# Distributed Access Control with Blockchain

Jordi Paillisse*, Jordi Subira*, Albert Lopez*, Alberto Rodriguez-Natal†,
Vina Ermagan†, Fabio Maino† and Albert Cabellos*
*UPC-BarcelonaTech, Barcelona, Spain - jordi.subira@est.fib.upc.edu, {jordip, alopez, acabello}@ac.upc.edu
†Cisco Systems, San Jose, CA, USA - {natal,vermagan,fmaino}@cisco.com

***Abstract*—The specification and enforcement of network-wide policies in a single administrative domain is common in today's networks and considered as already resolved. However, this is not the case for multi-administrative domains, e.g. among different enterprises. In such situation, new problems arise that challenge classical solutions such as PKIs, which suffer from scalability and granularity concerns. In this paper, we present an extension to Group-Based Policy -a widely used network policy language- for the aforementioned scenario. To do so, we take advantage of a permissioned blockchain implementation (Hyperledger Fabric) to distribute access control policies in a secure and auditable manner, preserving at the same time the independence of each organization. Network administrators specify polices that are rendered into blockchain transactions. A LISP control plane (RFC 6830) allows routers performing the access control to query the blockchain for authorizations. We have implemented an end-to-end experimental prototype and evaluated it in terms of scalability and network latency.**

***Index Terms*—blockchain, access control, authorization**

## I. Introduction

Group-Based Policy (GBP), or policy-based networking [1] is a declarative approach to defining network behaviour. Network administrators specify network endpoints, groups of endpoints and their policies using a high level language, which is later translated to network configurations. One of these languages, GBP is widely employed in the industry, for example in OpenStack's Neutron network API [2]. It can define rules between servers and clients, service chains, etc.

Until now, GBP has been conceived as a language for a single administrative domain. In this paper, we analyze if we can extend it to several administrative domains, preserving at the same time their independence. For example, we want to make it possible for an administrator in company B to allow a VPN connection from a user in company A by simply typing:

```
createPolicy from=userA to=VPNserverB action=allow
```

Typically, there are two solutions for this scenario: (i) manual, by means of issuing a digital certificate and giving it to the users (so they use it later to authenticate the connection), or (ii) leveraging structures based on PKI systems, namely cross-domain certification or bridge CA certificates [3]. These structures allow the co-existence of several CAs and ensure mutual trust.

However, these approaches present some limitations that have hindered their deployment. First of all, scalability: in scenarios with thousands or tens of thousands of users, the manual approach is unfeasible, and cross-certificating $N$ domains means -in the worst case- issuing $\sim N^2/2$ certificates [3], [4]. Second, granularity: it is not possible to define different policies for different users without issuing more certificates, further affecting scalability. Finally, management: PKIs are cumbersome to manage, especially day-to-day operations like adding and removing users, revocation (requires a CRL subsystem) or key rollover.

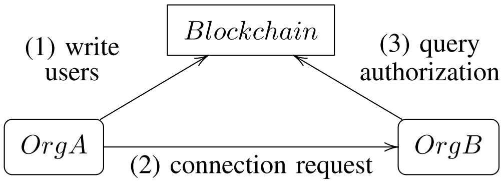

Fig. 1. Global architecture.

With these limitations in mind, in this paper we propose using a blockchain to overcome them. In such blockchain, each organization defines its users and resources, and specifies which users -from other organizations- can access its resources. Upon an access request, routers query the blockchain to verify authorization (fig. 1).

Thanks to blockchain's particular properties, we can design an access control system that improves on several of PKI's limitations: (i) *increased scalability:* when we establish a new relationship in a PKI, we have to cross-certificate the new entity with the rest. In a blockchain, however, we can directly reference previous transactions/users. This reduces the number of required certificates (transactions in this case). (ii) *improved granularity and flexibility*: since we can associate each resource or user with a private key, we can alter its state without affecting the rest. This includes both its validity and other data, for example, we can assign different policies to different users. (iii) *simpler management:* the transactional nature of blockchain makes management simpler: the aforementioned common operations (key rollover, revocation) can be encoded as new transactions, instead of requiring a dedicated subsystem, like CRLs and manifests.

In this paper we present an architecture to support this use-case, we describe a practical end-to-end implementation and evaluate its performance to demonstrate its feasibility. Our results show that we can store thousands of access polices with modest storage, and achieve linear update times on a permissioned blockchain.

978-1-5386-8088-9/19/$31.00 ©2019 IEEE

## II. Why Blockchain?

Our use-case presents two particular characteristics: (i) its participants have limited trust in each other, and (ii) they want to retain full control over the access policies. This is because in a multi-enterprise scenario: (i) companies are not willing to leave access control to a third party, and (ii) each company must be able to revoke any access policy at any moment in time, respectively.

These requirements match the characteristics of any blockchain. Regarding the first demand, its consensus algorithm ensures that no single entity controls the blockchain and avoids having to fully trust its participants. The second requirement is covered by the fact that blockchain assets are controlled by their associated private key owner, not by a centralized entity.

On the other hand, an approach like this is much more complex in a classical PKI because it cannot fulfill the previous two requirements. The first one because the CA is the single point of trust in the system, which forces all participants to trust it. Furthermore, it cannot meet the second as a consequence of the centralized trust: the CA can unilaterally alter state by means of certificate revocation.

As mentioned before, other PKI schemes could provide equivalent functionality, such as bridge CA certificates, but in this case the bridge CA certificate is still a single point of trust, thus it is not significantly different from a conventional CA. Cross-domain certification may prove useful, however, it presents scalability limitations because each new CA has to cross-certify with all the existing ones.

In addition, a blockchain can alleviate this scalability concerns: we can reduce the number of required certificates (transactions in this case), since a blockchain allows directly referencing existing transactions, instead of re-certifying with the PKI CA. In turn, this simplifies the verification of the chain of authorizations, i.e. it is not necessary to go up the CA, then down to the cross-certificated one. Authorization emerges directly from the originating organization.

Finally, thanks to emerging private blockchain platforms we can provide a certain degree of privacy for their users (as opposed to public blockchains like Ethereum) and improve some of their performance metrics (sec. III-B).

## III. Architecture

We can describe our architecture as a three-layer system (fig. 2):

***Policy:*** an intent-driven interface allows administrators to specify users, resources and access policies. These polices are rendered into blockchain transactions.

***Blockchain:*** a blockchain stores all the information and ensures its integrity and accuracy.

***Network:*** routers access the blockchain via an API to determine if a particular user can access a specific resource. If the user is authorized, they retrieve authentication information to establish a security association and allow the connection.

The following sections provide details on each element.

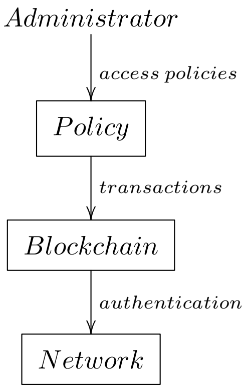

Fig. 2. Layered architecture.

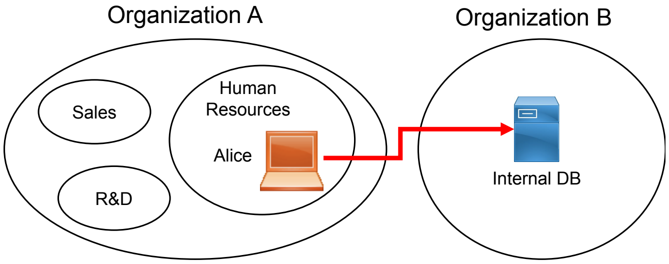

Fig. 3. Example scenario

### A. Policy interface

Administrators use a simple CLI, based on GBP, to perform management operations, such as creating/deleting users, groups of users, policies and resources, as well as querying the blockchain for specific policies, users, etc. We have chosen GBP because it is widely adopted in the industry [2] and its semantics align pretty well with our use-case.

Specifically, we can accommodate our use case to the OpenStack syntax simply re-using some of its commands. For example, consider that organization B wants to grant access to its internal database to Alice from the Human Resources department of an external organization A (figure 3). First, company A creates a member for Alice:

```
gbp member-create alice
```

Then, company B creates a group for company A's user and adds Alice into it:

```
gbp group-create dbaccess --add:orga.alice
```

It also creates the internal database as a member:

```
gbp member-create internalDB
```

Finally, organization B creates the policy associated to its database and company A's user, allowing access from the group *dbaccess* to its member *internalDB*:

```
    gbp policy-rule-create external-human-res
--src:dbaccess --dst:internalDB --actions allow
```

The GBP syntax can be extended with more options, for example, adding a one week timeout to Alice's membership in the *internalDB* group:

```
    gbp group-create dbaccess --add:orga.alice
--timeout 1w
```

This custom logic can be easily implemented thanks to the ability of some blockchains to run smart contracts.

### *B. Blockchain*

In this section we discuss two major design decisions we took for our blockchain.

***Participants:*** We believe that a private blockchain (only authorized members can access it) fits better in this scenario than a public, mainly because its participants are not willing to make their access policies public. Communicating access policies only to a group of companies is sufficient for correct operation.

***Consensus algorithm:*** We argue that a BFT protocol suits our use case, due to the following reasons: (i) Security: classical BFT protocols such as XFT [5] or BFT-SMART [6] offer proven security guarantees, as opposed to PoW or PoS algorithms, some of which lack a formal security analysis or a mature implementation. (ii) the access-control PKI of the private chain can be re-used in the BFT protocol, since they require some kind of node authentication. (iii) Higher throughput: BFT algorithms typically reach consensus faster than PoW or PoS, thus increasing the amount of transactions per second. (iv) Immediate finality: in a BFT protocol, when a transaction has been added in the chain, it will never be removed. On the contrary, a Bitcoin fork prevents immediate finality, and (v) We can avoid well-known PoW/PoS drawbacks, e.g. high energy consumption or limited throughput.

Finally, it should be noted that BFT-based chains suffer from scalability concerns, i.e., they cannot scale to as many users as well-know PoW or PoS chains like Bitcoin (in the order of millions). However, this is not our case: a chain with hundreds or even tenths of companies would be perfectly functional.

### *C. Network*

In order to perform the access control, we have chosen the Locator/ID Separation Protocol (LISP, RFC 6830). LISP is a request-response protocol that allows the communication of control and data planes. In our scenario, routers can easily retrieve the blockchain access control policies from the control plane with minimal modification of the base protocol. For this particular use-case, LISP is conceptually equivalent to OpenFlow [7]. Hence, we can use other protocols for this task, such as the aforementioned OpenFlow or P4 Runtime [8].

In a nutshell, we store the access policies in the LISP control plane and update them through the blockchain. LISP-enabled routers query the control plane to determine if a particular user can access the requested resource. Users authenticate to the router by means of including their signature in the LISP control plane messages (Map Request and Map Reply).

### *D. Typical Workflow*

Fig. 4 presents an example of the typical workflow in this architecture, in which two companies set up a secure connection from User A of company A to Resource B located in company B.

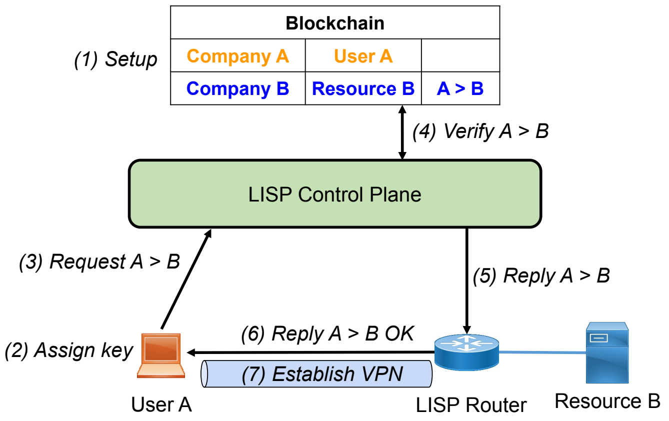

Fig. 4. Typical architecture workflow. For the sake of clarity, the LISP control plane is depicted as a single entity. In a real deployment each company would have its own control plane connected to its blockchain instance.

1) At setup time, administrators from both companies store the required information in the chain. Company A adds User A and its public key. Company B details its resource (Resource B) and grants access to company A's user (A > B). They use a CLI similar to the example in sec. III-A.
2) Company A assigns User A its credentials (public-private keypair), with the public key being the one in the blockchain.
3) When User A wants to connect to Resource B, it sends a LISP control message to the LISP Control Plane. The message is signed by User A.
4) The LISP Control Plane verifies the signature and checks the access policy against the blockchain.
5) If they are correct, it sends a reply message to Company B's LISP Router.
6) The LISP router sends a reply message with the cryptorgraphic material for data plane encryption, in order to establish a security association with User A. This message is encrypted with User A's public key.
7) The LISP router and User A start a secure connection, e.g. with LISP-CRYPTO (RFC 8061) or other VPN protocols (L3VPN or equivalent).

## IV. Implementation

We have built an end-to-end prototype encompassing the three components of the aforementioned architecture: GBP interface, Blockchain, and Network. It is available as open-source code[1].

### *A. GBP Command Line*

We designed a CLI inspired on GBP commands that allows the creation/deletion/retrieval of users, groups of users, resources and access policies, highly similar to the examples in sec. III-A. For example `create resource` creates a new resource for the organization. Additional options in the commands specify the IP address of the resource, the public key of a user, etc.

[1] https://github.com/JordiSubira/DGBP

### B. Blockchain

Our prototype is based on the Hyperledger project (HL), an open-source permissioned blockchain implementation. Specifically, we have chosen its Fabric [9] framework because of its maturity, flexibility and business-orientation. Moreover, thanks to Fabrics's channels, we can establish private communications among sub-sets of companies if privacy is a strong concern. In this section we summarize the different configuration parameters and implementation details of our prototype in the HL Fabric framework.

*1) Assets:* We defined the following elements in the chain:
***Users:*** Source endpoints, identified by a public key and including other information: originating organization, IP address, name and department.

***Departments:*** A group of users within an organization, identified by department name and their belonging organization[2].

***Resources:*** Destination endpoints, identified by an IP address and the associated organization.

***Policies:*** Access control lists that grant access either from a source endpoint (user) or from a group of users (department) to a destination endpoint (resource). They are identified by a composite key (source-destination). Typically, the source endpoint is a user or department of another organization and the destination is a resource of the issuing organization. Policies can contain other information, such as the time frame in which the connection is allowed or an expiry time.

*2) Membership Service Provider:* Each organization is identified by a MSP (Fabric's PKI for blockchain nodes).

*3) Chaincode:* We imposed as a global constraint that only the organization that creates an asset can alter its state (e.g. delete, associate with another asset, etc). We enforce this by binding all assets to their respective MSP, and rejecting any modification from a non-owner MSP.

In addition, any organization within the same HL channel can query information about any asset of any organization in the chain.

*4) Endorsement policy:* In our implementation, all members have to endorse any transaction. However, other schemes are possible thanks to Fabric's flexibility. Depending on the level of trust among the participating organizations and the particular use-case, we can adjust the minimum number of endorsements. Some examples are: half + 1 of the members, $2f+1$ valid signatures out of $n$ endorsers (assuming $f$ faulty endorsers and $n > 3f$), or AND/OR syntax (`member A OR members (B,C,D)`), etc.

*5) Ordering Service:* We leveraged the SOLO ordering service (i.e. a centralized orderer), so we could ease development. However, in a production setup Apache Kafka could be a good fit, because it can tolerate several faulty or disconnected nodes.

In scenarios with low trust among participants, BFT ordering services can be easily plugged thanks to Fabric's modular design. However, we believe that a CFT algorithm is enough for this use-case since a double-spend does not make sense here[3]. In addition, HL's endorse-order-validate transaction lifecycle offers a variety of mechanisms to prevent or detect misbehavior.

TABLE I
EXPERIMENTAL SETUP

| | |
|---|---|
| Number of organizations | 4 |
| Typical key + value size | 32 bytes |
| Number of channels | 1 |
| State database | LevelDB key-value store |
| Endorsement Policy | AND(Org1,Org2,Org3,Org4) |
| Ordering Service | SOLO |
| Block timeout | 100 ms |

### C. OpenOverlayRouter Software Router

In order to effectively perform access control, we took advantage of an open-source LISP implementation, Open Overlay Router (OOR [10]). We made a slight modification to its Tunnel Router mode: when it receives a Map Request packet, it queries Hyperledger with the source and destination endpoints, via an ad-hoc API. Hyperledger checks if the pair of (source, destination) is allowed to establish a connection and notifies OOR. If they can connect, OOR then responds to the source with a Map Reply message, otherwise takes no action. This way, unauthorized users do not receive a response in the form of a Map Reply and do not know where to connect. Of course, a production setup requires additional security mechanisms in the control plane, as mentioned in section III-C.

## V. EXPERIMENTAL EVALUATION

### A. Scenario

We set up an experimental scenario on a PC running Ubuntu 16.06 and a quad-core Intel i5 CPU 650 @ 3.20GHz. Table I summarizes HL parameters during the experiment. Thanks to the Docker containerization of HL, we could emulate 4 organizations, each with 2 peers, all in the same PC. We artificially generated around 1 million policies and users to evaluate the read latency, and added at most 15 endorsers to estimate the write latency.

### B. Results

We carried out several experiments on our implementation to characterize its performance and have an understanding of its scalability.

***Read latency:*** Fig. 5 presents the average query time for different number of elements in the chain. In this case the state DB was CouchDB[4]. We can see that it revolves around 40 ms regardless of the number of elements, because Couch DB is a key-value store (these type of databases present constant query latency). It should be noted that the query performs exact matches of pairs of source and destination IP addresses, and that future work should also support longest-prefix matching.

[2] They identify groups of users in order to simultaneously create policies for several users.

[3] n.b. in a CFT environment some attacks, such as censoring a transaction, may become feasible

[4] HL allows using both LevelDB and Couch DB as state DB, with similar performance.

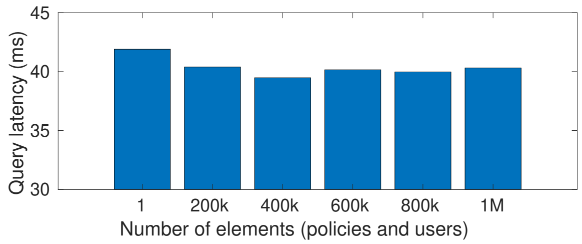

Fig. 5. Hyperledger CouchDB query latency.

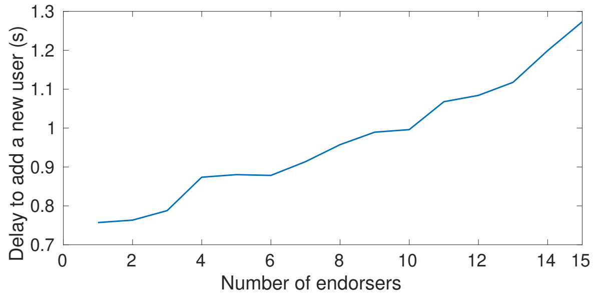

Fig. 6. Latency to add a new user for different number of endorsers.

***Write latency:*** Fig. 6 plots the time required to add a new user depending on the number of endorsers in the network. As we can see, the latency grows linearly with the number of endorsers, because each new endorser is an additional signature that the issuer has to verify (the current HL implementation makes this verification sequentially). This result is in line with in a recent benchmark of the HL platform [11].

***Chain size:*** We were also interested in the chain size, i.e. required storage. Figs. 7 and 8 show the total chain size depending on the number of transactions and endorsers, respectively. As expected, in both cases the size grows linearly with the number of transactions or endorsers (in the latter case because more endorsers mean more signatures per transaction). We can see that these situations require very modest storage. Thus, we can safely assume that scenarios with a considerable amount of participants (e.g. 1k endorsers would demand ∼25 GB) or a long transaction history (1M transactions take up ∼10 GB) can be easily supported.

***Network latency:*** Fig. 9 presents the query time CDF of a LISP control plane node (Map Server) storing 1k, 10k or 100k pairs of source, destination pairs. In other words, given a source node, how long does it take to find the authorized destination node(s). Since this test was performed in a local network, we consider the communication delay negligible.

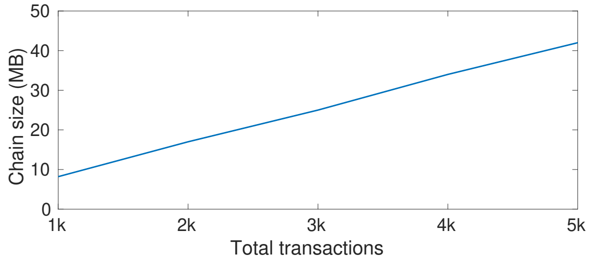

Fig. 7. Chain size vs. number of transactions, in a setup with four endorsers.

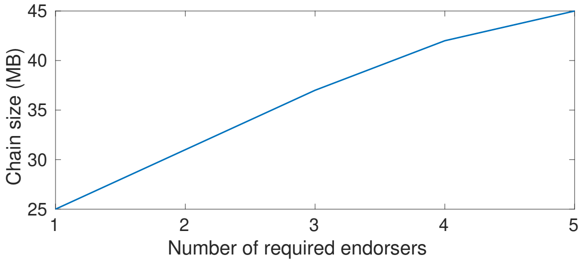

Fig. 8. Chain size depending on the number of required endorsers (5k transactions in the chain).

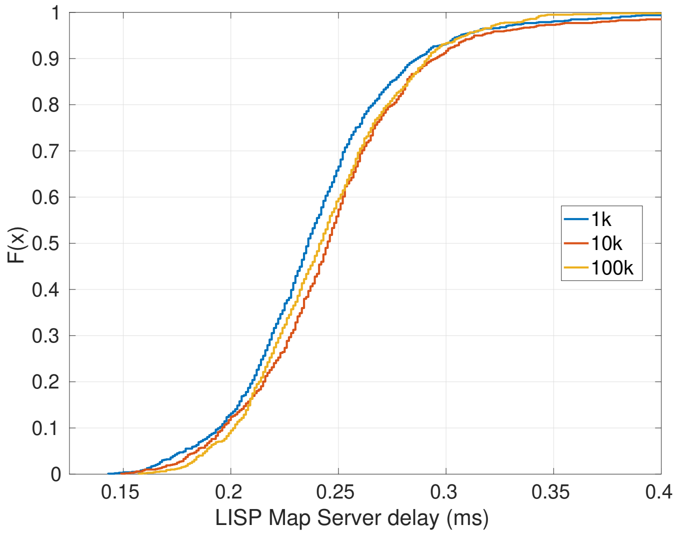

Fig. 9. CDF for LISP Map Server query delay for 1k, 10k or 100k pairs of source, destination nodes.

We can see that the majority of the queries are completed in less than 0.35 ms, roughly two orders of magnitude below the HL database delay. This is mainly due to two reasons: (i) the Map Server is implemented in C (whereas the queries in fig. 5 go through HL's Node.js API, CouchDB and back) and (ii) data is stored in a Patricia Trie, a tree optimized for prefix queries. In addition, the delay is independent of the number of elements thanks again to the Patricia Trie: the delay depends on the length of the elements (source endpoints, IP addresses in our implementation), not the number.

## C. Discussion

The previous results demonstrate that the proposed system can easily scale to meet the demands of a federation of several organizations. Table II outlines several metrics and their requirements in terms of scalability.

On one hand, both the read and network latencies can support high query rates. The read latency presents a constant response time regardless of the number of elements in the

TABLE II
SCALABILITY ANALYSIS

| | |
|---|---|
| Read latency (exact match) | constant |
| Write latency | linear w.r.t. number of org. |
| Chain size | linear w.r.t. number of transactions and endorsers |
| Network latency | linear w.r.t. identifier size + propagation delay |

chain (in case of exact matches), and the network server storing the access policies, linear with the length of identifiers.

On the other hand, the write latency can suffer if we have a large amount organizations in the same blockchain. However, it is not as critical as the read latency because we can tolerate a delay up to several of seconds when adding a new user.

Finally, the chain size obviously depends linearly on the number of transactions, but also on the number of endorsers. This last relationship puts an additional strain on scalability, because it affects both chain size and write latency. There is a tradeoff here between a small number of endorsers (small write latency and chain size, but more centralized trust in a narrow set of participants), and a large number of them (higher write latency and size but more distributed trust). Thus, special consideration should be put in the number of endorsers and the endorsement policy to achieve an equilibrium between a tolerable write latency and the expected number of endorsements.

## VI. RELATED WORK

There are already several proposals in the literature [12] that leverage blockchains for a wide range of network applications, such as mesh networks [13], IP addresses [14], etc.

The most closely related work to ours is [15], which implements Attribute-Based Access Control (ABAC) policies over the Bitcoin blockchain. It presents three main differences with respect to our work: (i) it focuses on access control for individual users, unlike our organization-based approach, (ii) it allows transferring access control rights between users, and (iii) does not consider using a private chain or a different consensus algorithm. Hadi [16] proposes a data distribution system in which the blockchain is the data persistence layer, but is also user-centric and more oriented towards data storage and messaging services rather than networking.

Finally, there is also a growing body of work on blockchain-based access control for IoT: [17] leverages a blockchain to store access permissions for IoT devices with a strong emphasis on key management and distribution. [18] also provides authentication, authorization and auditing for IoT but separates them in four independent blockchains, and is generic enough to support a wide range of access control models typical of IoT, while in this paper we concentrate on a specific language, GBP.

## VII. CONCLUSION

In this paper we have presented, implemented and evaluated an architecture to support access control in cross-domain communications. In order to reduce the burden on network administrators, the front-end builds on GBP, a well-known intent-driven language. A permissioned blockchain distributes network polices, and helps overcome drawbacks of conventional solutions while at the same time maintains the independence of each organization. Our experimental evaluation shows that this design can easily scale to -at least- tenths of organizations with modest storage requirements.

## ACKNOWLEDGMENT

This work has been supported by the Spanish MINECO under contract TEC2017-90034-C2-1-R (ALLIANCE) and the Catalan Institution for Research and Advanced Studies (ICREA).

## REFERENCES

[1] D. C. Verma, “Simplifying network administration using policy-based management,” *IEEE Network*, vol. 16, no. 2, pp. 20–26, March 2002.

[2] O. Foundation. (2017, November) Group-based policy for openstack whitepaper. [Online]. Available: https://wiki.openstack.org/w/images/a/aa/Group-BasedPolicyWhitePaper_v3.pdf

[3] A. Jøsang, “Pki trust models,” *Theory and Practice of Cryptography Solutions for Secure Information Systems*, p. 279, 2013.

[4] A. Slagell, R. Bonilla, and W. Yurcik, “A survey of pki components and scalability issues,” in *2006 IEEE International Performance Computing and Communications Conference*, April 2006, pp. 10 pp.–484.

[5] S. Liu, P. Viotti, C. Cachin, V. Quéma, and M. Vukolic, “Xft: Practical fault tolerance beyond crashes.” in *OSDI*, 2016, pp. 485–500.

[6] A. Bessani, J. Sousa, and E. E. P. Alchieri, “State machine replication for the masses with bft-smart,” in *2014 44th Annual IEEE/IFIP Intl. Conf. on Dependable Systems and Networks*, June 2014, pp. 355–362.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling Innovation in Campus Networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 69, 2008.

[8] P. L. Consortium. (2018, October) P4 runtime specification. [Online]. Available: https://p4.org/specs/

[9] L. F. Projects. (2018, June) Hyperledger fabric. [Online]. Available: https://www.hyperledger.org/projects/fabric

[10] A. Rodriguez-Natal, J. Paillisse, F. Coras, A. Lopez-Bresco, L. Jakab, M. Portoles-Comeras, P. Natarajan, V. Ermagan, D. Meyer, D. Farinacci *et al.*, “Programmable overlays via openoverlayrouter,” *IEEE Communications Magazine*, vol. 55, no. 6, pp. 32–38, 2017.

[11] P. Thakkar, S. Nathan, and B. Vishwanathan, “Performance benchmarking and optimizing hyperledger fabric blockchain platform,” *arXiv preprint arXiv:1805.11390*, 2018.

[12] N. Bozic, G. Pujolle, and S. Secci, “A tutorial on blockchain and applications to secure network control-planes,” in *2016 3rd Smart Cloud Networks Systems (SCNS)*, Dec 2016, pp. 1–8.

[13] M. Selimi, A. R. Kabbinale, A. Ali, L. Navarro, and A. Sathiaseelan, “Towards blockchain-enabled wireless mesh networks,” *arXiv preprint arXiv:1804.00561*, 2018.

[14] J. Paillisse, M. Ferriol, E. Garcia, H. Latif, C. Piris, A. Lopez, B. Kuerbis, A. Rodriguez-Natal, V. Ermagan, F. Maino, and A. Cabellos-Aparicio, “Ipchain: Securing ip prefix allocation and delegation with blockchain,” in *IEEE Intl. Conference on Blockchain*. IEEE, 2018.

[15] D. Di Francesco Maesa, P. Mori, and L. Ricci, “Blockchain based access control,” in *Distributed Applications and Interoperable Systems*, L. Y. Chen and H. P. Reiser, Eds. Cham: Springer International Publishing, 2017, pp. 206–220.

[16] S. H. Hashemi, F. Faghri, and R. H. Campbell, “Decentralized user-centric access control using pubsub over blockchain,” *arXiv preprint arXiv:1710.00110*, 2017.

[17] O. Alphand, M. Amoretti, T. Claeys, S. Dall’Asta, A. Duda, G. Ferrari, F. Rousseau, B. Tourancheau, L. Veltri, and F. Zanichelli, “Iotchain: A blockchain security architecture for the internet of things,” in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, April 2018, pp. 1–6.

[18] O. J. A. Pinno, A. R. A. Gregio, and L. C. E. D. Bona, “Controlchain: Blockchain as a central enabler for access control authorizations in the iot,” in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Dec 2017, pp. 1–6.