A Peer-to-Peer Alternative to Blockchain for Managing Distributed Data Transactions

Samuel R. Cauvin
Dept. of Computing Science
University of Aberdeen, U.K.
r01src15@abdn.ac.uk

Martin J. Kollingbaum
Dept. of Computing Science
University of Aberdeen, U.K.
m.j.kollingbaum@abdn.ac.uk

Wamberto W. Vasconcelos
Dept. of Computing Science
University of Aberdeen, U.K.
w.w.vasconcelos@abdn.ac.uk

Abstract

In this paper, we propose that access to data and knowledge be controlled through fine-grained, user-specified explicitly represented policies. Fine-grained policies allow stakeholders to have a more precise level of control over who, when, and how their data is accessed. A representation for policies is outlined, as well as a mechanism to control data access within a fully distributed system, in order to create a secure environment for data sharing. Security relies on an accurate recording of transactions to ensure that data access can be evaluated based on the prior actions of peers. We contrast the approach that we have taken in our implementation with Blockchain.

1 Introduction

With the emergence of data management applications, such as Smart Cities [CBN11, ZCWY14] and the Internet of Things [AIM10, GBMP13], we encounter large-scale data sharing as an important concept within these application domains and the need for the establishment of whole data sharing "ecosystems". Usually, data access is specified (and constrained) through the use of data access policies. These policies specify how data may (or may not) be accessed, changed and used. Access to data may lead to its transfer between two parties interested in this data. In order to perform such data transfer activities, these parties may establish extra agreements or "behavioural policies" – a "contract" between these communication partners – that govern such an exchange.

Traditional management of typical access policies tends to be centralised and, therefore, has a number of problems, such as information ownership and reliance on a central authority that may allow the manipulation of these policies and answer queries regarding current policy settings for data. We propose access to data and knowledge to be controlled through fine-grained, user-specified explicitly represented policies that regulate data exchange in a peer-to-peer environment, where some peers have data which they want to provide (called Providers) and some peers have data they want to acquire (called Requestors). Providers set policies that establish how their data can be accessed and by whom. These policies can be defined with different levels of granularity, allowing peers precise control over the sharing and use of their data.

Our policies may express general regulatory statements such as, for example, "no drug records and medical records can be obtained by the same party", or more specific, such as "I will only provide 10 records to each person". Fine-grained policies allow stakeholders to have a more precise level of control over who, when, and how their data is accessed. We propose a representation for policies and a mechanism to control data access within a fully distributed system, creating a secure environment for data sharing. We discuss data as if it were stored in a database, but this could be

In: A. Editor, B. Coeditor (eds.): Proceedings of the XYZ Workshop, Location, Country, DD-MMM-YYYY, published at http://ceur-ws.org

expanded to cover any form of structured information.

This proposal requires a method of enforcing policies without a reliance upon a centralised authority, and Blockchain[1] serves as a potential solution to this problem. Blockchain refers to a generalised concept for securing transactions, the most well known application of which (also called Blockchain) is a permissionless distributed database [Gri14, Pos16] based on the Bitcoin protocol [Nak16] that achieves tamper resistance by timestamping a hash of "batches" of recent valid transactions into "blocks". Each block references the prior timestamp, creating a cryptographically enforced chain. While Blockchain does eliminate some aspects of centralisation, it does still require the presence of data store nodes, or for all peers to store copies of the full chain. Throughout the paper we will be referring to the Bitcoin blockchain as Blockchain, as the generalised concept covers too wide a variety of applications to be adequately comparable to a specific solution. For instance, while the Bitcoin blockchain is permissionless, permissioned Blockchains do exist, such as Monax[2] where permissions allow control over who can participate in the consensus mechanism used to validate transactions.

Taking inspirations from encryption concepts in distributed applications, such as CryptDB [PRZB12], Blockchain [Gri14, Pos16] and Bitcoin [Nak16], we propose an alternative to Blockchain for managing data exchange in a fully-distributed environment. Our policies are enforced by a distributed infrastructure of "policy decision points" (PDP) throughout the network, taking inspiration from the traditionally centralized eXtensible Access Control Markup Language (XACML) PDP architecture [DKW08]. We regard a data exchange or sharing activity between peers (provider and requestor) as a transaction. Transactions are recorded and are an important means for checking policy compliance. During a data request, transaction records are taken into account to test whether a requestor meets the conditions of policies specific to such a request. Due to the distributed nature of policy decisions on each peer, we require the ability to secure components to be exchanged for this decision process.

In this paper we present the following:

- A description and architecture for our proposed mechanism (Section 2)
- An outline of how our proposed mechanism will perform in a standard transaction (Section 3)
- The syntax and formalism for a policy language that allows for fine-grained control over data access (Section 3.1)
- Details of a peer-to-peer simulation we have constructed as a prototype of our mechanism, and some initial evaluation of peer satisfaction within this prototype (Section 4)

[1]https://blockchain.info/

[2]https://monax.io/

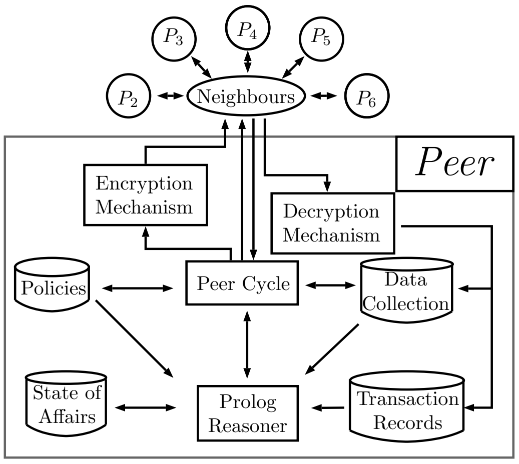

Figure 1: Architecture of a Peer

2 Approach

Our approach utilises a peer-to-peer network, in which every peer represents a party within an economy of data exchange. Each peer, upon joining the network, performs a bootstrapping operation by contacting a central server which issues them with a collection of neighbours and a unique (and signed) identifier. After this, all communication occurs between peers.

Each of our peers has a collection of policies that, initially, is defined specifically for the peer. These policies decide how data exchange is handled between peers, and can be updated as a peer's goals or knowledge change. Policies, aside from controlling the conditions under which data can be accessed, may also impose obligations upon anyone using them to access data. Obligations indicate an action, or set of actions, to be performed within a certain timeframe after receiving data. For instance, a policy might oblige you to not exchange data with any other peers for five execution cycles[3]. Obligations in our network must be policed, if a peer fails to complete an obligation within a set period of time that peer will assign itself the penalty cost attached to the obligation. This penalty comes in the form of "penalty cycles", cycles in which the operations your peer can perform are limited. In

[3]In our implementation, a cycle denotes a period of time in which each peer is allowed to perform three operations: respond to messages, process a single obligation, and send a single data request

a penalty cycle, you may only respond to others' requests for data, and may not send or process any data requests of your own. Since each peer pays a fixed cost[4] for participating in a cycle, and this cost is still paid for penalty cycles, obligations are effectively self-policed as it is within each peers interest to minimise the number of cycles they pay for but which provide them no benefit. This cost can be conceptualised as the price of power and bandwidth to participate, and is not paid to anyone within the network

To manage this cost, each peer upon joining the network sets a budget, which is the amount of value they are willing to spend as part of the network. Participating in cycles spends this value, and when a peer's budget is expended they will leave the network. Receiving data accrues value, according to values defined by each peer for each item of data, representing how much it is worth *to them*. This allows peers to weigh up the "profit" of a given transaction by balancing the value of the data gained against the loss from accepting the policies associated with it.

When deciding whether peers are allowed access to data, the reasoner must consult a peer's policies. To be able to go from this collection of policies to a decision requires knowing what transaction peers have (recently) taken part in. It is for this reason that our mechanism requires us to ensure that transaction records cannot be tampered with, and are stored exactly as they are issued. Data and message exchange are handled by the peer-to-peer protocol.

Blockchain at this point would timestamp a hash of "batches" of recent valid transactions into "blocks" on a number of data-store nodes. This block contains a reference to the prior timestamp, creating a cryptographically enforced chain. While this does provide the security we require for our transactions, it introduces an element of centralisation in the requisite always-on data-store nodes. Our mechanism is designed to be closer to fully distributed, with only a reliance on sporadic contact with a centralised server: once on joining the network, then periodic contact every few days to refresh our encryption mechanism.

With this in mind we propose a similar method of securing transactions that removes reliance on data-store nodes. Every peer in our network stores copies of their own transactions. These transaction records are generated during a data request by the party providing data (the provider). On generation they are signed by the provider and encoded using a refreshable encoding mechanism. By this, we mean an encoding mechanism which can be periodically updated to modify its protocol to ensure that if it is broken it does not remain so for long. We have not yet chosen a specific mechanism for this, but a simple example would be a periodically updated key phrase issued by the host cache hashed along with each record. The encoding mechanism and signature ensures that any peer can verify the authenticity of a transaction record, but no peer can impersonate another to falsify records. The encoding mechanism also prevents a human from injecting false records into their peer's collection, as they would be unable to replicate the encoding. These encoded records are then transferred to the requesting peer along with the data they requested. As part of the operation to extract requested data from this message, the transaction records are stored in the peer's collection.

Ultimately this system for recording transactions achieves the same purpose as Blockchain, however it potentially sacrifices some security in exchange for usability. Blockchain achieves security through consensus of encrypted transactions on data store nodes, where our approach relies on cryptographic security on each peer without a consensus element. We intend to carefully design our proposed encoding mechanism so that the predicted time to break it exceeds the refresh period for the encoding; ensuring that by the time it is broken it is no longer of any use. Prior to creating this encoding mechanism we had intended to use homomorphic encryption to secure transaction records, however the time required to perform operations was infeasibly long for our envisioned scenarios.

A (simplified) architecture of a peer is shown in Figure 1. Based on this architecture, peers can be implemented that employ (possibly encrypted) communication. Any incoming messages are recorded as transaction records. The peer reasons about what transaction records are relevant, whether to accept a set of policies, or whether to permit access to data. Each peer holds four main collections: "Policies", "Data", "Transaction Records", and "State of Affairs". The data set "State of Affairs" holds the information of everything the peer knows about the network, e.g., who has what data, who has denied them data in the past, what they are currently obliged to do.

3 A Typical Transaction

To help illustrate how our mechanism operates, we will now walk through a typical transaction, noting any significant operations within the process. The transaction will proceed as follows, where each number corresponds to the numbers in Figure 2:

1. Requestor R sends a data request for data D to provider P.

2. P processes this request, and if the provider pos-

[4] Costs are parameters which can be "tweaked" by engineers/designers to favour certain behaviours

sesses D, it will create a list of policies relevant to D or R[5]. If any policy in this list prohibits sending D to R (regardless of transaction records), then the data request will be denied, a transaction record will be generated and sent to R, and the process will terminate here. If not, P will send a message to R containing the policies associated with D.

3. R will reason on these policies to determine whether to accept them using a cost evaluation function.

4. If the policies are accepted, R then must determine which transaction records (representing historical data access information) are "relevant" to these policies. To achieve this, the mechanism checks each policy and extracts a list of unique data elements referred to in the policy. The mechanism will then identify which of R's transaction records reference these relevant data elements. The identified records are then sent by R to P.

5. P receives records from R and determines if the records permit the provision of D to R. If there are any pre-obligations associated with the policies then:

   (a) P will inform R of these pre-obligations

   (b) R will take the necessary steps to fulfil these pre-obligations

   (c) R will inform P of the completion of these pre-obligations, attaching appropriate proof(s)

6. If the data exchange is permitted, then sending D (the requested data and a record of the transaction, encrypted in a single package) to R is approved, and D will be sent from P to R. If the data exchange is prohibited, then a message will be sent containing just the encoded record of the transaction

7. R will decrypt the package, adding the transaction record to its collection, and storing the data (if any was sent). This single encrypted package is received by the mechanism, ensuring that the transaction record will be stored as ignoring it will prevent receipt of data.

[5]For now a simple relevance check is performed, checking if each policy references either R (or a group they are part of) or D

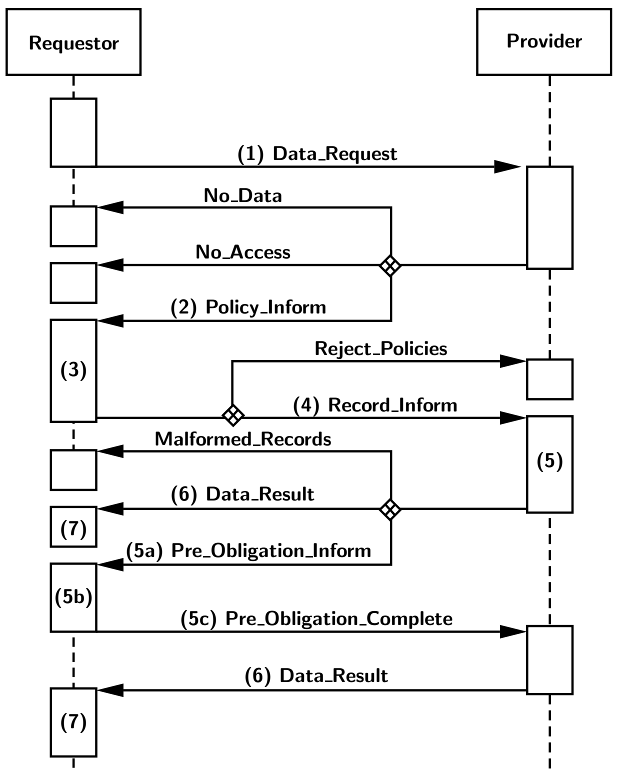

Figure 2: AUML for a Typical Transaction

3.1 Policy Language

Policies enforce how data can be shared within the network. Some are network-wide (e.g., "no drug records and medical records can be obtained by the same party"), while others can be specified by an individual provider (e.g., "I will only provide 10 records to each person"). These policies are stored by each peer locally. We define our policies as follows:

Definition 1 (Policies). *A policy π is a tuple $\langle M(\mathsf{odality}), I(\mathsf{d}), D(\mathsf{ata}), \varphi(\mathsf{Conditions}), \sigma(Pre-Obligations), \omega(\mathsf{Obligations})\rangle$ where*

- *$M \in \{\mathsf{F}, \mathsf{P}\}$ is a deontic modality/operator, denoting a prohibition (F) or a permission (P).*
- *$I \in \{id_1, \dots, id_n\}$ is a unique peer identifier*
- *$D \in \mathbf{T}$ is a descriptor of a data element (representing a single record or a dataset), where a taxonomy $\mathbf{T} \subset \mathbb{N}$ is a subset of natural numbers. We define a reflexive and transitive subsumption relation $\sqsubseteq \subseteq \mathbf{T} \times \mathbf{T}$, over a taxonomy $\mathbf{T}$ to define its structure.*
- *$\varphi = L_1 \wedge \cdots \wedge L_m$ is a conjunction of literals (cf. Def. 2)*
- *$\sigma = L_1 \wedge \cdots \wedge L_m$ is a conjunction of literals (cf. Def. 3)*

- $\omega = (L_1^1 \wedge \cdots \wedge L_m^1 \wedge \rho^1 \wedge \sigma^1) \wedge \cdots \wedge (L_1^n \wedge \cdots \wedge L_m^n \wedge \rho^n \wedge \sigma^n)$ *is a conjunction of conjunctions of literals (cf. Def. 4)*

A sample policy using our definition is $\langle \mathsf{P}, \mathit{peer}_1, 010000, \mathit{recordsAccessed}(\mathit{peer}_1, 010000) < 5, \top, (\mathit{provide}(020000, 5, \mathit{peer}_2), 5, 10)\rangle$ representing a permission to allow the peer with id 1 to access up to 5 elements of 010000, but in return obliging them to provide 5 elements of 020000 to the peer with id 2 with a penalty of 5 units if this not completed within 10 cycles. Our policies above refer to descriptions of data elements – these are labels describing, for instance, fields of a data base or names of predicates of an ontology [CJB99]. These labels are represented using a simple numerical encoding where a natural number represents a data label and it's inheritance relationship(s).

Our policies allow the representation of *conditions* under which the policy should hold – this is what the component φ of Def. 1 is meant for. These conditions are joined using conjunction, disjunctions can be captured through the use of alternative policies (one policy representing conditions A and B, one representing C and D). We have designed a simple vocabulary of "built-in" tests which are relevant to our envisaged application scenarios, and these are defined below:

Definition 2 (Conditions)**.** *A literal L is one of the following, where $D \in \mathbf{T}$ (a descriptor of a data element), $\circ \in \{<, >, \leq, \geq, =\}$ is a comparison operator, and $n \in \mathbb{N}$ is a natural number:*

- *$\mathit{recordsAccessed}(I, D) \circ n$ holds if the number of retrieved instances of data element D from peer I satisfies the test "$\circ\ n$".*
- *$\mathit{recordsRequested}(I, D) \circ n$ holds if the requested number of instances of data element D from peer I satisfies the test "$\circ\ n$".*
- *$\mathit{requests}(I, D) \circ n$ holds if the number of requests of data element D from peer I satisfies the test "$\circ\ n$".*
- *$\mathit{lastRequest}(I, D) \circ n$ holds if the last request (successful or not) for data element D from peer I satisfies the test "$\circ\ n$", where n represents a time period*
- *$\mathit{lastAccess}(I, D) \circ n$ holds if the last successful request for data element D from peer I satisfies the test "$\circ\ n$", where n represents a time period*
- *$\mathit{time}(T) \circ n$ holds if the date/time (in various formats) T satisfies the test "$\circ\ n$", where n represents a date/time in the same format as T.*
- *$\bot$ and $\top$ represent, respectively, the vacuously false and true values.*

Our policies also allow the representation of *pre-obligations* (represented by σ in Def. 1), actions which must be undertaken *before* data exchange can occur, and *obligations* (represented by ω in Def. 1), actions which must be undertaken, within a given timeframe, *after* data exchange has occurred to avoid penalisation. Pre-obligations have a subset of the vocabulary designed for obligations. Our pre-obligation vocabulary is given below:

Definition 3 (Pre-Obligations)**.** *A literal L is one of the following:*

- *$\mathit{obtain}(D, N)$ obliges a peer to obtain N items of data type D*
- *$\mathit{provide}(D, N, I)$ obliges a peer to provide N items of data type D to peer I*
- *$\mathit{adopt}(\pi, T)$ obliges a peer to adopt policy π, for a duration of T "cycles"*
- *$\bot$ and $\top$ represent, respectively, the vacuously false and true values.*

In addition to this, our obligation vocabulary is given below:

Definition 4 (Obligations)**.** *Obligations consist of a conjunction of conjunctions of literals, where a literal L is one of the following, and each conjunction of literals is associated with a pair of natural numbers representing the penalty (ρ a numeric representation of the penalty accrued by violating the associated obligations) and duration (σ a numeric representation of the time in which the associated obligations must be fulfilled):*

- *$\mathit{obtain}(D, N)$ obliges a peer to obtain N items of data type D*
- *$\mathit{provide}(D, N, I)$ obliges a peer to provide N items of data type D to peer I*
- *$\mathit{adopt}(\pi, T)$ obliges a peer to adopt policy π, for a duration of T "cycles"*
- *$\mathit{inform}(I)$ obliges a peer to send a message to peer I when the obligations associated with this policy are completed, attaching proof of completion*
- *$\bot$ and $\top$ represent, respectively, the vacuously false and true values.*

Our policy language as described in this section is a compromise between expressiveness and complexity, both of mechanisms to manipulate it and the accessibility of the language.

4 Evaluation

To evaluate the feasibility of our mechanism, we have constructed a prototype using a combination of PeerSim[6] and SWI-Prolog[7]. PeerSim provides a platform for easily constructing peer-to-peer simulations in Java with a high level of experimental control. SWI-Prolog allows us to perform reasoning tasks more easily than in Java. This prototype is still under construction, however enough has been implemented to provide preliminary results as to the feasibility of our solution.

Our simulation measures two metrics to track the effectiveness of our mechanism. The first is satisfaction, which is the number of peers who have been able to get all the data that they want to get. The second is normalised profit, which is the average profit peers have accrued through data exchange, "normalised" by removing the cost of participating in cycles. Without normalising the profit is an extremely low negative value as many peers are unable to get their data and expend their whole budget on remaining in the network without any profit. With higher levels of satisfaction, the average profit should increase significantly, removing the need to normalise it.

Each data item in our network has a type (label) and a payload which is a random string. This type could, for instance, be a temperature reading; in this case the data body would represent the actual value in degrees, and perhaps additional metadata. The data body gives the data item uniqueness, which makes more sense in a data exchange context. Otherwise the simulation would just be exchanging generic temperature records with no concept of what that represents.

Since all of our peers are generated randomly, the chance of a given peer being satisfied are a result of a number of factors. Each peer can be visualised as series of concentric circles with a number of "ports" which can be open or closed depending on their state (this visualisation is given in Figure 3). To get the data that you want, you must find a peer whose "ports" are in a configuration which allows you to pass through to the centre. The conditions that must pass are:

1. Does the peer have the data you need?
2. Does the peer have a policy which *could* allow you to access the data?
3. Does this policy provide you with an overall gain in value (or an acceptable loss in value)?
4. Do your previous transactions allow you to use this policy?

[6]http://peersim.sourceforge.net/
[7]http://www.swi-prolog.org/

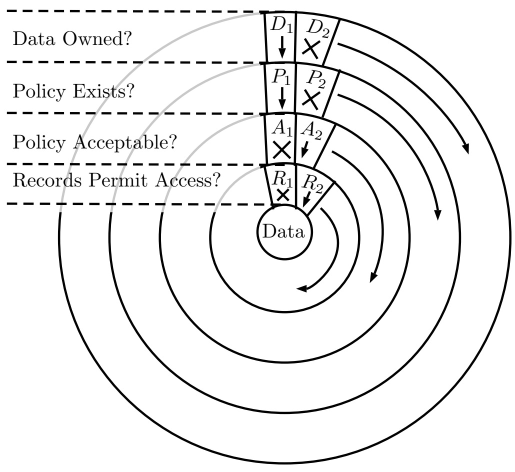

Figure 3: Peer Data Access

To increase the chances of a peer getting data, the number of ports in each of these circles should be maximised. This highlights a number of variables in our simulation that can control the likelihood of achieving a high percentage of satisfaction, as these effectively control the number of ports which exist, and thus which could potentially be open. Our network is draconian and assumes a default prohibit stance unless otherwise permitted by a policy. Due to this the largest limiting factor in our simulation is the number of policies each peer generates, as this affects the potential access points to their data. The other variables which are controlled are:

- Number of neighbours each peer starts with
- Amount of data that each peer owns
- Amount and value of data that each peer wants
- The total types of data in the network

All of our experiments are run with 1,000 peers (each with 10 neighbours) over 250 cycles. There are 10 types of data in the network; each peer owns between 1 and 10 of these types and desires between 1 and 5 of these types. In the network, 500 peers will only provide data (and are ignored in satisfaction metrics), 250 will only request data, and 250 will both provide and request data. We will be varying the number of data access policies that each peer may generate.

The results of our evaluation are shown in Figure 4. Our initial evaluation shows relatively low satisfaction rates for up to 1000 policies ($< 30\%$), this is likely due to the random nature of our simulation. Since all peers are generated randomly, including their policies, data collection, and neighbours, there are potentially a lot of peers who in fact have no way of accessing the data they want. The number of peers satisfied is taken as an average of ten iterations of the experiment.

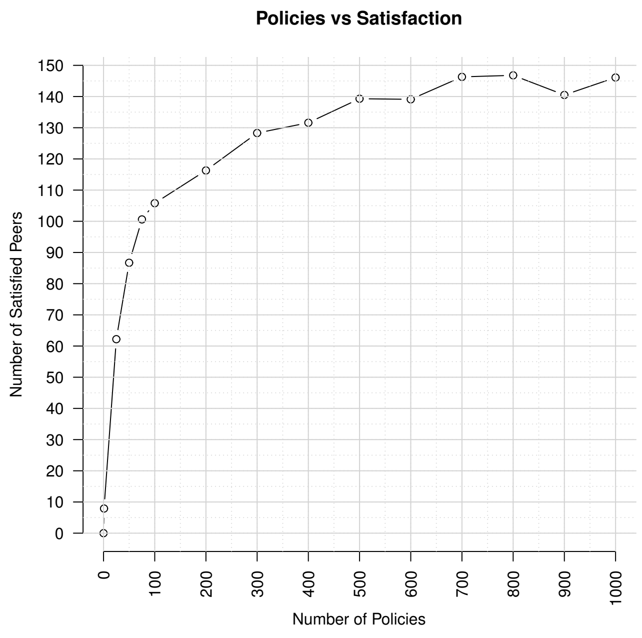

Figure 4: Results comparing Number of Policies and Number of Peers Satisfied

Within our simulation, load on each peer remains relatively constant while increasing number of peers, as each has a relatively small number of neighbours they are in direct contact with. Due to this, increasing the number of peers simply results in a linear increase in simulation time; increasing peers by a factor of ten also increases simulation time by, on average, a factor of ten. Increasing the complexity of reasoning processes performed by each peer will similarly have effects on the time each peer takes per cycle, but this should be relatively constant between peers. Increasing the number of cycles tend to result in diminishing time per cycle, as peers drop out of the network.

In future evaluations, it would make sense to add weightings to the random generation functions, and also to add sanity checks to peers to ensure that every item of their data is accessible by at least one peer (without any other conditions). We can see that the mechanism is feasible, since increasing the number of policies greater than 500 does have reasonable levels of satisfaction. Having such a high number of policies increases the chances of producing a set of policies which allow access to their data, and this subset of policies are likely to be those which contain a reasonable number of conditions (and no contradictions). Above 500 policies we observe that satisfaction levels begin to plateau, which further demonstrates that satisfaction is actually the combined effect of a number of variables. This number of policies would seem to have effectively removed this as a limiting factor, and it is likely that the other variables are what is now restricting the satisfaction levels.

5 Related Work

Our work is based on established peer-to-peer (P2P) technologies and operations [BYL09, SYBA10]. P2P refers to networks in which "peers" communicate directly with each other, with minimal use of a centralised server. P2P networks can have a variety of different topologies but at a high level they are either structured, where peers must organise themselves according to a set of conditions, or unstructured, where peers have a set of unrelated "neighbours" with which they communicate.

Our peer-to-peer model makes use of an unstructured hybrid approach in which peers have an initial collection of neighbours which they communicate with, but also that they can add to by storing paths as they exchange data with other peers. This can aid in the efficiency of data sharing, as peers will slowly converge on a set of neighbours which deal with the types of data they are interested in. It can also potentially mitigate issues of neighbours leaving the network and isolating peers. Our peer-to-peer protocol adopts a directed flood request model, in which peers consult their knowledge base before sending data requests to attempt to more efficiently determine a target.

Peer-to-peer simulations are based on a "network" of agents, lightweight independent processes which each act according to their own agenda [Hay99]. Agents can be cast as a community of interconnected components, as in the Internet of Things [AIM10, GBMP13]. Within our simulation peers act as agents, adapting their behaviour as the simulation progresses.

Within our peer-to-peer system it is important for peers to have control over who, when, and how their data is shared. This can be achieved through the use of policies/norms [Ser01, ST95, vW63]. Norms are a formal representation of expected behaviours of software agents, such as prohibitions, and duties. An integral part of norms concerns deontic logic [MW93, VW51], considering permissions, prohibitions, and obligations. Norms and agents are often paired together, as norms provide means to control behaviour in societies of self-interested components [Dig99]. When considering norms it is also important to be aware of conflict detection and resolution[VKN09].

This research builds upon the work we reported in [CKSV16], but expands upon it by introducing new concepts into our mechanism and detailing the prototype that we are currently building. The combination of policies and data presented in this paper draws upon the techniques and methods reported in [PV15], but it has a significantly different focus, and most importantly, provides a distributed solution which can scale up and is resilient to many kinds of attacks. There have been other research threads which consider "Policy Carrying Data" [SWA15, WYD$^+$13], which suggests similar concepts to our approach but without the focus on a distributed environment. They instead focus on a centralised scenario which creates a single-point of failure, a lack of scalability, and (potential) data ownership issues.

We note a substantial overlap between our proposal and initiatives such as Blockchain and Bitcoin[8]. Blockchain is a permissionless distributed database [Gri14, Pos16] based on the Bitcoin protocol [Nak16] that achieves tamper resistance by timestamping a hash of "batches" of recent valid transactions into "blocks". Each block references the prior timestamp, creating a cryptographically enforced chain. Blockchain requires either a group of always-on data-store nodes, or for every individual "peer" to store a copy of the full chain. There are important similarities between Blockchain and our proposal, but Blockchain is centralised in nature and has high storage requirements on data store nodes. In particular, Blockchain and its application in smart contracts (for instance, the Ethereum Project[9]) could be cast as comparable to our work. In [IGRS16] they propose a method for capturing logic-based smart contracts in Blockchain which appears to be closer to our methods in that it utilises a rule-based engine to process the smart contracts rather than traditional procedural approaches.

6 Conclusions and Future Work

In this paper we have detailed a solution to managing distributed data sharing without the presence of a centralised authority. We have highlighted one of the key security problems with this solution, and we have provided details of a mechanism which can solve this problem. For the sake of presenting a crisp and compact research problem, we have simplified many aspects of our solution within this paper.

The next task we want to complete with this research will be to finish the development of the prototype. From this we intend to run a series of in depth evaluations on the prototype to better explore the effect of the variables discussed in Section 4. This will allow us to not only improve the prototype, but to use this as a drive to further improve the underlying concepts of our mechanism. Following this we want to do some further work on our transaction record encoding scheme. Currently we are merely assuming the existence of some hypothetical refreshable encoding scheme, but it would be interesting to find a candidate for this and to see how it performs in practice.

In the future it would also be interesting to attempt to implement a version of our approach as a smart contract using Blockchain, this could provide a baseline for comparison with our current peer-to-peer driven approach. This could then allow is to conduct a more detailed qualitative comparison between our proposed method and Blockchain, something which we have only touched on briefly in this paper.

We acknowledge that our policy language is still fairly basic, and as such believe that there is more we can do to enrich the vocabulary of it to allow peers even more control over how their data is exchanged. It would also be interesting to investigate in more detail the interplay of complex policies within a network, with a view to determining conflict detection and resolution strategies. At the present we have only a basic mechanism for detecting conflict between policies, and it is resolved using a cost evaluation function to determine which policy(s) to break: the new (conflicting) policy or existing policy(s).

[8] `https://bitcoin.org/en/`

[9] `https://www.ethereum.org/`

References

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[BYL09] John Buford, Heather Yu, and Eng Keong Lua. *P2P networking and applications*. Morgan Kaufmann, 2009.

[CBN11] Andrea Caragliu, Chiara Bo, and Peter Nijkamp. Smart cities in Europe. *Journal of Urban Technology*, 18(2):6582, 2011.

[CJB99] Balakrishnan Chandrasekaran, John R Josephson, and V Richard Benjamins. What are ontologies, and why do we need them? *IEEE Intelligent systems*, (1):20–26, 1999.

[CKSV16] Samuel R Cauvin, Martin J Kollingbaum, Derek Sleeman, and Wamberto W Vasconcelos. Towards a distributed data-sharing economy. In *COIN@ECAI*, 2016.

[Dig99] Frank Dignum. Autonomous agents with norms. *Artificial Intelligence and Law*, 7(1):69–79, 1999.

[DKW08] Vijayant Dhankhar, Saket Kaushik, and Duminda Wijesekera. Securing Workflows with XACML, RDF and BPEL. In *Proceedings of the 22nd Annual IFIP*

WG 11.3 Working Conference on Data and Applications Security, pages 330–345, Berlin, Heidelberg, 2008. Springer-Verlag.

[GBMP13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[Gri14] Ilya Grigorik. Minimum viable block chain. "`https://www.igvita.com/2014/05/05/minimum-viable-block-chain/`", Accessed On: 2014.

[Hay99] C. C. Hayes. Agents in a nutshell-a very brief introduction. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):127–132, 1999.

[IGRS16] Florian Idelberger, Guido Governatori, Régis Riveret, and Giovanni Sartor. Evaluation of logic-based smart contracts for blockchain systems. In *International Symposium on Rules and Rule Markup Languages for the Semantic Web*, pages 167–183. Springer, 2016.

[MW93] John-Jules C. Meyer and Roel J. Wieringa. Deontic logic in computer science normative system specification. In *International Workshop on Deontic Logic in Computer Science*, 1993.

[Nak16] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `www.cryptovest.co.uk`, 2008, accessed: June 2016.

[Pos16] Postscapes. Blockchains and the internet of things. `http://postscapes.com/blockchains-and-the-internet-of-things`, Accessed: March, 2016.

[PRZB12] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Processing queries on an encrypted database. *Communications of the ACM*, 55(9):103–111, 2012.

[PV15] Julian Padget and Wamberto W Vasconcelos. Policy-carrying data: A step towards transparent data sharing. *Procedia Computer Science*, 52:59–66, 2015.

[Ser01] Marek Sergot. A computational theory of normative positions. *ACM Transactions on Computational Logic (TOCL)*, 2(4):581–622, 2001.

[ST95] Yoav Shoham and Moshe Tennenholtz. On social laws for artificial agent societies: off-line design. *Artificial Intelligence*, 73(1):231–252, 1995.

[SWA15] Stefan Saroiu, Alec Wolman, and Sharad Agarwal. Policy-carrying data: A privacy abstraction for attaching terms of service to mobile data. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pages 129–134. ACM, 2015.

[SYBA10] Xuemin S. Shen, Heather Yu, John Buford, and Mursalin Akon. *Handbook of peer-to-peer networking*, volume 34. Springer Science & Business Media, 2010.

[VKN09] Wamberto W. Vasconcelos, Martin J. Kollingbaum, and Timothy J. Norman. Normative conflict resolution in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 19(2):124–152, 2009.

[VW51] Georg H. Von Wright. Deontic logic. *Mind*, 60(237):1–15, 1951.

[vW63] Georg Henrik von Wright. *Norm and Action: A Logical Enquiry.* Routledge and Kegan Paul, 1963.

[WYD+13] Xiaoguang Wang, Qi Yong, Yue-hua Dai, Jianbao Ren, and Zhang Hang. Protecting Outsourced Data Privacy with Lifelong Policy Carrying. In *10th IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, HPCC/EUC 2013, Zhangjiajie, China, November 13-15, 2013*, pages 896–905, 2013.

[ZCWY14] Y. Zheng, L. Capra, O. Wolfson, and H. Yang. Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology*, 5(3):38:1–38:55, 2014.