks devoid of privacy with hard caps on scalability.

As discussed in the Transactions section, Daml's ledger data model takes a different approach to address these privacy and scalability challenges. In Daml's ledger model, the ledger is not fully replicated among the parties; it is segmented according to privacy rules, and each party stores only its view, or shard, of the ledger. As a result, there is no ledger view common to all parties in the network. Instead, there is a ledger for each party that includes only the contracts of that party. As a result, instead of one ledger that all of the parties in the network must replicate, each party to a transaction updates its ledger to reflect that transaction. However, this creates a problem: if the record of smart contracts is spread across many ledgers, each visible to a certain party, then it would be difficult for any party on the network to know whether their smart contract is accurate.

Daml solves this problem by ensuring that each party's view is a subset of a single global, virtual ledger. In other words, conceptually, all parties of Daml ledgers perceive a single ledger while each party has read-access only to a subset of this ledger's state. This global ledger is virtual in the sense that it does not exist in any one data store. Since, conceptually, all users are reading from the same ledger, all users have a consistent view of any application state they share, for example, ownership of assets. The Canton protocol, described below, is the mechanism that ensures that the views of all properly functioning nodes in the network are consistent subsets of a single valid, global, virtual ledger. All of this is done while ensuring that no party sees or stores information to which it is not a party. As a result, parties can transfer digital assets with the confidence that the party transferring the digital asset actually owns that asset while also being certain that no other party on the network will know of the transfer unless such party is explicitly permitted to do so.

Daml's fully decentralized and party-centric ledger model enables decentralization in one additional way – rather than being structured as a single network, Daml enables users to create

[14] Alexander Bernauer et al., "Daml: A Smart Contract Language for Securely Automating Real-World Multi-Party Business Workflows" (arXiv, March 7, 2023), https://doi.org/10.48550/arXiv.2303.03749.

their own subnets. A party can connect to a single or multiple subnets. And if a party is connected to multiple subnets, the Canton protocol can synchronize digital asset transactions across them. Thus, the Daml Ledger Model enables a network of networks. This design ultimately enables privacy, performance, and scalability in a decentralized environment.

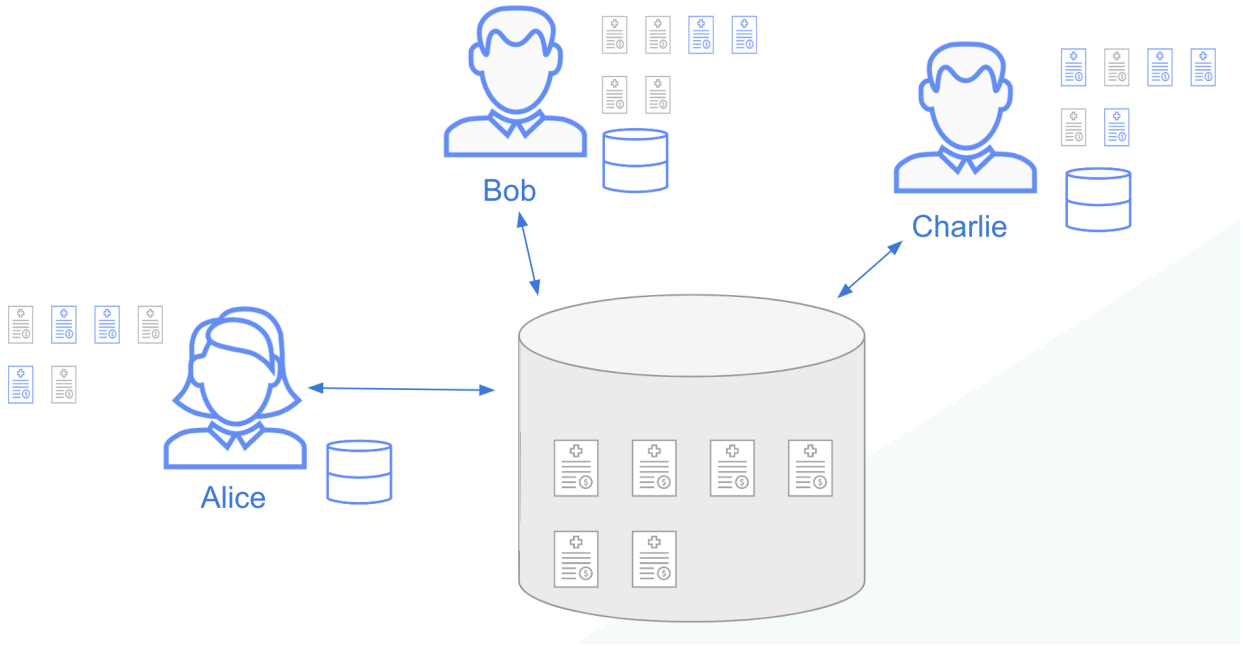

Figure 2: Canton's ledger model. Each party has its own valid ledger, which is kept consistent by the Canton protocol with the global ledger. The global ledger is virtual, i.e., it is not stored in its entirety by any single party. In this example, the Active Contract Set (ACS) consists of six contracts, but each party only has access to a subset of 2-4 contracts, denoted in blue.

For an in-depth review of Daml's Ledger Model, see Daml SDK documentation - Daml Ledger Model

Canton

Canton is an open-source[15] privacy-enabled blockchain protocol.

Canton implements Daml's ledger model as described above. Canton currently supports the Daml language, though it can support any language with a similar hierarchical sub-transaction privacy model.

Network topology

Nodes in the Canton Network are called participant nodes. A user or company, represented in Daml as a `Party`, deploys one or more participant nodes; these participant nodes act on behalf of that `Party`. To transport data between nodes and determine the order of messages, each

[15] https://github.com/digital-asset/canton

participant node connects to one or more private or public Canton Service Providers (CSP) which operate a Canton component called a synchronization domain ("sync domain"). Thus, connecting to sync domains allows a `Party` to transact with all other parties whose participant nodes are connected to a common sync domain. Anyone can become a CSP and deploy sync domains at will; reasons to deploy new sync domains may include increasing throughput, reducing latency, requiring data transport only through certain jurisdictions or certain blockchain networks, or other operational concerns. To promote privacy and net neutrality[16], data in transit over sync domains is encrypted, preventing CSPs from accessing message contents. Sync domains can be thought of as highly available, fault-tolerant messaging queues between participant nodes that sequence, timestamp, and serve encrypted messages to participant nodes. CSPs can be single entities or "virtual CSPs" in which a consortium of parties runs a distributed sync domain[17]. At launch, the Canton Network will have at least one open virtual CSP (vCSP) that is run by a consortium and accepts connection requests from any participant node. Application providers can choose to use this open vCSP or any other CSP. As such, Canton creates a mesh network of composable Daml applications in which each application may make different tradeoffs between trust, access control, and operational complexity.

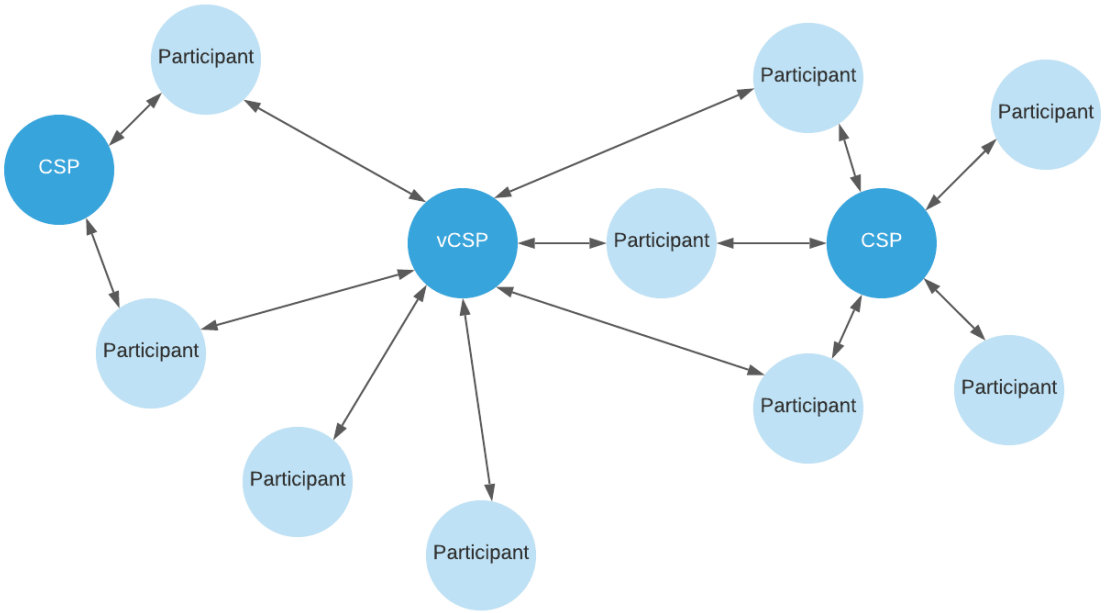

Figure 3: Canton Network topology. Participants connect to each other via Canton Service Providers (CSPs) or consortium vCSPs. Parties can transact if their participant nodes are connected to a common CSP or vCSP. No single node processes all network transactions.

While single nodes have processing and storage limitations, the Canton Network has no intrinsic scaling bottlenecks: a participant node processes only its data and workflows, which different sync domains synchronize in parallel. Parties connect to any sync domains they choose as long as the CSP operating the sync domain accepts them. Open sync domains

[16] https://en.wikipedia.org/wiki/Net_neutrality
[17] For more information on distributed Domains, see Proof-of-Stakeholder: Consensus with privacy

accept any well-formed requests to join. Canton enables a public permissioned network, in that anyone can deploy a Canton sync domain, thus becoming a CSP, for any reason. Sync domains are not silos: parties who share one or more common sync domains can compose higher-order workflows, including atomic transactions across multiple applications, processed via a sync domain selected by the applications. Contract signatories and observers control which sync domain will synchronize their contracts and can choose to reassign which sync domain sequences a given contract, avoiding sync domain lock-in or censorship[18]. The diagram below illustrates the sequence of events involved in reassigning synchronization responsibility for a contract from sync domain 1 to sync domain 2:

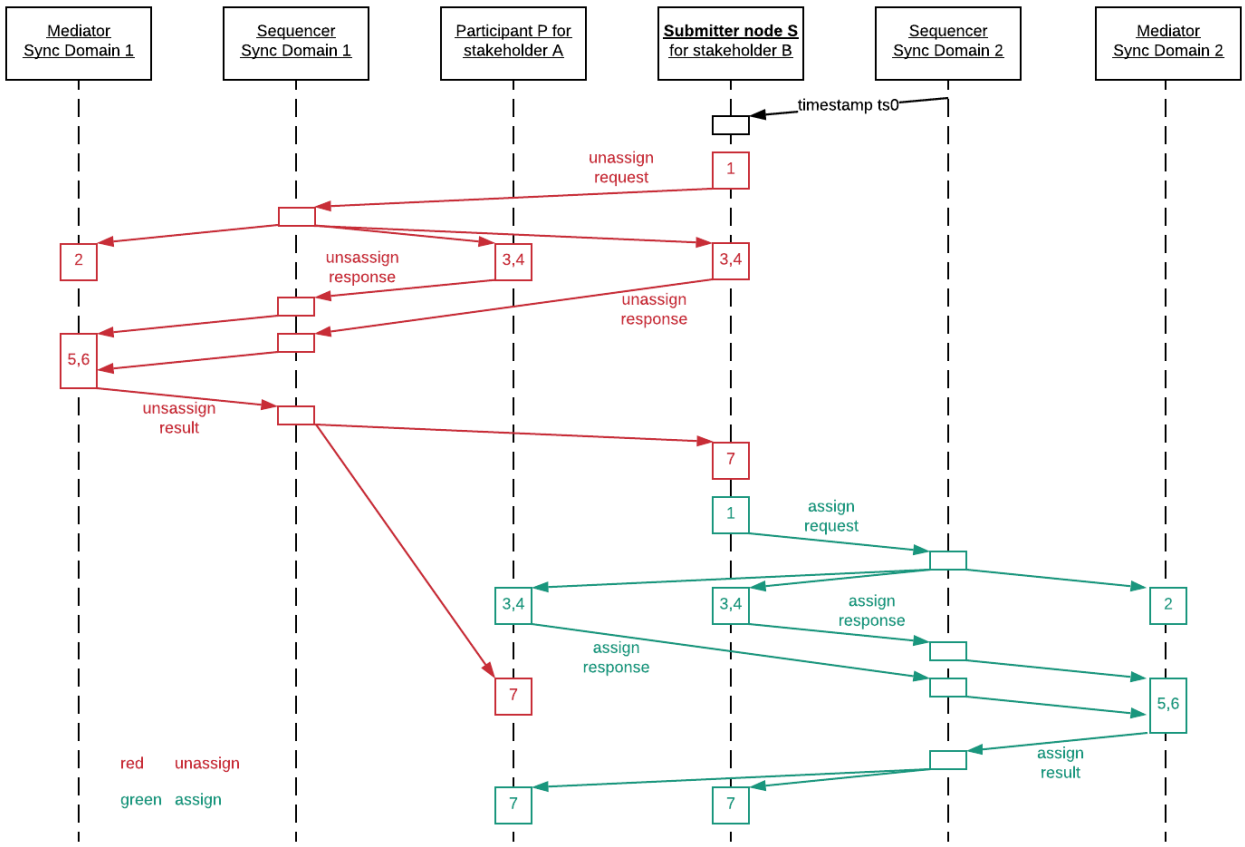

Figure 4: Reassigning of sync domain - sequence diagram. Contract signatories can jointly agree to reassign the role of synchronizing a contract from one sync domain (and CSP) to another.

The Canton Network has no single centralized governance or policies for access and usage; each constituent node or subnet sets its own policies.

[18] The diagram illustrates a regular domain reassignment. If the origin sync domain is unresponsive, a different protocol is used. See the Canton documentation for more details.

Data pruning

Canton provides history-pruning and redaction capabilities for its log. Participant and sync domain operators can configure their nodes to store or prune historical cryptographic data, allowing them to trade-off between auditability and the ability to delete archived contracts to comply with right-to-forget regulations such as the European Union's General Data Protection Regulation[19] (GDPR). Historical data can be moved to offline storage to retain full and immutable audit logs while reducing data storage costs and increasing sustainability of production environments. Canton nodes continuously exchange cryptographic commitment to their shared state, so parties remain secure against attempted repudiation by malicious or malfunctioning counterparties even when configured to prune historical data. Thus, individual node operators may trade off maintaining full and immutable historic auditability versus other operational and regulatory compliance requirements[20].

Proof-of-Stakeholder: Consensus with privacy

Canton's primary goal is to provide consistent data across parties. In other words, Canton aims to achieve consensus amongst parties on the active contracts on which they are joint stakeholders, and on the validity of the transactions that led to this state. The standard approach to consensus is state machine replication, where all participants replicate the same global state. However, replicating the entire global state is not acceptable for privacy and scalability reasons. Instead, Canton's proof-of-stakeholder consensus protocol is split into two layers of consensus. To achieve consistency along with privacy, the first consensus layer is a two-phase commit protocol that replicates each contract to the contract's stakeholders[21] while concurrently enabling each stakeholder to validate the transaction. Conceptually, this can be thought of as having a replicated state machine for every subset of parties[22]. For this first layer to commit transactions consistently, nodes must agree on the order in which conflicting transaction requests are applied to the ledger. Therefore, the second consensus layer is a sequencing protocol that receives encrypted transactions and determines a timestamp[23] for each transaction. This sequencing layer can be run on a central CSP or, when connected to a virtual CSP's distributed sync domain, this sequencing protocol runs as a replicated state machine secured by a Byzantine Fault Tolerant (BFT) consensus algorithm. Thus, the virtual CSP determines a total order on transaction requests within a sync domain, and transaction processing is deterministic.

[19] https://gdpr.eu/right-to-be-forgotten/
[20] See Canton Protocol white paper.
[21] This is similar to the atomic commit protocols of sharded databases
[22] In technical terms, every subset of parties defines a projection of the global ledger. Any projection of the global ledger is itself a valid ledger
[23] This is a vector clock, not a real-world clock time

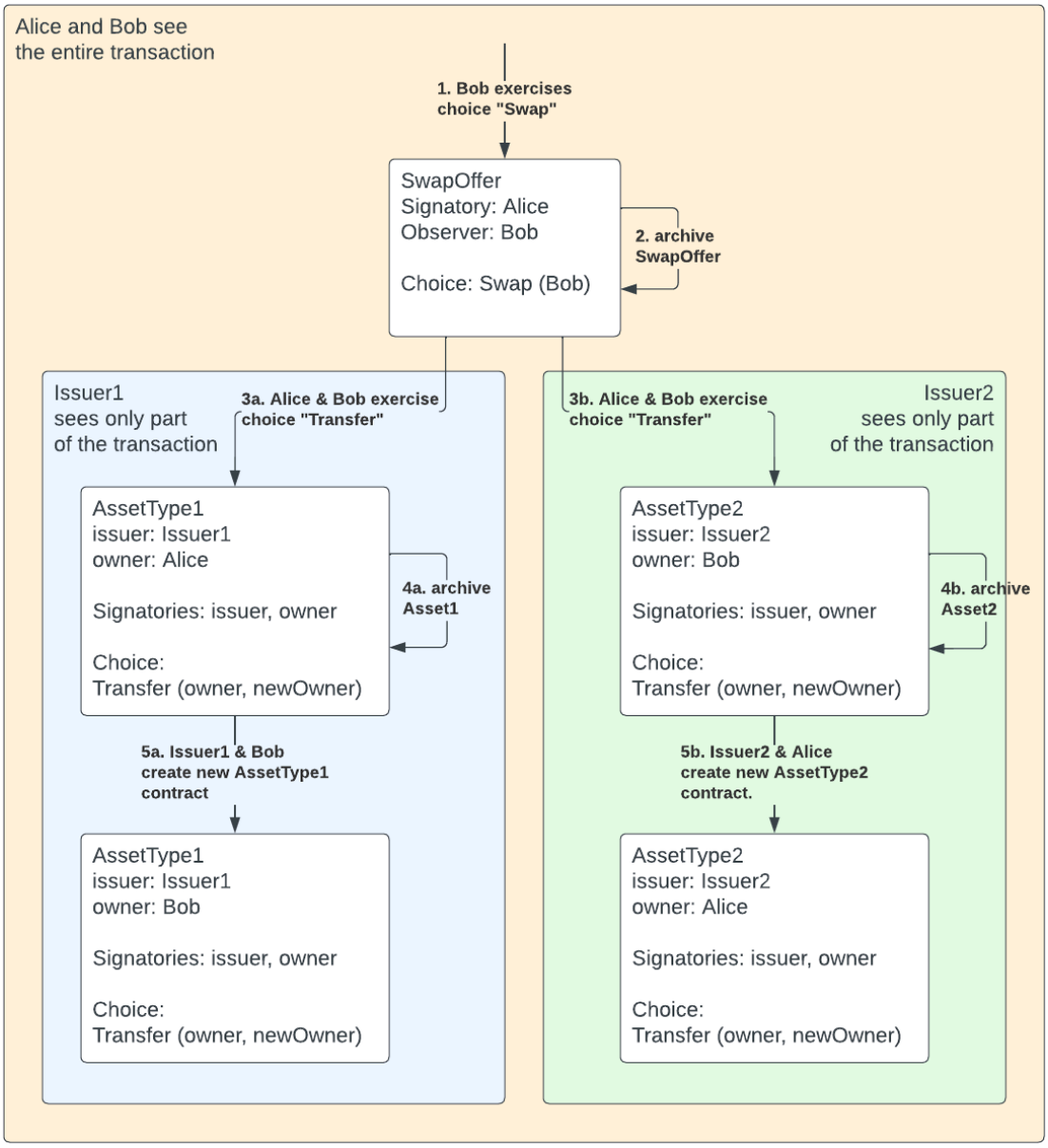

Figure 5: Atomic asset swap transaction. The asset swap will succeed only if the signatories agree to both sub-transactions. Otherwise, the asset swap is rejected and the assets are not transferred. Alice and Bob see the entire transaction, while Issuers 1 and 2 are each entitled to view only parts of the transaction.

Let us consider the example transaction shown in the figure above. Before this transaction is applied to the ledger, three contracts are active: (AssetType1) Alice owns an asset issued by Issuer1, (AssetType2) Bob owns an asset issued by Issuer2, and (SwapOffer) Alice has proposed an offer to Bob to swap their assets. By accepting the swap offer, Bob will cause all three contracts to be archived, and two new contracts will be created[24]: (AssetType1) Bob will own an asset issued by Issuer1, and (AssetType2) Alice will own an asset issued by Issuer2.

[24] In UTXO blockchain terms, the transaction consumes three UTXOs and creates two UTXOs

Some rules encoded in the contracts are omitted for brevity; for example, asset amounts must be preserved throughout the transaction.

In this transaction, Bob executes the “Swap” choice [1], which archives the SwapOffer [2]. Each choice defines whose authority is required to call it (denoted in parentheses in the diagram above). Alice created the offer such that only Bob’s authority is required to exercise the “Swap” choice. Since Alice is a signatory to the SwapOffer, the Swap choice can use her authority alongside Bob’s to call the “Transfer” choice on both assets [3], which subsequently archives these asset contracts [4] and creates two new asset contracts with the owners swapped [5]. Since the issuers were signatories of the two assets, their authority can be used to call the “Transfer” choices on those assets.

Not all parties involved in the transaction can determine if it is valid, but every party can determine that the sub-transaction they are allowed to see is valid. For example, Issuer2 is only entitled to see assets that it has issued and not assets issued by Issuer1 or the bilateral SwapAgreement between Alice and Bob. And, Issuer1 should only accept the transaction if it is sure that Alice authorized the transfer. To ensure resilience against malicious participants, Canton achieves consensus by processing transactions in two steps. First, the submitter sends a confirmation request to every other signatory, attaching only the part of the transaction the other signatory should see, encrypted. Each signatory decrypts their sub-transaction, checks whether the request is valid, and responds with a signed confirmation response. Their checks ensure two things: First, they ensure that the Daml authorization model is respected and that the correct parties are notified of the transaction, thwarting any malicious behavior by the submitter. Second, they prevent double-spending. Attempts to double spend are not necessarily a sign of a malicious submitter; they can occur under conflicting concurrent workflows. The sync domain’s total ordering and Daml’s determinism allow everyone to resolve conflicts in the same way. This way, the transaction is applied atomically across all signatories or rejected; while each party has a different view of the ledger, consensus is maintained.

Application composability

In Canton, two or more applications can compose and rely on atomic transactions[25] even if they are synchronized via different sync domains. Thus, for example, two central banks may each synchronize local currency transactions using country-local CSPs, while owners of these currencies can still atomically swap them in a cross-Domain transaction[26].

Global composability could be achieved, similar to other blockchains, by having a single global Canton synchronization sync domain that allows atomic composition of arbitrary workflows. However, having multiple sync domains is beneficial for multiple reasons. For example, companies and individuals may want more control over their network resources. A single global sync domain would impose a high communication latency for some or all participants. Multiple sync domains can help increase throughput: requests from different sync domains can be

[25] https://en.wikipedia.org/wiki/Atomicity_(database_systems)
[26] See “Multiple Domains and global composability” section in the Canton protocol white paper

processed completely in parallel. There might also be operational concerns; for example, for critical workflows, a new sync domain with restricted access can be used. Finally, different sync domain operators may charge differently for their services. In the example above, transactions in each local currency would be processed completely within the jurisdiction of the central bank's country.

However, having multiple sync domains opens up a new challenge of how to compose workflows across sync domains. In Daml, composing workflows specifically means that contracts synchronized via different sync domains can be used within a single transaction. Without such an ability, we would not truly solve the composability problem: this would create multiple siloed networks with a single sync domain each, instead of multiple subnets of a single interoperable network.

Canton guarantees the atomicity of cross-subnet transactions. Since different sync domains have no common notion of ordering between the different sub-transactions that each sync domain processes, reconciling atomicity with resilience can become impossible. To overcome this, Canton allows cross-subnet transactions only whenever there exists at least one sync domain to which all transaction participants are connected. Contract changes required by the cross-subnet transaction are synchronized via this common sync domain. Furthermore, Canton makes it possible to change which sync domain provides the synchronization service for a contract. This synchronization reassignment marks a different sync domain as the new authority for ordering actions on the given contract.

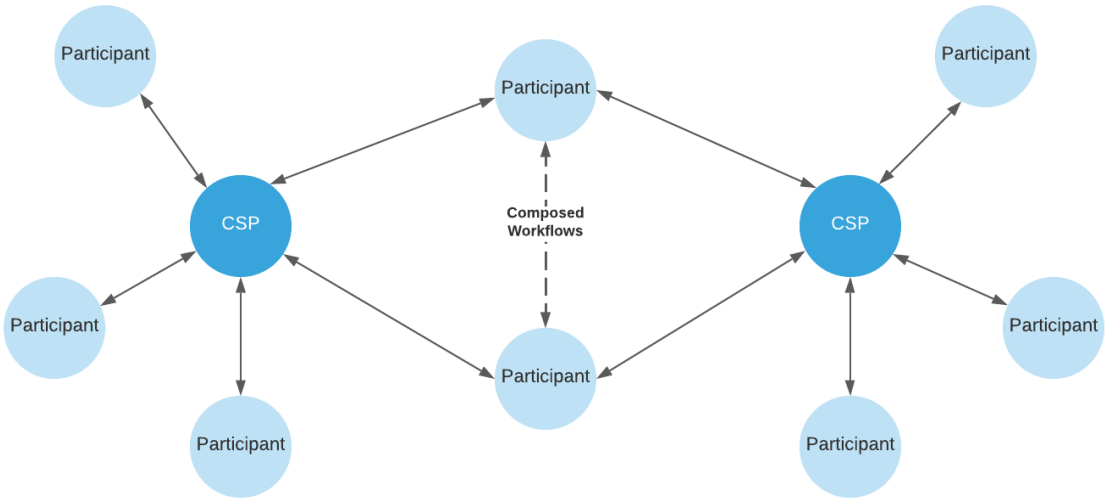

Figure 6: Example network topology. Participants connected to more than one sync domain can compose atomic cross-subnet transactions, enabling Participants to build transactional workflows across multiple applications and networks.

For an in-depth review of the Canton protocol's design, see the Canton Protocol White Paper.

Canton Network

The Canton Network is the network of networks spanned by all sync domains and their connected participant nodes. It is a global system of interconnected participant nodes, sync domain nodes and the private, semi-private, and public smart-contract applications deployed to these nodes.

Canton Network users act in three main roles:

1. Application Providers - application providers build and maintain smart-contract applications. They operate one or multiple participant nodes, application backend infrastructure, and frontend web interfaces for those applications. Application providers optionally act as CSPs for their applications or they can use the service of other CSPs.
2. Application Users - most users interact with applications via application programmable interfaces (APIs) and web user interfaces (UIs). Users must have a participant node and can choose to operate their own participant nodes or use hosted nodes managed by others[27].
3. Canton Service Providers (CSPs) - infrastructure providers, who are typically also application providers, connect participant nodes by operating a Canton sync domain.

The Canton Network consists of multiple subnets. A Canton subnet is any one or more participant nodes that can transact with other participant nodes via one or more sync domain nodes.

The network will become publicly available with the launch of a public sync domain operated by a vCSP that will accept all incoming connection requests from participant nodes. A group of independent companies called the Super Validator Collective (SVC) will run this public sync domain.

The SVC will charge a fee for network bandwidth consumption. Fees are fixed per unit of bandwidth and denominated in United States dollars; thus, network users have predictable network usage costs. The SVC may revise these fees occasionally. Use of the SVC public sync domain is optional; any network participant may choose to launch additional public or private sync domains with different payment mechanisms and fee structures.

[27] Users can run their own participant nodes while interacting with applications via untrusted UIs served by application operators. Participant nodes include an application permissions manager, which allows the user to grant limited permissions to application providers to call smart contract choices on the user's behalf. Thus, Canton enables distributing control over data and assets to users' nodes without requiring application providers to distribute all infrastructure.

Conclusion

In this paper, we introduced the Canton Network, a novel smart-contract network of networks. We started from the constraints of existing public blockchains, namely the lack of privacy and scaling limitations introduced by the globally replicated state, and demonstrated how Canton creates a global network without these limitations. We further discussed the upcoming opening of the Canton Network to public use with the launch of a virtual Canton Service Provider (vCSP) operated by a Super Validator Collective (SVC). Permissioned Canton networks currently in production will be subnets of the public Canton Network, making the Canton Network the first public permissioned blockchain for institutional assets.