AnyLog: a Grand Unification of the Internet of Things

Daniel J. Abadi
University of Maryland
abadi@cs.umd.edu

Owen Arden
University of California,
Santa Cruz
owen@soe.ucsc.edu

Faisal Nawab
University of California,
Santa Cruz
fnawab@ucsc.edu

Moshe Shadmon
AnyLog
moshe@anylog.co

ABSTRACT

AnyLog is a decentralized platform for data publishing, sharing, and querying IoT (Internet of Things) data that enables an unlimited number of independent participants to publish and access the contents of IoT datasets stored across the participants. AnyLog provides decentralized publishing and querying functionality over structured data in an analogous fashion to how the world wide web (WWW) enables decentralized publishing and accessing of unstructured data. However, AnyLog differs from the traditional WWW in the way that it provides incentives and financial reward for performing tasks that are critical to the well-being of the system as a whole, including contribution, integration, storing, and processing of data, as well as protecting the confidentiality, integrity, and availability of that data. Another difference is how Anylog enforces good behavior by the participants through a collection of methods, including blockchain, secure enclaves, and state channels.

1. INTRODUCTION

The world wide web (WWW) has had an extraordinary impact on our day-to-day lives. An enormous amount of information is available to any participant at extremely low cost (usually this cost is paid via one's attention to advertisements). However, the interface is fundamentally limited. A user must either have pre-existing knowledge of the location of the needed information (e.g., the correct URL), or use a search interface which generally attempts to match words in a search query with the natural language found on the web. It is totally impossible to query the entire WWW with a single SQL query (or any other structured query language), and even if you could, the data available is not published in a format which would be amenable to such queries.

AnyLog aims to create a new WWW for structured data (e.g., data that fits in rows and columns of relational tables), with an initial focus on IoT data. Anybody can publish structured data to AnyLog using their preferred schema, and they retain the ability to specify the permissions of that data. Some data will be published with open access—in which case it will be queryable by any user of AnyLog. Other data will be published in encrypted form, in which case only users with access to the decryption key may access it.

AnyLog is designed to provide a powerful query interface to the entire wealth of data produced by IoT devices. Questions such as: "What was the maximum temperature reported in Palo Alto on June 21, 2008?" or "What was the difference in near accidents between self-driving cars that used deep-learning model X vs. self-driving cars that used deep-learning model Y?" or "How many cars passed the toll bridge in the last hour?" or "How many malfunctions were reported by a turbine of a particular model in all deployments in the last year?" can all be expressed using clean and clearly specified SQL queries over the data published in AnyLog from many different data sources.

We choose the Internet of Things as our initial focus for AnyLog since the data is machine-generated and usually requires less cleaning than human-generated data. Furthermore, there are a limited number of unique devices, with typically many instances of a particular unique device. Each instance of a device (that is running a particular software version) produces data according to an identical schema (for a long period of time). This reduces the complexity of the data integration problem. In many cases, device manufacturers can also include digital signatures that are sent along with any data generated by that device. These signatures can be used to verify that the data was generated by a known manufacturer, thereby reducing the ability of publishers to profit off of the contribution of "fake data" to the platform. Despite this initial focus on the Internet of Things, our long term goal is to expand AnyLog and enable any structured data to be published and queried using the platform.

Unlike the previous generation of decentralized database systems [15, 24, 9], publishers on AnyLog receive a financial reward every time the data that they contributed participates in a query result. This reward accomplishes three important goals: (1) It motivates data owners to contribute their data to the platform (2) It motivates data owners to make their data public (since public data will be queried more often than private data) (3) It motivates data owners to use an existing schema to publish their data (instead of creating a new one).

The first goal is an important departure from the WWW, where data contributors are motivated by the fame and fortune that come with bringing people directly to their website. Monetizing this web traffic through ad revenue disincentivizes interoperability since providing access to the data through a standardized API reduces the data owner's ability to serve advertisements. Instead, AnyLog en-

10th Annual Conference on Innovative Data Systems Research (CIDR '20)
January 12-15, 2020, Amsterdam, Netherlands.

ables data contributors to monetize data through a SQL interface that can answer queries from any source succinctly and directly.[1] Making this data public, the second goal, increases the potential for monetization.

The third goal is a critical one for structured data: the data integration problem is best approached at the source—at the time that the data is generated rather than at query time [16]. AnyLog aims to incentivize data integration prior to data publication by allowing free market forces to generate consensus on a small number of economically viable schemas per application domain (similar to how market forces have resulted in SQL being the dominant interface to most economically viable database systems). Of course, this incentivization does not completely solve the data integration problem, but we expect AnyLog to be useful for numerous application domains even when large amounts of potentially relevant data must be ignored at query time due to data integration challenges.

As a fully decentralized system, anybody can create an interface to the data in AnyLog. We envision a typical interface would look like the following: users are presented with a faceted interface that helps them to choose from a limited number of application domains. Once the domain is chosen, the user is presented with another faceted interface that enables the user to construct selection predicates (to narrow the focus of the data that the user is interested in within that domain). After this is complete, one of the schemas from all of the registered schemas for that domain is selected based on which datasets published using that schema contain the most relevant data based on the user's predicates[2]. After the schema is chosen, the interface aids the user in creating a static or streaming SQL query over that schema[3]. The entire set of data in AnyLog that was published using that schema, and for which the user who issued the query has access to, is queried. The results are combined, aggregated, and returned to the user.

AnyLog's architecture incorporates third-party contractors and coordinators for storing and providing query access to data. Contractors and coordinators act as middlemen between data publishers and consumers. This aims to overcome existing limitations of IoT systems that rely on the owner or publisher to provide the resources for storage and processing. This also facilitates managing IoT data at the edge[4].

Despite making the system easier to use for publishers, the existence of contractors and coordinators in the architecture present two challenges: (1) How to incentivize them to participate, and (2) How to preserve the integrity of data and query results when untrusted and potentially malicious entities are involved in the storage and processing. AnyLog proposes an infrastructure to solve both these challenges.

Contractors and coordinators are incentivized similarly to publishers, by a financial reward for every query they serve. Querying the platform requires a small payment of tokens. These payment tokens are shared between the publishers that contributed data that was returned by the query, along with the contractors and coordinators that were involved in processing that query.

The financial reward received per query incentivizes participation of contractors and coordinators in query processing. However, it does not ensure that the participation is honest and correct query results are returned. In fact, without safeguards, contractors and coordinators can make more money by avoiding wasting local resources on query processing, and instead returning half-baked answers to query requests.

Indeed, one of the main obstacles to building decentralized data management systems like AnyLog is how to secure the confidentiality, integrity, and availability of data, query results, and payment/incentive processing when the participants in the system are mutually distrustful and no universally-trusted third party is likely to exist. Until relatively recently, the security mechanisms necessary for building such a system did not exist, were too inefficient, or unable to scale. Today, we believe recent advances in secure query processing, blockchain, byzantine agreement, and trusted execution environments put secure decentralized database systems within reach. AnyLog's infrastructure uses a combination of these mechanisms to secure data and computation within the system.

In the rest of this paper, we present the architecture of AnyLog. Section 2 overviews the system model and security components. Then, we present AnyLog's decentralized compensation scheme in Section 3. Details about normal-case operation in AnyLog and security risks and challenges are presented in Sections 4 and 5. We provide a discussion of other use cases for AnyLog in Section 6. Related work is presented in Section 7 followed by a conclusion in Section 8.

2. ARCHITECTURAL OVERVIEW

2.1 System Model and Components

Figure 1 shows five types of members of an AnyLog network:

- **Clients**, which produce static or continuous SQL queries, and consume the results.
- **Coordinator** servers that receive SQL queries, and parse, plan, optimize, and coordinate their parallel execution across potentially many contractors.
- **Contractor** servers that store all data at rest in the AnyLog network, and provide a query interface to access locally stored data.
- **Publisher** nodes (e.g., IoT devices or hubs) that produce data and ship it to contractors for storage and queryable access.
- **Blockchain** infrastructure that maintains the configuration and bookkeeping information of the global state of AnyLog and arbitrates disputes between participants.

AnyLog's architecture is fully decentralized—there are no restrictions regarding who can join as any member type. There are no inherent assumptions that any network member is trustworthy or is optimizing for the best interest of the network as a whole. However, members may take advantage of established trust relationships to reduce the overhead of security protocols and increase performance.

A client works with its preferred coordinator (similar to how we choose our preferred search engine on the WWW today) to generate queries over data in AnyLog. The coordinator parses, optimizes and generates a query plan for the query. The query is then performed in parallel (using standard parallel query processing techniques) across all of the involved contractors. Each contractor receives a micropayment in tokens from the coordinator in return for its effort during query processing. The coordinator then aggregates

[1] Note that this interface does not preclude the data from ultimately being monetized by advertisements. A website may serve advertisements to subsidize the cost of querying data stored in AnyLog, which, as described below, will be passed on to the publishers.

[2] Alternatively, a user can specify a desired schema from the start.

[3] Advanced users and machines would skip all of the previous steps and issue static or continuous SQL queries over AnyLog directly.

[4] Gartner currently predicts that 75% of IoT data will be managed at the edge by 2025 [33]

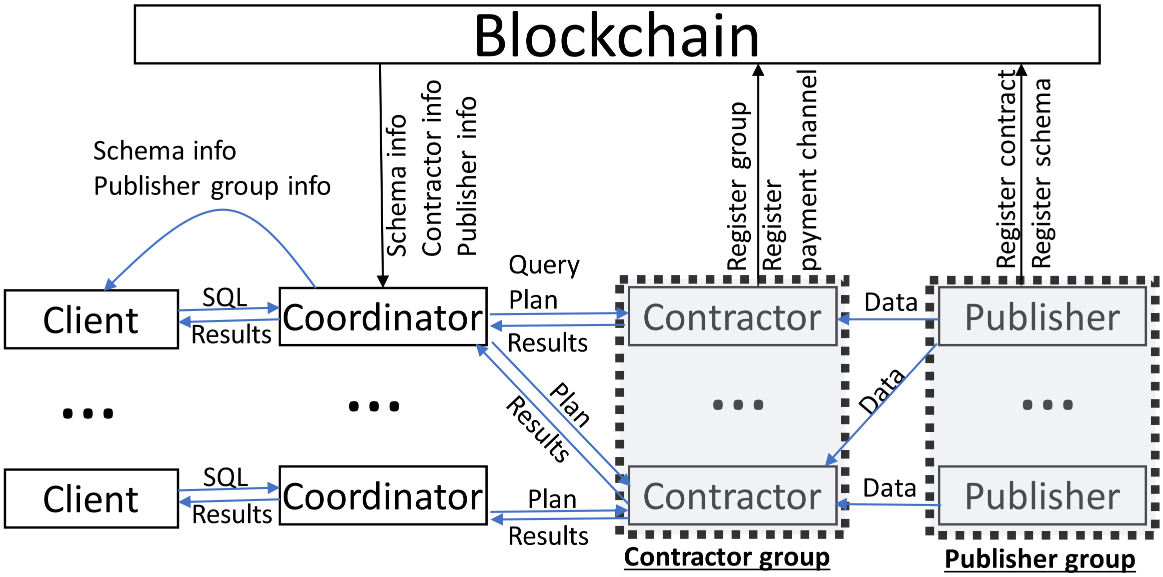

Figure 1: AnyLog Architecture

results and sends them to the client in return for a payment in tokens from the client.

A **Blockchain** serves as a global log of configuration and meta information. Through the blockchain component, AnyLog participants can register their existence along with verifiable promises of their activities (such as schemas they use, service levels they guarantee, or contract terms with other participants). These registered promises are used as input to automated arbitration of disputes (Section 3.5). The blockchain is also used for registering payment channels so that different entities on AnyLog can exchange token micropayments quickly, without requiring reading or writing to the chain during the micropayment (Section 3.4). In general, writing to the blockchain is restricted to infrequent events that configure the state of the AnyLog network. However, frequent collection, storage, and querying of data is performed off-chain. This helps limit the overhead incurred due to writing to the blockchain log.

Publishers produce data in batches. Prior to publishing data, publishers must register a contract with one or more contractors. This contract establishes that these contractors will be responsible for maintaining and serving the publisher's data. The contract may also include storage or processing requirements, such as the level of fault-tolerance and SLA guarantees. After a contract is registered, the publisher can send data to the contractor(s) in the contract.

Contractors maintain and serve data that is generated by publishers. After agreeing to a contract with a publisher, the contractor receives data from publishers and stores and provides query access to this data. Contractors provide access to the data through standard SQL queries, which can be used by coordinators as part of client query processing.

Contractor and Publisher Groups. In AnyLog, many contractors can form a contractor group and many publishers can form a publisher group. This is useful to enable multiple contractors to collectively satisfy more stringent performance and fault-tolerance guarantees than would be feasible for singleton contractors. Furthermore, this enables publishers that belong to the same organization to act as a single unit and reduces load on the blockchain by allowing contracts to be registered at the group level. For uniformity, we assume that any contractor is part of a contractor group and that any publisher is part of a publisher group. A contractor or publisher group might consist of a single node.

Groups can be centralized or decentralized. Centralized groups are managed by a group manager node (typically the creator of the group). The manager issues signed group credentials to each member; other nodes authenticate membership by verifying these credentials. To remove a node, the manager adds the node's credentials to a revocation list it maintains for the group. The manager also acts as the interface between the group and the rest of the AnyLog network, registering contracts and processing payments on behalf of the members.

Decentralized groups are managed by a smart contract in the blockchain (see Section 2.2 for some background on how smart contracts work). AnyLog nodes invoke the contract's functionality to add nodes, enter into contracts with publisher groups, and distribute payments. For some operations, the contract may require a threshold of group members to authorize the action. Each group management smart contract maintains a list of the current group membership and AnyLog nodes authenticate membership using this list. Instead of a revocation list (as used by centralized groups), a member is removed from a decentralized group when the contract removes it from the group list.

Coordinators are the access points to the AnyLog network. Coordinators continuously monitor registrations in the blockchain, in addition to tracking other metadata stored by the contractor groups across AnyLog, in order to be aware of all contractor groups, publisher groups, and schemas that exist, along with the properties of the data which is available across the network. The coordinator uses this information to aid clients generate queries, determine which contractors contain potentially relevant data for a given query, and plan and optimize queries across these contractors.

Clients are users and machines interested in querying the AnyLog network by connecting to coordinators or contractors through high-level user interfaces, APIs, or query languages. This is similar to how accessing the web can take various forms, such as access through web browsers for users and APIs for automated scripts. Like the web, access to AnyLog can start with simple, ad-hoc interfaces that evolve to widely-adopted standards.

2.2 Security Components

AnyLog is designed to be a general decentralized system, providing support both for cases where publishers, contractors, and coordinators are mutually distrustful, and also for cases where there are existing trust relationships amongst these entities. AnyLog leverages several security mechanisms to enforce the confidentiality, integrity, and availability of the data it stores and processes, which are primarily applied in situations of mutual distrust. In this section we review the basic security properties of these mechanisms.

Public-key cryptography All entities in AnyLog—publishers, contractors, and coordinators—are identified by a public key. A public/private key pair may be used to sign and authenticate messages between AnyLog nodes. By exchanging signed messages containing public encryption keys, AnyLog nodes may also establish secure, authenticated channels.

Query Result Integrity AnyLog supports enforcing the integrity of query results of untrusted nodes through a combination of trusted hardware and authenticated data structures. Platforms such as Intel SGX or ARM TrustZone enable a Trusted Execution Environment (TEE). A TEE prevents the host from observing or manipulating the memory or code of programs running in the secure enclave of the TEE. Additionally, the TEE provides secure measurement operations that enable *remote attestation* protocols that help an untrusted host prove to a remote entity that a particular program is running in the TEE. This enforces the *confidentiality* and *integrity* of computation in the TEE (though it does not enforce availability.) Publishers who do not trust the security of a particular TEE technology (or whose anticipated workloads may not scale with the TEE's available resources) may choose to utilize Authenticated Data Structures (ADSs) [12, 23]. ADSs provide another approach for authenticating query results that enables remote attestation without special hardware. Merkle trees [21] are an example of an ADS.

Data Confidentiality. We expect the vast majority of AnyLog publishers will either publish public data, or will have existing trust relationships with at least one contractor group that they believe is sufficiently trustworthy to protect the confidentiality of the publisher's data. Nonetheless, AnyLog publishers that wish to protect the confidentiality of their data from contractors will publish encrypted data to contractors. Depending on the desired workload and security guarantees, these contractors will either answer queries using encrypted query-processing techniques similar to CryptDB [28], or process queries within a TEE.

To release query results to clients without revealing the keys of the publisher's entire dataset, these publishers also deploy *re-encryption enclaves* that reveal results to clients (or coordinators) based on authorization tokens. A re-encryption enclave may be deployed in a number of ways: on contractors, coordinators, or clients. The enclaves receive an encrypted result set and an authorization token as input and then output a result set that is re-encrypted to the recipient specified by the token. Using an enclave to re-encrypt result sets is a flexible way to enforce the publisher's access control policies without imposing additional hardware or trust requirements on contractors, whose resources are most in demand. Queries processed within TEEs could potentially encrypt results directly to the intended recipient, but doing so would require additional key management mechanisms.

Smart Contracts and State Channels. A *smart contract* is simply code that "runs on a blockchain." Specifically, blockchain protocols such as Ethereum and (to a lesser extent) Bitcoin, support transactions that publish programs to the blockchain, as well as transactions that invoke these programs and send data or funds to them for processing. Because the semantics of the program is defined by the blockchain protocol, all miners in the blockchain network are incentivized to execute the code honestly or risk their blocks being rejected by the network (and thus forfeiting their mining reward).

There are several drawbacks to using smart contracts in a distributed system protocol. The first drawback is cost: each invocation of the smart contract involves a blockchain transaction that include a *transaction fee* to incentivize miners to include the transaction in their next mined block. The second drawback is latency: because smart contracts are invoked via transactions, the system must wait on a new block to arrive to receive the results of a call. The third drawback is throughput: blockchain protocols typically place an upper limit on block sizes, limiting the number of transactions that can be included in each block.

AnyLog avoids these drawbacks whenever possible by using "off-chain" protocols like *state-channels* (*e.g.*, [22]) to avoid the cost, latency, and throughput limitations of blockchain protocols. At a high level, most AnyLog protocols can be conducted via direct peer-to-peer interactions between nodes. By signing and authenticating messages, nodes maintain evidence of the agreed-upon state of a protocol. As long as nodes continue to interact in accordance with the AnyLog protocol, no interaction with the blockchain is necessary. If a node fails to send an expected response, or sends an invalid response, only then will a blockchain transaction be needed to arbitrate between the participants. We discuss in more depth how smart contracts and state channels are used in AnyLog in Section 3.5 and Section 5.

3. DECENTRALIZED COMPENSATION

A key contribution of AnyLog is the development of decentralized compensation. We believe that a misaligned incentive model was an important impediment of prior peer-to-peer structured data sharing systems. AnyLog's decentralized compensation enables an incentive model that overcomes previous shortcomings. While using blockchain transactions to compensate nodes for their resources plays a role in this scheme, a primary concern is ensuring those transactions are made correctly and in a timely manner without introducing performance bottlenecks. In this section, we provide an overview of how incentives are assigned and propagated across the AnyLog network (Sections 3.1 and 3.2). Then, we present the main three design components of decentralized compensation: metadata maintenance including registration and contracts (Section 3.3), compensation processing (Section 3.4) and handling disputes (Section 3.5).

3.1 Overview

In the AnyLog network, a node is compensated for each of the four following activities:

1. **Data publishing:** a publisher is compensated for the data it publishes every time it is accessed at a contractor. The source of the compensation is either the client (if the data access is direct from the client to the contractor) or the coordinator (if the data access is performed through a coordinator). In both cases, the payment arrives at the publisher through the contractor.

2. **Data storage:** a contractor may be compensated by a publisher for storing and maintaining the publisher's data. The amount (which may be nothing) and frequency of compensation is defined in the contract between the publisher and the contractor,

and occurs after the contractor provides a proof of retrievability [18, 30, 3].

3. **Data processing at the contractor:** a contractor is compensated for performing any processing on the data it maintains. The amount of compensation depends on the type of processing and is agreed upon in the contract between the publisher and the contractor. The source of the compensation is the node that makes the request to the contractor.

4. **Data processing at the coordinator:** a coordinator is compensated for the distributed query processing and data served to the client. Since coordinators are relatively independent, the compensation model is flexible and is advertised by the coordinator to clients. The source of the compensation is the clients issuing the request or query to the coordinator.

Registrations and contracts (Section 3.3). The rules and metadata of compensations are registered in the blockchain in the form of registrations or publishing contracts. A registration serves as an announcement of a node. For example, a publisher group can register itself in blockchain to announce its interest in publishing data with specific properties and requirements. Likewise, a contractor group can register itself to announce its existence and properties. These registrations are optional and serve as a way to help publishers and contractors find each other. A contractor can register itself more than once to advertise different levels of guarantees and properties. Likewise, a publisher can register more than one schema if it generates different types of data and does not want to utilize existing schemas.

Once a publisher and contractor have found each other and have agreed to the terms of a contract, they register the data publishing contract on the blockchain. Data publishing contracts serve as an announcement to clients and coordinators to inform them of the availability of data and the way to access it.

Off-chain compensation (Section 3.4). Rather than relying on the blockchain for every payment, AnyLog uses verifiable compensation schemes that enable most fund transfers to occur off-chain, while still leveraging the blockchain to ensure that payments occur correctly.. This approach helps avoid the prohibitive performance and monetary costs of doing payments directly on the blockchain.

Disputes management (Section 3.5). The decentralization of compensation and processing can lead to cases where malicious nodes fabricate information or avoid paying compensations. AnyLog proposes the use of an automated dispute management protocol that utilizes smart contracts on the blockchain as arbiters of disputes.

3.2 Example

The life-cycle of data from publishing to being served to clients is illustrated in Figure 2. First, a publisher group P (optionally) registers itself in the blockchain. This registration contains a reference to the schema of the data it plans to publish[5]. A contractor group C also (optionally) registers on the blockchain along with its SLA guarantees and other relevant information that enables publishers to select from existing contractors (step 1). Then, P and C find each other and request establishing a data publishing contract (step 2). Both parties asynchronously agree on the contents of the contract and one of them uploads the contract—signed by both—to the blockchain (step 3) In the meantime, P and C set up a payment channel and register it in the blockchain. After the contract is uploaded, data and compensation are continuously sent between P and C (step 4)

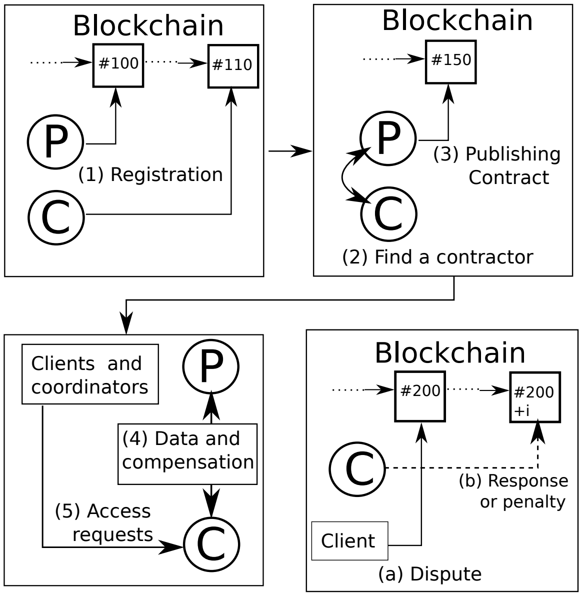

Figure 2: Compensation and dispute example. Node P denotes a Publisher and node C denotes a contractor. (P and C in all four figures represent the same two nodes across different times.)

Once the contract is published in the blockchain, coordinators and clients can request access to the published data by contacting the contractor (step 5) A client or coordinator accesses data by sending the request and compensation to C. C sends the publisher's share of the compensation to P through the payment channel and sends back the response to the client.

As an example, a dispute can arise if C does not respond to an access request from a client (or coordinator). The correct behavior for C is to either process the request and send a response or to reject the request and refund the client's payment. To force C to provide the response or a refund, the client sends a dispute record containing its request and C's node ID to the smart contract on the blockchain (step a). C must respond to this dispute within a predetermined interval (measured in the number of blocks written to the blockchain after the block that contains the dispute). C's response is either a proof of its response or proof it has satisfied its SLA and therefore may reject the request. If C fails to respond, the smart contract—using C's escrowed funds— refunds the client's payment and transaction cost to send the dispute record to blockchain (step b). Depending on the terms of the SLA, the contract may take additional punitive steps such as fines or termination of the contract.

In the rest of this section, we present the details of the three main design components for decentralized compensation.

3.3 Publishing Contracts

The process of writing a publishing contract involves coordination between a publisher group and contractor group (we refer to them in shorthand as publisher and contractor).

Before the coordination to establish a publishing contract starts, certain metadata that will be referenced by the contract needs to be previously registered in the blockchain:

(1) Schema registration. Schema registrations can be initiated by any entity in AnyLog, but will be most commonly initiated

[5] If it does not plan to use a previously registered schema, it needs to register a new schema first prior to referring to it.

from a publisher. Schemas are treated like regular data in many respects: whoever creates a schema is considered a "publisher" of that schema, and must contract with one or more contractor groups for storage and query access to the metadata related to this schema. The cost to register a schema is thus the transaction cost to register it on the blockchain, plus the contract cost with the contractor group that will store the metadata about this schema.

The metadata for a schema (version) includes the following:

- The globally unique **identifier** for that schema
- A plain text high-level **description** of the schema
- **A backwards reference** to the identifier of the previous version of this schema (if it exists)
- **The DDL commands** (using standard SQL:2003) that define the schema. The best schemas will make heavy use of integrity constraints so that others that want to use the schema have a clear understanding of the important semantics of the data published using this schema.
- **An additional set of rules (in plain text)** that describe the semantics of the data that are hard to express using integrity constraints (e.g., that temperature readings should use Celsius).
- **A list of all publisher group identifiers that have been certified** as making correct use of this schema version.
- The **contact information** and/or link to a public forum where questions, concerns, and ideas about this schema can be discussed.

(2) Contractor registration and contract document registration. Contractor registrations include the following information: (1) the globally unique identifier of the group, (2) the IP address/port information of at least one server in the group (if this changes, a new transaction must be written to the blockchain). Separately, contractors can register one or more contract documents which specify the SLA and other contract terms that it plans to support. All other metadata regarding the group is stored by the group itself. This off-chain metadata includes: (1) information about the current members of the group, (2) information about the publisher groups that it is currently contracting with, and (3) all information typically stored in a catalog on a traditional database system that describe the data being stored and important statistics about the data that are leveraged during query planning and optimization. Because this metadata is likely to change more frequently than the on-chain metadata, storing these records off-chain reduces the performance overhead for modifications. Nevertheless, decentralized contractor groups may choose to store some or all of this metadata in their smart contract to protect its integrity and ensure consensus among their members.

Publishing Contract Protocol. Publishers search through the contract document options that were previously published on the blockchain (see Section 4.1 for a discussion of how data in the blockchain is accessed) or alternatively negotiate a new contract document offline with a specific contractor. Once a publisher has found a contractor with contract terms that it finds acceptable, the contract between them is made official by writing a smart contract to the blockchain.

The contents of the publishing contract includes the following: (1) the globally unique identifier of the publisher group that is registering the contract, (2) the globally unique identifier of the contractor group that it is contracting with and its signature agreeing to the contents of the contract, (3) the period of the contract, (4) the globally unique identifier of the contract document that was previously published in the blockchain that is being agreed to by the parties involved in this contract (5) any changes or revisions to this referenced contract document that is specific to this particular agreement (5) (if required by the contract terms) a deposit of funds by the publisher to be placed in escrow to ensure payment for services supplied by the contractor such as storage of the data.

As part of the contract, the contractor group becomes responsible for storing and providing query access to all metadata about the publisher group it is contracting with, including (1) the members of the publisher group, (2) information regarding the rules that all members of the publisher group must adhere to (e.g., all members are associated with a particular device version from a particular manufacturer), and (3) the globally unique identifier of the schema version (that was previously registered in blockchain) associated with data published by members of this group.

3.4 Compensation Processing

In AnyLog, the payments between nodes can potentially occur frequently, and in small amounts. For example, the payment from a coordinator to a contractor for its data processing contribution to a particular query may be on the order of a few cents, or even less. Transaction costs on popular blockchain implementations make processing each micropayment via a blockchain prohibitive. The problem of performing micropayments on blockchains is well-known, and several practical solutions have been proposed.

The most popular of these solutions is to perform micropayments via payment channels [27] between entities. Two entities that wish to exchange micropayments deposit funds to a smart contract that holds these funds in escrow. The two entities may then directly exchange signed messages "off-chain" that add or subtract from the balance of their escrowed funds. To close the channel, either entity may submit the log of signed transactions to the smart contract to settle the off-chain transactions against the on-chain balances and withdraw their remaining funds.

State channels [22] generalize the idea of payment channels to more general protocols. For example, data storage compensation for contractors is governed by a state-channel protocol. Publishers periodically challenge contractors to provide proofs of retrievability [18, 30, 3] (PORs). If the proof is valid, the publisher sends payment for the storage period. If the contractor fails to respond or to provide a valid POR, the publisher may attempt to cancel the contract (possibly triggering additional penalties) by sending a dispute record to the smart contract. If the contractor provides a valid POR to the contract, it receives payment for the storage. If the publisher fails to pay the contractor, the contractor may initiate a dispute to force payments or cancel the contract.

Two common challenges for off-chain protocols are *routing* and *collateral management*. Because payment channels require an on-chain deposit of funds, routing payments (for a small fee) between entities with pre-existing channels saves time and lowers the collateral needed to usefully interact with the system. Furthermore, if the transfers that occur off-chain deplete the deposits stored on-chain, additional funds must be deposited so that off-chain payments are properly collateralized. Until these deposits are finalized on-chain, off-chain payments via the channel cannot be accepted.

Fortunately, AnyLog's architecture lends itself to relatively simple solutions to these challenges. First, payments between publishers and contractors only occur after a contract has been registered on-chain, so establishing direct payment channels (typically as part of the publishing contract's functionality) avoids the need to route these payments via other nodes. Payments from publisher to contractors for storage are regular, and thus easy to anticipate. Contractors make payments to publishers only after receiving payments from coordinators or clients, and publishers can detect if contractors miss a payment by auditing periodic summaries of processed requests (Section 5.2).

Some coordinators will specialize in particular datasets to add value for clients through indexing and other aggregation services. Thus these coordinators may only need to establish direct payment channels with a limited number of contractors. More general-purpose coordinators may choose to route payments through these specialized coordinators rather than establish direct channels.

Clients will typically only need to establish payment channels with the coordinators they choose. Since the coordinator will likely have direct channels to contractors or indirect channels via other coordinators, any payments made by the client can be routed through the coordinator. A failure to route a payment can be resolved by the sender raising a dispute with the payment channel contract.

Coordinators will periodically need to transfer off-chain balances from client channels to contractor channels, but except during periods of extreme volatility in the workload, the need for such transfers should be easy to anticipate so as to not interrupt service.

3.5 Disputes

Many interactions in AnyLog are decentralized, happening directly between nodes. This introduces the potential of problems caused by malicious or unresponsive nodes. Violations to SLA or AnyLog protocols are handled in AnyLog via a decentralized dispute mechanism. This dispute mechanism is implemented via smart contracts that are run in the blockchain. The dispute smart contract has the capability of performing actions such as charging a fine to the offending party. The fine and penalties are paid from escrow funds that are created at the time when the relationship between nodes are established.

There are many different types of disputes that could occur, including disputes over unanswered queries from contractors, disputes over missing compensation, and disputes over incorrect query results. All these disputes follow the same pattern: a response or compensation is expected, but is not received or is incorrect. The receiving party sends a transaction to the SLA smart contract to initiate the dispute, and includes any evidence relevant to its dispute claim (e.g., a signed acknowledgment by the contractor accepting a query). If the dispute can be (mechanically) adjudicated on the basis of this evidence alone, then the smart contract can verify the dispute claim and transfer funds or cancel the contract as specified by the SLA to resolve the dispute.

Many disputes, however, regard claims about the *absence* of a message. These disputes cannot be resolved immediately since the receiver cannot prove their claim directly: a malicious receiver could attempt to negatively impact another party by falsely claiming it did not receive a message. To avoid abuse in the processing of these claims, the accused party is given an opportunity to present the expected message directly to the smart contract (as opposed to the intended party). If the message is received within a specified interval after the dispute, then no negative consequences are assessed to the accused node. In Section 5 we provide further details on how how disputes are resolved and the role disputes play in enforcing the integrity and availability of AnyLog data and computation.

4. NORMAL-CASE OPERATION

In this section, we present the protocols that AnyLog nodes follow to perform various tasks such as accessing metadata, data publication, storage, and query processing.

4.1 Accessing important metadata

All participants in an AnyLog network require access to metadata during normal operation. For example, coordinators require metadata about contractor groups in order to know where to find relevant data for a particular query, metadata about publishers and schemas in order to enable clients to express queries over data most relevant to them, and metadata about the data being queried in order to properly perform authentication, optimization, and planning.

As we described earlier, some of this metadata is present in the blockchain (e.g. IP addresses of contractor groups), and the rest is maintained by contractor groups as required by contract terms (e.g. schema and publisher group metadata). The data in blockchain needs to be converted to a format that is easy and fast to query since it may be accessed frequently. In theory, each AnyLog participant (such as a coordinator or a publisher) is capable of iterating through the history of the blockchain, extracting all changes to the metadata, and inserting this into an indexed relational database table for future use. Clearly, this is a lot of work, and it would be wasteful if each participant did this redundant work independently. AnyLog's incentive structure will likely make this redundancy unnecessary. Any AnyLog participant may publish its version of the metadata it has extracted from the blockchain in an manner no different than how any other data is published in AnyLog. Other participants can choose to query this data instead of maintaining their own tables if it is cost efficient to do so.

4.2 Data Publication

Publishers are devices that contribute data to the network in batches. Any machine on the Internet can become a publisher. Each publisher is associated with a globally unique ID, a public key, and digitally signs all data that it contributes with that ID. For IoT devices that are not provisioned with a public key or whose computational resources would be inadequate for digital signatures, a trusted IoT hub or host can aggregate device data over the local network and publish on the device's behalf.

All published data is annotated with the publisher group of the data, and client queries may include selection predicates that constrain the publishers of the data returned by a query. Publisher groups that are carefully managed and curated will generally produce more reliable data. Therefore, publishers will be motivated to join an existing group, if possible. The members of a publisher group are not required to have a uniform security policy.

All members in the same publisher group must produce data according to the same schema (version). A well managed publisher group will be careful to ensure that there are no semantic differences in the way data is produced across members of the group. The group manager of a publisher group (either a node or a smart contract), is responsible for enforcing group policies. For example, the manager may periodically query data produced by publishers to verify compliance, or provide incentives for contractors to report malformed data submitted by a publisher. The manager of a publisher group can also (optionally) specify other conditions that all members of the group must meet, such as frequency of data production, physical location of the publisher, and hardware/software running on the device. Enforcement of some conditions may require publishers to trust the group manager to act fairly if evidence of compliance is infeasible to collect or verify mechanically. Decentralized publisher groups managed by a smart contract may thus be limited in the conditions they can effectively enforce.

The manager of a publisher group controls the metadata for that group, and is empowered with choosing which contractor group that the publisher group contracts with. The contract terms detail the profit sharing agreement (when the published data is queried) between the contractor group and publisher group, and may also require the contractor group to keep track of which individual publishers produced the data that was queried, so that the income for a query can be divided across the publishers within a group fairly.

Some publication groups may be set up such that individual publishers can leave a publisher group at any time. The status of the previous data that this individual publisher published before leaving the publisher group is dependent on the group rules. In some cases, this data remains associated with the old group, and in some cases the old group loses the rights to this data. In the latter case, contract terms with contractor groups must be set up to enable the removal of data produced by independent publishers that leave a publisher group.

By default, schemas are public and open. Any publisher can publish using any existing public schema. Certain schemas will start to dominate an application domain, and new publishers will be motivated to publish using the dominant schema (to increase the probability that the data they publish will be included in a query result and generate income). Unfortunately, it is inevitable that some publishers will violate the semantic rules specified by the schema manager in the plain text part of the schema definition (either by mistake or on purpose). To counteract this, AnyLog provides an ability for schema managers to "certify" publisher groups, and this information is included in the metadata of the schema (see Section 3.3). In certifying a publisher group, the schema manager states that it believes that this group is abiding by all of the semantics and rules specified by the schema metadata. The choice of whether or not to certify any groups, and what is required in order to achieve certification is left to the discretion of the schema manager. AnyLog includes a list of certified publisher groups as a first class citizen of schema metadata in order to make it easy for clients to express queries over only certified publisher groups for a schema if they chose to do so.

4.3 Storage and Query Processing

Published data is maintained and stored in contractor nodes. The contractors and coordinators of the network are responsible for query processing. In this section, we discuss the storage and query processing tasks associated with these nodes.

4.3.1 Contractors

Contractors are servers that store and process data in AnyLog. Any machine on the Internet can register under a contractor group on AnyLog as a contractor. It is the responsibility of the contractor group, and specifically the group manager, to ensure its members uphold the SLAs of any contract entered into by the group.

A contractor group may choose to present a unified interface via its group manager (or other designated nodes) to interact with the rest of the AnyLog system. This single interface gives the group more flexibility to adaptively partition data and distribute queries internally to optimize changing workloads. The flexibility comes at a price however, since violations of SLA terms may be levied on the group as a whole rather than individual contractors. In these cases the contractor group is essentially treated as a single (but distributed) contractor by other nodes. Alternatively, data assignments and query distribution may be visible the rest of the AnyLog network allowing publishers and coordinators to determine which specific contractors are responsible for storing and processing data. In these cases, the individual contractors may be held accountable for violating SLA terms.

Contractor groups enter into publishing contracts with publisher groups (Section 3.3). These contracts typically obligate the contractor group to provide *integrity guarantees*, such as the authenticity of stored data and query results, *availability guarantees*, such as response time and access to stored data, and *confidentiality guarantees* of the stored data. The enforcement of these guarantees are discussed in depth in Section 5.

4.3.2 Coordinators

Coordinators are servers (or groups of servers) that manage the processing of queries in AnyLog. Upon receipt of a SQL query, coordinators perform all of the standard tasks that occur prior to query execution in a traditional parallel database system: query parsing, query rewrite and optimization, and plan generation.

The contractors enforce whether the coordinator is permitted to receive the results of a query (before executing it), or the result set is encrypted and the coordinator (or the client) must obtain the decryption key from the publisher group to view it. When coordinators wish to perform tasks on encrypted data such as building indexes or aggregating results across multiple contractors, they can use similar approaches to those used by contractors, such as trusted execution environments.

The query optimization problem is more challenging for AnyLog coordinators than in traditional parallel database systems. The data relevant for any particular query may be stored across many different contractors, that each have substantially different levels of quality of service and query processing prices. Care must be taken to avoid long tail latencies where queries get bottlenecked waiting for slow contractors running at low levels of QoS. Furthermore, some data stored in low levels of QoS may be totally offline and not possible to include in a query result. Since each contractor charges the coordinator for access to its data, it is particularly important to avoid getting contractors involved in a query unnecessarily. Finally, coordinators may have trust relationships that vary from contractor to contractor which may require coordinators to authenticate some results based on their source.

We anticipate a competitive marketplace of coordinators—each one with their own indexing and caching layers to improve performance of queries and reduce costs to the client. Coordinators have total freedom to decide how many tokens to charge a client to process a query (we expect that the most successful coordinators will provide accurate estimates prior to query processing and keep costs as low as possible).

5. SECURITY RISKS AND CHALLENGES

In Section 2.2 we gave an overview of properties of the security mechanisms that enable query processing in distrustful environments. In this section, we discuss how these mechanisms are applied in AnyLog. We are primarily concerned with the enforcement of *decentralized* SLA contracts where publishers, contractors, and coordinators are mutually distrustful. The enforcement mechanisms we discuss may also be useful when a trusted mediator is available, but such contracts can define specialized versions of our mechanisms that leverage trust relationships to reduce overhead.

5.1 Query result integrity

Contractors that are not trusted by the publishers and coordinators they interact with can still participate in the AnyLog network by using protocols that enable coordinators to verify query results over data assigned to that contractor. This verification can be done through a combination of trusted hardware and authenticated data structures. The SLA advertised by a contractor specifies which specific enforcement mechanisms will be employed for the contract. For example, untrusted contractor groups without TEEs available may still offer ADS-based contracts. Publishers who do not trust the security of a particular TEE technology (or whose anticipated workloads may not scale with the TEE's available resources) may choose to only establish contracts with contractors that implement enforcement mechanisms that satisfy their requirements.

All data shipped to a contractor is signed by the publisher to establish its integrity and authenticity. If the contractor enforces integrity of query results using TEEs, the publisher additionally verifies the authenticity of each contractor's TEE and the query processing program using remote attestation. If the TEE and program are verified, then the publisher issues a certificate that the contractor provides to the coordinator to authenticate all query results from this contractor. Since only the TEE is capable of producing results that are signed by TEE's key, the coordinator knows that the program running in the TEE is trusted by the publisher of the data.

For ADS-based enforcement, the publisher issues a signed message containing a root digest of the current data stored by the contractor and a version number. When the next batch of new data is published, the publisher updates this root digest and increments the version number. These messages may be distributed directly to coordinators and contractors or published on the blockchain. New data cannot be included in query results until the hash is updated. With a query result, contractors can provide an ADS proof based on the current root digest. Using the publisher's signed digest, coordinators verify the integrity of the results.

If a coordinator receives invalid results (either because of an invalid signature from a TEE or an invalid ADS proof), coordinators can dispute the result by sending the offending results to the SLA contract. If the contract determines that the result is in fact invalid, the contract performs actions specified by the SLA contract such as charging a fine to the contractor.

5.1.1 Optimizations

For large query results that would be infeasible or expensive to send to the blockchain, the SLA contract may specify an entity (or group of entities) trusted by the coordinator and the contractors to validate results. For example, a host running validation code within an SGX enclave could be designated as the mediator for an SLA contract. If the coordinator wishes to dispute a query result, it sends the results to the enclave, which could even be running locally on the coordinator's host. For TEE-enforced SLAs, the enclave will attempt to verify the signature on the results. For ADS-based SLAs, the enclave will attempt to verify the ADS proof. The enclave outputs a (small) signed validation result, which the coordinator can then send to the SLA contract.

Generating ADS proofs may add significant overhead to the contractors' workload. Fortunately, the AnyLog setting offers some opportunities for amortizing these costs. For example, an SLA contract may allow contractors to initially respond to queries without generating a proof. To disincentivize the contractor from processing queries dishonestly, coordinators are occasionally permitted to request an ADS proof for one of their previously answered queries. Queries requiring proofs are chosen based on a public, unpredictable value such as a block hash[6]. Since the contractor must commit to the query results before it knows which queries it must produce proofs for, it runs a risk of getting caught proportional to the ratio of proofs to queries. By setting the penalties for invalid results appropriately, an SLA contract can ensure contractors will not benefit (in expectation) from producing invalid results.

5.2 SLA enforcement

It is inherently difficult to enforce SLAs in a decentralized environment: monitoring aspects of query processing like response time is extremely challenging when contractors and coordinators may behave maliciously. For example, a coordinator might lie about when a request was sent, and a contractor might lie about when it was received. Furthermore, mechanisms like ADSs and TEEs are capable of enforcing the integrity guarantees of an SLA, but not the availability guarantees. AnyLog uses state-channel-based protocols that leverage the SLA contract and the underlying blockchain to enforce course-grained SLA guarantees.

We first discuss how AnyLog handles disputes over unanswered queries sent from a client (or coordinator) to a contractor (we call these *availability violations*.) This discussion also applies to the other types of availability disputes. Later in this section we will discuss disputes over compensation that is not forwarded from contractors to publishers.

Contractors are incentivized to process queries if they have resources available since they only receive payment for the queries they process. They may, however, attempt to overcommit their resources to multiple contracts in order to ensure a higher utilization rate and more revenue opportunities. This strategy is permissible provided the contractor meets the terms of its SLAs. However, contract terms will usually require it to pay a penalty if it cannot handle the required load and fails to meet the SLA terms. In some cases, AnyLog participants may prefer to utilize legal channels outside of the AnyLog network to resolve SLA violations. Nonetheless, AnyLog provides a mechanism for detecting and resolving certain types of SLA violations within the network.

The availability component of decentralized SLA contracts may be specified in terms of a *duration* and minimum *record number*. Such a specification in a SLA smart contract requires a contractor to process a certain number of records during the specified duration. The contractor is not allowed to reject requests within this duration until it has reached this specified minimum. When the input load is insufficient to allow the contractor to reach its required throughput, a contractor may request a reduction in the required minimum record number by sending a message to the SLA smart contract. This message contains a time range for which the contractor wishes to claim a below-average request load. This insulates contractors from bursty input loads causing SLA violations.

To allow efficient bookkeeping of these availability metrics, contractors process queries in batches. The number of batches and their size is based on the maximum (contracted) capacity of the contractor over the *batch interval*, usually a multiple of the underlying blockchain's block arrival time. Thus, each batch number corresponds to a specific deadline (a block number) by which the requests of that batch must be processed[7].

For example, suppose the duration of a contract is roughly 48 hours and the minimum record number is 288,000,000. This duration is divided into 288 batches with each batch interval lasting for 43 blocks, or about 10 minutes on the Ethereum blockchain. The contractor is expected to process at least 1 million records per batch. Each coordinator with a request in batch n should receive the results of their query by the time the blockchain has increased by $(n + 1) * 43$ blocks.

Contractors respond to queries from coordinators with a signed acknowledgement of the request, an assigned batch number, and a digest of the previous batches. This digest forces contractors to fill

[6] It is well known that using block hashes as pseudorandom numbers is problematic if a miner might be incentivized to forgo the reward for mining a new block in order to influence the next block hash. We expect that block rewards will far exceed the cost of AnyLog queries. If not, some other source of unpredictable numbers should be used.

[7] Block numbers are simply a convenient way to objectively specify a time interval—batch assignment does not require communication with the blockchain unless a dispute occurs.

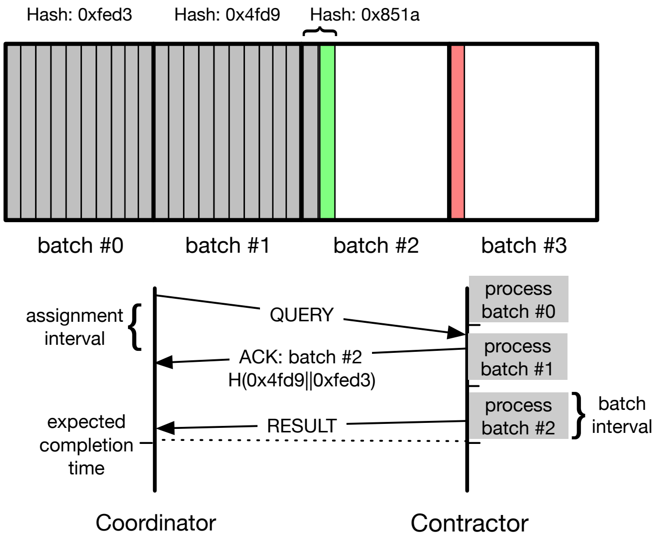

Figure 3: Query processing batches

the current batch before assigning queries to later batches, which would result in longer response times and sparser batches. If the contractor has exceeded its capacity, it may respond with a negative acknowledgement containing a digest of the contents of its batches to prove the SLA terms and thus is permitted to reject the query. For example, Figure 3 illustrates the request queue of an AnyLog contractor. Batches #0 and #1 are full, so when a new request is received, the contractor must assign it to batch #2 and send a hash of batches #0 and #1 in the acknowledgement.

If the batch interval expires and the coordinator has not received the result of the query, the coordinator presents its batch assignment to the SLA contract and the contractor must submit valid results to the contract within a specified number of blocks or receive a penalty. In addition to the batch interval, the SLA contract also specifies an *assignment interval* within which the contractor must respond with a batch assignment or a negative acknowledgement. If the coordinator does not receive a response within the assignment interval or the response is invalid, it may present its request to the smart contract to force a response from the contractor. The contract will wait a specified number of blocks for the contractor to submit a batch assignment or negative acknowledgement. If the contractor does not submit a valid response, the contract will penalize it for violating the SLA.

Contractors periodically publish *checkpoints*, the digests of their accepted batches, to the SLA smart contract. A coordinator that received a negative acknowledgment may compare the digest it received to the digests of the batches in the checkpoint. If the digest fails to match, the coordinator may dispute the negative ack to receive a penalty payment from the contractor as specified by the SLA. For example, in Figure 3, if the contractor attempted to assign the request to batch #3, it would be forced to hash the current contents of batch #2. If the contract later assigned additional requests to batch #2 (and included them in the checkpoint), the coordinator could present its acknowledgement containing the assignment to batch #3 to the contract and receive a penalty payment.

Publishers ensure they are paid for requests on their data in one of two ways. When contractors hosting a publishers data use TEEs, a summary of payments to publishers (and their transaction IDs) is included with each checkpoint. Since this summary can only be generated and signed by trusted code running inside the TEE, the publisher can trust the accuracy of this report.

If a contractor is not using a TEE, then the contractor transmits the requests associated with each checkpoint directly to publishers (or a publisher's representative) for auditing. Publishers use these logs to ensure they have received payment for the requests processed in each batch. By examining these requests, the publisher can determine whether its data participated in the results. If the publisher determines a payment was withheld by the contractor, it may submit its data and the relevant request log as evidence to the smart contract to receive payment and possibly trigger a penalty for the contractor. If the contractor fails to provide requests for audit to the publisher, or these requests do not match the on-chain hash, the publisher may also initiate a dispute to penalize the contractor or cancel the contract[8].

Note that even though decentralized SLA contracts are only capable of enforcing relatively coarse-grained service-level guarantees, we expect contractors will offer "soft guarantees" that exceed enforceable guarantees to compete with other contractors. Contractors with a good reputation for meeting these soft guarantees will be more likely to be chosen by publishers for new contracts.

5.3 Data confidentiality

We expect the vast majority of AnyLog publishers will either publish public data, or will have existing trust relationships with at least one contractor group that they believe is sufficiently trustworthy to protect the confidentiality of the publisher's data. Nonetheless, in some (minority of) cases, the publisher may not trust any contractor group, and thus choose to encrypt the data it sends to the contractor (group). Encrypting the data protects its confidentiality, but it also limits the queries the contractor is able to process on the data. One option is for publishers to deploy a TEE to each contractor that decrypts data and processes queries within the enclave. If TEEs are already used as an integrity enforcement mechanism as discussed above, using a TEE for confidentiality is a natural choice. However, TEE limitations may not be suitable for all workloads, and not all contractors may have compatible TEE hardware.

Previous work such as CryptDB [28] protects the confidentiality of data against an untrusted database operator via *encrypted query processing*. This approach creates a tradeoff between the strength of encryption and supported operations. For example, to support queries with equality constraints, deterministic encryption is used so that query constraints can be matched against database entries. Because all database entries are encrypted deterministically, the database host is able to learn which entries are equal to each other.

An advantage of the CryptDB approach is that it requires very little modification of existing database systems. Unfortunately, some aspects do not immediately generalize to the AnyLog setting. The primary issue is that CryptDB employs a proxy that manages encryption and decryption keys on behalf of clients. This proxy is problematic in the AnyLog setting for two reasons. First, AnyLog assumes a more decentralized setting than CryptDB: a proxy operator that is sufficiently trusted by publishers, coordinators, and clients may not exist. Second, publishers may not anticipate which users will want to perform queries on the data sent to the contrac-

[8] There is some risk of collusion between clients and contractors where contractors accept direct payment from clients for "off-book" transactions that are not assigned to a batch and not included in the checkpoint. Collusive behavior such as this between clients and contractors can be disincentivized by offering a reward to clients paid with the contractor's escrowed funds that demonstrate processed requests not included in a checkpoint.

tors. Therefore, releasing results to the clients becomes challenging. If the publisher simply provided access by revealing the decryption key to the client, then a client could collude with a contractor to decrypt the entire data set. Alternatives such as requiring that all results are decrypted by a trusted host publisher create potentially severe performance bottlenecks.

To avoid such bottlenecks, AnyLog publishers that wish to protect the confidentiality of their data will deploy TEEs that manage decryption keys so that results may be revealed to clients (or coordinators) based on the authorization tokens provided to the client. An enclave may be deployed in a number of ways: on contractors, coordinators, or clients. The enclaves receive an encrypted result set and an authorization token as input and output a result set that is re-encrypted to the intended recipient, specified by the token. Using TEEs only for re-encrypting result sets enforces the publisher's access control policies without imposing additional overhead or hardware requirements on contractors, whose resources are most in demand.

5.3.1 Freeloading

Clients must pay for each query they send to AnyLog. Once a client obtains access to the results of a query, there are few effective technological mechanisms for preventing or constraining further disclosure. A client may attempt to resell the data they have access to or disclose it publicly. Therefore publishers must take unauthorized disclosures into account when setting query prices or providing access to decryption keys. In some settings, relying on more traditional deterrence mechanisms such as legal contracts may be appropriate.

The value of some data may in part be based on its authenticity. Using a scheme similar to the "freeloading protection" scheme proposed by Zhang *et al.* [34], we can remove a client's ability to authenticate the data to a third party, making the data less valuable to a third party without undermining its value to honest clients.

For example, recall that publishers must distribute digests of stored data to clients for verifying the authenticity of ADS queries. Instead of the publisher signing these digests with its own private key, the publisher could instead require that the client generate a key and share it with the publisher to use for signing digests. The client can authenticate digests from the publisher since it knows only the publisher has the relevant key. A third party, however, cannot rule out that the client is acting maliciously— either the client or the publisher could have signed the digest. Under this scheme, coordinators may also share a signing key with publishers to permit coordinators and clients to independently authenticate query results without forcing the client to trust the coordinator.

Note that publishers could also delegate the task of freeloading protection to an entity they trust or even to an untrusted entity running a TEE. In either case, the publisher would send the data signed with its own key over an encrypted channel and the enclave or trusted party would verify and then resign the data with the key generated by the coordinator or client.

5.3.2 Data integrity

Malicious publishers can attempt to undermine the integrity of data in AnyLog in several ways:

- They can insert junk/useless data into AnyLog
- They can insert misleading/fraudulent data into AnyLog
- They can resell other publishers' data without permission

There is little financial incentive for the first type of maliciousness— junk/useless data will generally not be queried frequently enough to cover the contractor cost for its storage, and thus the publication of junk/useless data will typically result in a negative cash flow for their publisher. However, there may be significant financial incentive for the other two types of maliciousness.

The problem of "fake news", fraudulent information, and plagiarized articles are not new problems—these problems have plagued the Web for decades. Unfortunately, there is no silver bullet solution to these problems, and none likely to appear in the near future. Nonetheless, these problems have not prevented to tremendous utility and benefit brought by the existence of the Web. Users are aware that some Websites are more reputable and reliable than others, and are more likely to rely on information gleaned from reputable sites (though obviously this method of relying on human-perceived reputation is inherently fallible).

The analog of "reputable Websites" in AnyLog are "reputable publisher groups". Some publisher groups will be associated with large and well-known companies, groups, or individuals, and AnyLog users will be more likely to rely on information published by such groups instead of less reputable groups. We expect that selection predicates on publisher groups will be common.

In addition to reputability estimates done by an end-user, popular existing search engines such as Google and Bing also tend to rank results from reputable sites ahead of results from lesser-known sites. The analog in AnyLog are coordinators: we expect that the most popular coordinators in AnyLog will filter out data from non-reputable publisher groups by default, but will have a mechanism for communicating with the end-user what was filtered out in order to give the end-user the option of including this less reliable data in the results.

Prioritizing the reputability of publishers also makes freeloading protection schemes such as the one discussed in Section 5.3.1 more effective since publishers that are unable to prove the authenticity of their data will likely be filtered out from most search results. This reduces the ability of AnyLog participants to profit from reselling other publishers' data without permission.

6. USE CASES

This paper has focused thus far on a single use case for AnyLog: unifying datasets produced across different IoT device vendors into a global platform with a high performance SQL query interface. We now discuss several other use cases for AnyLog:

* **Cloud alternative.** Several IoT vendors currently have their devices upload their data into the cloud and build a query infrastructure over it there. AnyLog provides a storage and query infrastructure that avoids lock-in to a particular cloud vendor, and that can reduce the storage costs by enabling any machine on the Internet to participate. (In some sense, AnyLog is a unifying cloud above the individual cloud vendors—any cloud vendor can create new contractor group and register their own machines as contractors within that group.)

* **Data storage at the edge** Storing the data produced across a large network of devices producing large amounts of data can be expensive—whether in the cloud or on the network of independent contractors in AnyLog. Furthermore, this data may not be queried often enough to justify the network and organizational costs of collecting it into a central location. Instead, the data producing device itself (or a nearby device) can serve as the contractor for data produced by that device, and serves queries over that data during the rare situations where the data needs to be accessed.

* **Publication-funded hardware** Instead of paying $100 for a new smart thermostat, an IoT vendor can give away the device for free in return for the device software publishing data generated by that device on AnyLog. This data would likely be valuable to a vari-

ety of downstream applications, such as an insurance company that builds an application which alerts its customers if preemptive measures need to be taken in order to circumvent costly events (e.g. pipes freezing from thermostats set too low).

* **Easy fulfillment of data storage/sharing obligations** Some entities have legal obligations to store (and potentially make available for queries) data that they generate. Contractor groups can be designed for such entities that charge publisher groups a fixed price (per unit time and per unit of data stored) to join the group. Publishers can then be set up to publish their data to the contractor group to fulfill their legal obligations (whether the obligation is to share the data publicly or only with entities with the appropriate permissions).

7. RELATED WORK

There have been several efforts to create a version of the WWW for "structured" data, such as the Semantic Web [5] and Freebase [6]. Our approach differs substantially by (1) providing economic incentives for data to be contributed and integrated into existing schemas, (2) offering a SQL interface instead of graph-based approaches, (3) including the computational and storage infrastructure in the architectural vision.

Our vision closely relates to the vision of P2P databases such as PIER [15], PeerDB [24], and XPeer [9]. However, our architectural implementation of this vision and query interface differ significantly. First, P2P databases have a uniform architecture where nodes that choose to participate perform a similar set of tasks. In contrast, AnyLog divides responsibility of data publication, data storage/processing, and query planning across different types of nodes. This is important for IoT applications that typically have lightweight devices at the edge of the network. Second, AnyLog's architecture makes SLAs first-class citizens—it is extremely challenging to perform query optimization and achieve high performance distributed query processing over machines that do not make any QoS promises. Third, AnyLog supports structured data storage and querying, instead of PIER's key-value interface and PeerDB's IR-based keyword search approach. Finally, previous P2P databases do not provide sufficient incentives nor decentralized dispute resolution to motivate the effort involved in preparing and processing data for sharing.

Several projects attempt to extract structured data from existing WWW sources such as YAGO [32, 14], DBpedia [4], Elementary / DeepDive [25, 11], Knowledge Vault [13], WebTables [8], and commercial projects such as Google's Knowledge Graph and Microsoft's Satori. AnyLog differs from these approaches by reaching data publishers at an earlier stage in the data lifecycle and providing the original repository for data storage (along with a mechanism for publishers to profit from their data).

AnyLog targets IoT data in its initial implementation, similar to recent time series and IoT database systems such as TimeScaleDB [2] and InfluxDB [1]. These systems are complementary to AnyLog and are candidates for use as the database system distributed with the contractor code in AnyLog.

AnyLog allows publishers to map/publish their data to an existing schema rather than create a new one. This bottom-up approach to data integration is similar in vision to Orchestra [16]. However, AnyLog provides an economic incentive for publishers to perform this bottom-up integration. AnyLog also targets use cases (e.g. IoT) where there are many devices already producing data according to a uniform schema.

AnyLog's economic model of reimbursing query processing nodes for their efforts in performing query evaluation is similar to Mariposa [31]. AnyLog extends this model to include reimbursement for data sharing as well. Nodes in AnyLog are not as autonomous as in Mariposa, which allows for a higher performance implementation for intermediate data, less complexity (and more predictability) in query optimization, and better enforcement of SLAs.

AnyLog's architecture optionally incorporates trusted hardware [20, 26] in situations where parties do not trust each other. Running distributed analytics over secure enclaves has been investigated in detail in prior work [29, 7]. The use of ADSs has also been proposed to perform database functions such as transaction processing [17] and query processing [36, 35].

The Relation Cloud [10] proposed a vision of databases-as-a-service (DBaaS) that shares many of AnyLog's goals, but in a centralized cloud setting. In this model, the DBaaS provider is honest-but-curious. It is trusted to execute queries faithfully, but may not be trusted with confidential data. In contrast, contractors in AnyLog may attempt to undermine the confidentiality, integrity, or availability of data and queries in the system. Furthermore, because AnyLog is decentralized, a trusted entity may not be available to coordinate large-scale data migrations or partitioning.

Fabric [19] is a federated distributed system for building secure applications on distributed object stores. Similarly to AnyLog, Fabric is an open and decentralized system: any node can join the Fabric network, and Fabric nodes may differ on which nodes they consider trustworthy. Also Fabric programs process distributed persistent data, but since these computations operate on Java-style data structures like lists, sets, and hash tables, common query optimizations available to a SQL data base are nontrivial to apply. Furthermore, Fabric only uses cryptography to secure channel communication. AnyLog's use of TEE and ADS protocols enables more functionality between distrustful peers. Finally, Fabric provides no availability guarantees, whereas AnyLog enforces SLA contracts even in the presence of malicious nodes.

8. CONCLUSION

AnyLog is a decentralized platform for data publishing and querying that targets IoT data. AnyLog proposes an architecture that divides the responsibilities of publishing, storage, and processing across different node types. Also, it proposes a decentralized compensation scheme that motivates contributing data and using existing schemas for better integration. The decentralized compensation mechanism of AnyLog is enabled by the recent advances in security mechanisms such as secure query processing, blockchain, and trusted execution environments. We envision that AnyLog's architecture and decentralized compensation will enable a powerful infrastructure to query the wealth of data produced by IoT devices.

This paper has given a high level overview of the architecture. There remain several important research challenges that will be critical to the ultimate success of AnyLog. The challenges include improving the scalability and efficiency of query processing that occurs within trusted execution environments, improving the scalability of authenticated data structures and expanding their applicability to a wider range of query processing operations, federating query processing over many independent contractors, and achieving accurate cost estimates of query processing across many different entities that are entitled to token payments in return for their contributions to generating the query results.

9. ACKNOWLEDGMENTS

We are grateful to Joe Hellerstein, Amol Desphande, Gang Liao, Tasnim Kabir, and the anonymous reviewers for their feedback on earlier drafts of this paper. This work was sponsored by the NSF under grants IIS-1910613 and CNS-1815212.

10. REFERENCES

[1] InfluxDB. https://www.influxdata.com.

[2] TimeScaleDB. https://www.timescale.com.

[3] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 598–609. Acm, 2007.

[4] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In *Proc. of ISWC/ASWC*, 2007.

[5] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5), May 2001.

[6] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proc. SIGMOD*, 2008.

[7] S. Brenner, C. Wulf, and R. Kapitza. Running zookeeper coordination services in untrusted clouds. In *Proc. of HotDep*, 2014.

[8] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: Exploring the power of tables on the web. *Proc. VLDB Endow.*, 1(1), Aug. 2008.

[9] G. Conforti, G. Ghelli, P. Manghi, and C. Sartiani. Scalable query dissemination in xpeer. In *Proc. of IDEAS*, 2007.

[10] C. Curino, E. P. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational cloud: A database-as-a-service for the cloud. 2011.

[11] C. De Sa, A. Ratner, C. Ré, J. Shin, F. Wang, S. Wu, and C. Zhang. Deepdive: Declarative knowledge base construction. *SIGMOD Rec.*, 45(1), June 2016.

[12] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine. Authentic third-party data publication. In *Data and Application Security*, pages 101–112. Springer, 2002.

[13] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proc. of KDD*, 2014.

[14] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artif. Intell.*, 194, Jan. 2013.

[15] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier. In *Proc. of VLDB*, 2003.

[16] Z. Ives, N. Khandelwal, A. Kapur, and M. Cakir. Orchestra: Rapid, collaborative sharing of dynamic data. In *In CIDR*, 2005.

[17] R. Jain and S. Prabhakar. Trustworthy data from untrusted databases. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 529–540. IEEE, 2013.

[18] A. Juels and B. S. Kaliski, Jr. PORs: Proofs of retrievability for large files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 584–597, 2007.

[19] J. Liu, O. Arden, M. D. George, and A. C. Myers. Fabric: Building open distributed systems securely by construction. *J. Computer Security*, 25(4–5):319–321, May 2017.

[20] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, and C. Rozas. Intel&reg; software guard extensions (intel&reg; sgx) support for dynamic memory management inside an enclave. In *Proc. of HASP*, 2016.

[21] R. C. Merkle. Secure communications over insecure channels. *Commun. ACM*, 21(4):294–299, Apr. 1978.

[22] A. Miller, I. Bentov, R. Kumaresan, and P. McCorry. Sprites: Payment channels that go faster than lightning. *CoRR*, abs/1702.05812, 2017.

[23] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. *Trans. Storage*, 2(2):107–138, May 2006.

[24] W. Ng, B. Ooi, K.-L. Tan, and A. Zhou. PeerDB: a p2p-based system for distributed data sharing. In *Proc. of ICDE*, 2003.

[25] F. Niu, C. Zhang, C. Ré, and J. W. Shavlik. Elementary: Large-scale knowledge-base construction via machine learning and statistical inference. *Int. J. Semantic Web Inf. Syst.*, 8(3), 2012.

[26] E. Owusu, J. Guajardo, J. McCune, J. Newsome, A. Perrig, and A. Vasudevan. Oasis: On achieving a sanctuary for integrity and secrecy on untrusted platforms. In *Proc. of CCS*, 2013.

[27] J. Poon and T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016. http://lightning.network/lightning-network-paper.pdf.

[28] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 85–100. ACM, 2011.

[29] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. Vc3: Trustworthy data analytics in the cloud using sgx. In *Proc. of SP*, 2015.

[30] H. Shacham and B. Waters. Compact Proofs of Retrievability. In J. Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, pages 90–107, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[31] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: A wide-area distributed database system. *The VLDB Journal*, 5(1), Jan. 1996.

[32] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A large ontology from wikipedia and wordnet. *Web Semant.*, 6(3), Sept. 2008.

[33] R. van der Meulen. Edge computing promises near real-time insights and facilitates localized actions. https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders/, 2018.

[34] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi. Town Crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 270–282. ACM, 2016.

[35] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. vSQL: verifying arbitrary SQL queries over dynamic outsourced databases. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 863–880. IEEE, 2017.

[36] Y. Zhang, J. Katz, and C. Papamanthou. IntegriDB: Verifiable SQL for Outsourced Databases. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1480–1491, New York, NY, USA, 2015. ACM.