The Journal of Systems & Software 180 (2021) 111030

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

A novel blockchain protocol for selecting microservices providers and auditing contracts☆

Wilton Jaciel Loch [a], Guilherme Piêgas Koslovski [a,*], Maurício Aronne Pillon [a], Charles Christian Miers [a], Marcelo Pasin [b]

[a] Graduate Program in Applied Computing – Santa Catarina State University, Brazil
[b] University of Neuchâtel (UniNE) – Institut d'informatique, Switzerland

ARTICLE INFO

Article history:
Received 28 April 2021
Accepted 10 June 2021
Available online 20 June 2021

MSC:
00-01
99-00

Keywords:
Blockchain
Microservices
Selection
Audit
Protocol
SLA

ABSTRACT

Software architectures based on containers and microservices are often used to develop and manage large-scale distributed applications. Still, large vertical deployments spanning over multiple cloud and edge infrastructures are cumbersome to negotiate for, as each infrastructure provider is usually unique concerning prices, management strategies and Quality-of-Service (QoS) levels. In this scenario, Service Level Agreement (SLA) contracts are primarily crafted through pre-established templates and clients must trust providers to manage provisioned resources. The present paper proposes Dawn, a novel blockchain protocol for selecting microservice providers and auditing contracts. The protocol exploits the distributed and verifiable storage of a blockchain, as well as its decentralized consensus to enable contracts establishments in unreliable environments. Besides providing a formal definition of the protocol, this work discusses the possible threats to the correct operation of the network, originated by tenants and providers. We show that Dawn is secure under the evaluated terms, that it can efficiently help the contract establishment process as well as it guarantees a functional systematic way of auditing through monitoring. Finally, we studied both best and worst case scenarios regarding the number of issued messages, stored data volume and network traffic to execute Dawn with different numbers of clients and providers.

1. Introduction

Microservices are a form of software architecture in which the system is executed by the intercommunication of lightweight blocks implementing very specific functions. This approach, although considerably simple, is very well suited for constructing large-scale distributed systems which can be malleably allocated atop multiple providers. The joint provisioning of microservices on edge and cloud providers is a revolutionary approach to implement and deliver complex systems with Quality-of-Service (QoS) and Quality-of-Experience (QoE) requirements. Network function virtualization, data stream processing, adaptive and dynamic caches are contemporary examples of distributed applications simultaneously hosted by edge and cloud infrastructures. Although disruptive, the joint scenario is composed of multiple economically competitive actors. In fact, cloud and edge infrastructures differ on scale, availability of services, provisioning cost, and QoS management tools (Di Francesco et al., 2019; Tuli et al., 2019).

There is an inherent problem with edge and cloud-based service contracts (*e.g.*, Infrastructure as a Service (IaaS), Function as a Service (FaaS)) in which tenants are unaware of physical hardware details as well as orchestration, and resource sharing policies. Consequently, they must fully rely on information entirely collected and made available by service providers (Garcia Lopez et al., 2015; Rodrigues et al., 2019; Lins et al., 2018). Common cases of Service Level Agreement (SLA) breakdown caused by providers are related to inaccurate resource capacity provisioning, especially in terms of processing capacity, available memory, bandwidth, and response time. The reasons may change from inefficient scheduling policies of physical resources to intentional overbooking. On the other hand, tenants may act dishonestly, accusing providers of negligence in the service provision and thereby seeking some kind of marketing or legal advantage (Zhang et al., 2016; Shi et al., 2016). Therefore, in the face of a contractual dispute, it becomes a complex task to correctly audit, assess and judge which party involved in the contract was injured and which sought an unlawful benefit.

The present paper presents the design and the analysis of a novel blockchain protocol which seeks to guarantee transparency

☆ Editor: J.C. Duenas.

* Corresponding author.
E-mail addresses: wilton.loch@edu.udesc.br (W.J. Loch), guilherme.koslovski@udesc.br (G.P. Koslovski), mauricio.pillon@udesc.br (M.A. Pillon), charles.miers@udesc.br (C.C. Miers), marcelo.pasin@unine.ch (M. Pasin).

https://doi.org/10.1016/j.jss.2021.111030
0164-1212/

for transactions and requests between providers and tenants when allocating microservices, as well as distributed algorithms to verify complaints, reward participants who provide correct information and penalize those who do not. The centering of the proposal around blockchain is justified by its ability to foster a distributed environment with easy verification of the existence and correctness of data. This feature makes it possible to audit information on previous and ongoing SLA contracts between tenants and providers, serving as a starting point for the decision making algorithm. The decision procedure is centred on an asynchronous and distributed voting mechanism based on data collected independently by tenants and providers through a trustful monitoring mechanism. Inspired on commonly used e-commerce auction and shopping websites (*e.g.*, eBay, Mercado Libre) as well as ride-sharing and food delivery systems (*e.g.*, Uber, UberEats), a reputation-based mechanism is used to define the credibility of users. In short, voting is used to collective infer about SLA breaches that are likely to happen, and to recalculate the reputation accordingly. We further remark that the proposal does not simply employs existing blockchain technologies as a base on which additional algorithms are built on top. Instead we propose a novel form of blockchain network as a whole, whose both transactions and validation processes are specifically tailored to the end of providing a contracting and auditing platform for microservices. Thus, the behaviour and characteristics that we claim the network to have are intertwined on the very fabric of the proposal itself, and not achieved solely from pre-existing technologies harvest from specialized literature.

The main contributions of the present paper are three-fold: *(i)* the definition of a unified market for microservices provision to aid the contracts establishments achieved in a decentralized manner; *(ii)* a form of dynamically ranking the involved parties given by their service quality and network behaviour; and (iii) a practical auditing process to judge and repay possible SLA breakdowns.

This manuscript is organized as follows. Section 2 presents the scenario, actors and problem definition. Section 3 introduces the protocol, its transactions and the working of each component. Section 4 shows a reduced practical example of a Dawn, while Section 5 exposes a broad analysis of possible threats and how the protocol architecture deals with them. Section 6 presents an analytical modelling to investigate the theoretical limits of Dawn on issued messages and data volume. Section 7 discusses the related work and finally, Section 8 presents our considerations and future work.

2. Overview

When using classical microservice provisioning, the burden of service identification and contract establishment is taken by the client, mostly by contacting providers through conventional means and by sending the information needed to enact the service. This process must be executed by the client for each provider, asynchronously and individually. Negotiations are held with each provider, without an unified form of service contraction, let alone having forms of splitting partial requests among different parties. Fig. 1 depicts the contracting scenario.

Client $C1$ (Fig. 1) submits a microservice request to providers $P1$ and $P2$. A graph is used to represent the request, in which nodes denote the microservices, and edges indicate the communication requirements. Both providers, using distinct protocols offer proposals (QoS indicators and prices) to the client. Once the client selects a provider and confirms the service with a signed contract and SLA, the resource provisioning starts. Given a certain period of time, SLA violations may occur and in most cases the power resides exclusively in the hands of the providers. Clients have no way of challenging these violations and, in some cases, they cannot even prove it happened.

2.1. Main actors

The main actors in a microservices market are summarized in two groups: tenants and service providers.

Tenants are all participants in the microservices market who come to search for one or more providers which meet their price and QoS requirements, as well as reputation standards. A set of six main actions can be triggered by tenants in the protocol: *(i)* submit a properly formatted request for microservices provisioning, detailing all the requirements; *(ii)* extend an ongoing request to accept an already received proposal and to revive the request itself to the network; *(iii)* commit to a SLA contract with providers; *(iv)* call a voting event reporting SLA breakdowns; *(v)* vote on ongoing disputes originated by tenants or providers; and *(vi)* finalize an ongoing provisioned request.

The second group of actors are the *Service Providers*, including brokers and providers (cloud and edge). A simplification of two main actions was selected to represent this group: *(i)* to propose a provisioning offer fully or partially attending a tenant's request; and *(ii)* vote on dispute events originated by service providers and tenants. Each node on the network is personified by a user that belongs to one of these groups and that controls all the actions taken in relation to the chain and to other participants. The nodes are then individually maintained by each user which is either a real world customer, seeking for microservice provisioning services, or a service provider offering its products.

In regard to the blockchain network, any participant may be referred only as *User*, regardless of the original belonging group. Each user belongs to a single group and has an address that is constructed based on the public key, ideally through one way cryptographic processes like SHA256 or mixes of multiple such alternatives. We do not deepen the discussion regarding addressing as this is mostly an implementation directed issue, which is not our focus here. These properties are defined when the user enters the network and naturally cannot be further modified. In order to fulfil all of its possible actions, each microservice infrastructure broker that wants to take part in the network must have two users associated with, responsible for representing both perspectives (a user with tenant's profile to contract services and a second one with service provider's perspective to resell them).

Rather than proposing federating algorithms that would require changes in the service providers, we propose the broker user as an intermediary between tenants and service providers. This type of user leverages well established software design patterns to cope with the problem of incompatible provision forms. The complexity of having to deal with different protocols is isolated inside the broker, which in turn is used by many different tenants in a unified way. A similar non-federating layered model was proposed in past works (Veríssimo et al., 2012), further justified by the fact that service providers are often deliberately incompatible and chances are that they would remain that way.

2.2. Tenants and microservice providers reputation

Many different distributed or web-based services existing today keep track of reputation, for customers, providers or simply peers. Common examples applications are car-sharing, end-to-end sales and auctions, and marketplaces. Reputation has already been proposed as a means for selecting appropriated cloud providers (Comi et al., 2015) and to compose cloud federations (Mashayekhy et al., 2019).

In this work, reputation represents the efficiency and correctness of resource provisioning within a market of microservices. In other words, a provider with high reputation delivers the SLA-contracted capacity (*e.g.*, processing power, minimum bandwidth, maximum latency) throughout the provisioning time, with little performance fluctuation. Conversely, the user's reputation denotes the correctness and veracity of data provided to resolve disputes in the market.

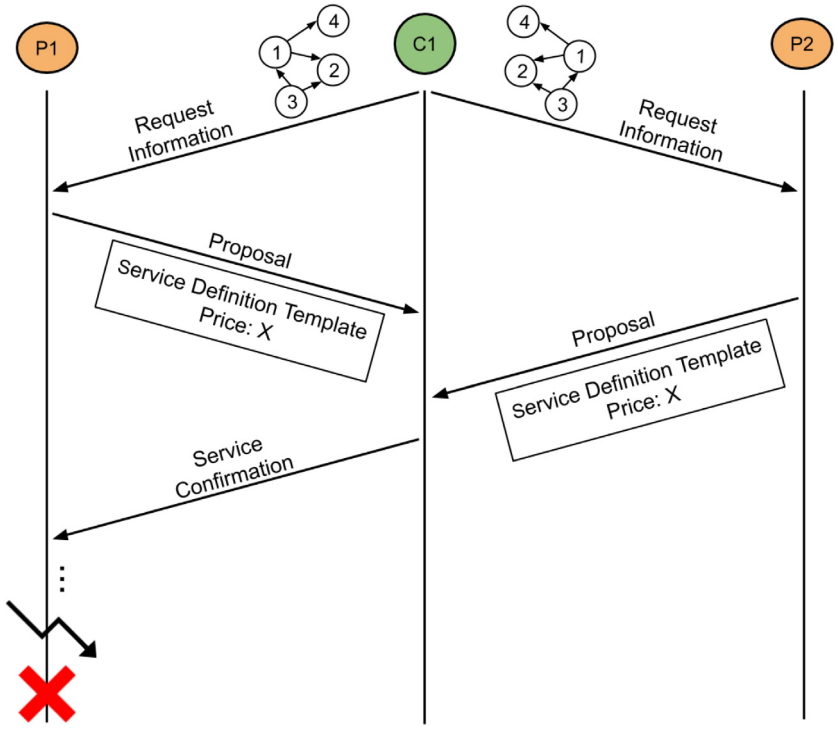


Fig. 1. A simplified scenario for contracting microservice providers (cloud and edge).

2.3. Problem statement

We place our work in an heterogeneous, competitive, and unreliable scenario composed of cloud and edge providers, tenants, and brokers. Multiple providers, architecturally positioned in the cloud or on the edges of the Internet, have different prices, strategies and qualities in their services. We assume that any user may misbehave to gain benefits. While providers may want to overbook requests to increase revenue, clients may disturb the market to reduce costs. We need a mechanism providing means to establish reliable contracts, assisting in the selection of providers and offering a protocol to audit them.

Currently, without any federating or brokering service, the distribution of the negotiation is not automated. Clients must individually contact multiple providers to retrieve their provision informations such as prices and policies, and to provide specific forms of QoS requirements. This form of negotiation, besides often demanding human interaction, represents a barrier to the division of a single request amongst multiple providers and to the aggregation of their disjoint resources in the deployment of a physically distributed, but functionally unified job. Exclusively hiring service brokers is a way of mitigating this problem, but it just shifts it from the tenants to the brokers. Another consequence is that there is no direct competition between providers towards the client's individual request, since each negotiation is individually implemented between pairs of tenants and providers.

Once a contract is established and the job deployed, the SLA requirements may be missed by the provider. These generally have less impact on a microservice architecture than on a monolithic counterpart (Gill and Buyya, 2018), however it can lead to monetary losses in the same manner, depending on the intensity and the number of affected units (Heorhiadi et al., 2016). Although SLA contracts are legal agreements which could be used to redeem the appropriate restitution when a SLA miss occurs, clients many times have no concrete way of proving that the miss in fact happened. Also, often the cost of lawsuits can be much higher than the actual monetary damage caused by the miss, leaving clients with no means to get repayment. Existing alternatives that explore the inverse way (Clack et al., 2016), namely attaching express legally-enforceable rules from legal documents over smart contracts still present several challenges before being useable in real world scenarios.

Clients, on the other hand, can also exhibit dishonest behaviour by forging accusations and monitoring data, mostly envisioning rewards via restitution or cost reduction, causing market disturbance. This turns the process of judging eventual breaks into a complex task since none of the involved parties can be fully trusted. As a result of all of the conditions discussed, the present scenario manifests itself as an untrustful environment in which distinct actors must interact to achieve a common goal, without any guarantees of honest behaviour from each other.

3. Protocol

The protocol, termed Dawn, is loosely based on the concept of transactions from the Bitcoin concept (Nakamoto, 2009). In the original Bitcoin implementation, they serve as a ledger that records all users coin transfers, while in Dawn context they allow a sense of complete and incomplete states. Unreferenced transactions – whose hash have not been listed as source in specific subsequent transactions – represent processes that have not yet been completed, such as proposals for a service or openings for votes, while responding through a new transaction to an anterior one indicates the end of a previously opened process.

Most of the information exchanged between users and consequently written in the ledger refers purely to contractual details and monitoring data. Only a small fraction of the total transactions refers to transfers of actual values (reputation, described in Section 3.1). Dawn's blockchain data is public, so even users who have no financial correlation with the system or formally interact with it can anonymously view transaction and reputation data.

Fig. 2 is used along this paper to illustrate the protocol, transactions, negotiation, and auditing phases. The clients group is

represented by C while the providers group is given by P, respectively. As mentioned before, the broker user employs both client and provider forms of interaction, thus this group of users is actually diluted into the other two and is not individually presented here. Following the group identifier letter, there is also an individual numeric identifier used to specify different users inside the same group of actors.

The protocol combines publicly available data about providers with reputation scores calculated by participating users. Dawn covers the microservices lifecycle from the initial specification stages, through the establishment of contracts, to the audit and voting phases. In this sense, the Dawn design is presented as follows. Initially, the base algorithm to account users reputation is described in Section 3.1, while the common attributes of transactions and the certificate publishing are presented in Section 3.2. Latter, the protocol transactions related to contract negotiations are presented (Section 3.3), followed by the SLA monitoring and auditing phase (Section 3.4).

3.1. Users reputation

The reputation of users is a key item of the Dawn protocol, as it is the product of both the successful elaboration of contracts and the performance of effective audits on them. As there is no possibility for a user to deliberately transfer reputation points to another, a specific transaction for this is unnecessary. The users reputations are constructed through information contained in other transactions (*e.g.*, requests, voting). The representation adopted by Dawn is a value that varies from 0 to 100, whose limits are related to the minimum and maximum possible reputation, respectively. The alternative of using reputation limits, especially the superior one, is chosen to prevent honest providers and brokers from accumulating very high reputation values that new ones would not be able to reach, creating a biased, unfair and not easily modifiable scenario for selling services.

All users who join the network receive a reputation value equivalent to 50 points, since it is not possible to judge *a priori* whether this user has a tendency towards honest behaviour or not, and hence higher or lower value assignments would be unfair. Latter, a user gains reputation through votes and contract establishments, and loses only through the former. The reputation reward received from contract establishments is constant. According to this rationale, nothing would prevent the sum of the amounts received by a user to exceed the maximum of the allowed reputation. To tackle this, Dawn proposes the account of reputation for a given user based on a simple temporal reconstruction of that user's transaction history on the network and not on actual values written on the ledger. Therefore, the reputation is not a token, but actually an abstract measurement of all processes that have already happened throughout that user's participation.

Algorithm 1 represents the pseudocode for building an user's reputation, in which r and v denote the reputation and the default contract reward, respectively. CCT is the set of contract commit transactions already concluded by a user, while RT is the set that contains all reputation transfers designated to user u. Functions *get_contracts*() and *get_transfers*() search transactions on the network (first locally, when available), *sort_by_timestamp*() sorts a transaction set by their timestamps, older first. Finally, $get\_value(T_i, u)$ recovers the reputation transferred to user u on T_i, if $T_i \in RT$.

Input: public address of user u
Output: reputation value r

```
r = initial_reputation
v = default_reward
CCT = get_contracts(u)
RT = get_transfers(u)
CCT = sort_by_timestamp(CCT)
RT = sort_by_timestamp(RT)
T = sort_by_timestamp(CCT ∪ RT)
for T_i ∈ T do
    if T_i ∈ CCT then
        r = min(100, r + v)
    else
        r = min(100, r + value(T_i, u))
    end
end
return r
```

Algorithm 1: Building the network user's reputation.

Table 1
Common transactions attributes.

Attribute	Description	Bytes
Version	Transaction version	4
Address	Creator user address	33
Signature	Used to ensure that the owner of the address is the true creator of the transaction	70–72
Type	Used to differentiate the transactions	1

3.2. Common transactions attributes and certificate publishing

The protocol is composed of 9 different transaction types: Certificate Publishing (CP), Microservice Request (MR), Request Extension (RE), Microservice Proposal (MP), Contract Commit (CC), Voting Call (VC), Vote and Bet (VB), Verdict and Transfers (VT) and Contract Finalization (CF).

In Dawn, there is a group of attributes which is common for all transactions, mainly used to identify and verify the author's and the transaction's authenticity, as summarized by Table 1. The version is used to enable the protocol's evolution, while the address field indicates the public address of the transaction's creator. The signature is generated by the private key corresponding to the user's address through asymmetric encryption. In addition, the signature comprises all the data of the transaction – including specific fields of the different types further described – and ensures its authenticity, proving that the transaction's creator is the same as the user indicated in the address field. Finally, the type of the transaction indicates how the rest of the following variable data should be interpreted.

In addition, Dawn proposes a Certificate Publishing (CP) transaction to disseminate to network participants the existence of a certificate belonging to a given user (who created the transaction). Certificates must be granted by some external and trustful entity to all users who wish to join as microservice providers (cloud and edge). The certification is essential to guarantee that a user is a member of the provider's group and consequently can offer and provision microservices, thus preventing the creation of fake provider accounts.

3.3. Contract negotiations

Microservices negotiation involves the Service Request, Microservice Proposal, Request Extension, and Contract Commit transactions. In order to avoid market speculation and network overload, a limit on the number of active contract negotiations for a specific user is set.

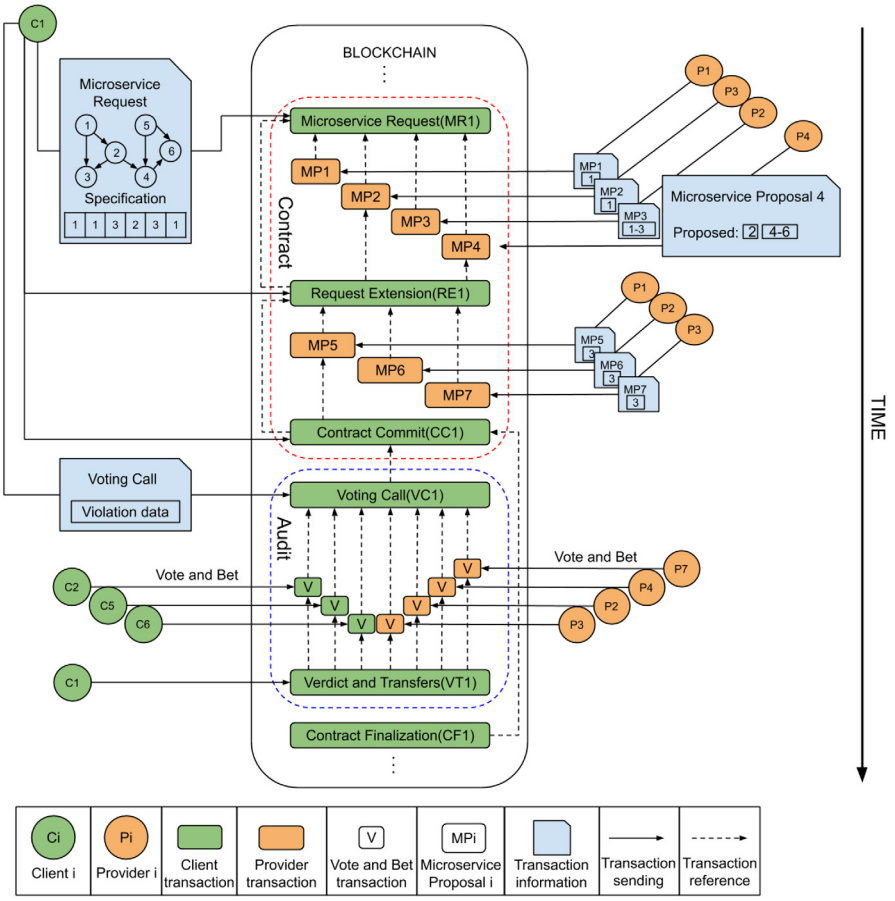


Fig. 2. Execution scenario of Dawn protocol. The data flow to and from the blockchain is orchestrated by the transactions (described on Sections 3.3 and 3.4).

Table 2
Microservice Request transaction attributes.

Attribute	Description	Bytes
Data size	Size in bytes used to carry data	4
Data	Service description and requirements	Variable

3.3.1. Microservice request

The Microservice Request (MR) describes a set of microservices represented by a directed graph (as depicted by Fig. 2), where nodes represent the microservices and the edges denote the network dependencies among them. In addition, a vector is given for each node and edge in the graph denoting the capacity and Quality-of-Service (QoS) requirements (for instance, number of replicas, processing capacity, or minimum bandwidth). Specific languages or models for representing virtual resources and communication requirements are not specified by Dawn, being extensible in this sense. Examples are CloudFormation from Amazon (AWS CloudFormation, 2020), Heat for OpenStack (OpenStack, 2020), and virtual resources modelling languages (Koslovski et al., 2009). Table 2 summarizes the attributes for a MR transaction (in addition to Table 1 data). The data size field indicates the size in bytes to carry the service request (Data attribute).

3.3.2. Microservice proposal

Once a MR is stored in the blockchain, MPs can be submitted by providers and brokers. A MP fully or partially fulfils the requirements for a microservices request. Table 3 presents the attributes for a MP transaction. CP hash references a previous Certificate Publishing transaction that naturally must carry a valid certificate for the creator user. MR/RE hash refers to the request currently being addressed. In turn, Seed data is required to perform a possible upcoming voting transaction (detailed in Section 3.4.4). Pricing policies are based on a predefined indexed list (*e.g.*, per minute, per hour, *etc.*), while the price assumes floating point values (*e.g.*, IEEE 754). Finally, a service provider (cloud or edge) can offer only a subset of microservices requested.

3.3.3. Request extension

Accepting a MP is a client decision guided by private economical and QoS indicators. Moreover, a MP can partially fulfil a request. Both aspects motivate the definition of a transaction for Request Extension (RE). Clients can create a RE to inform which of the proposals made so far have been chosen for the request, indicated by referencing the MP transactions. After the publish of a RE transaction, the effect generated in the network is the revival of the original request, which happens due to the propagation of the new transaction, bringing back the attention of the providers. Consequently, providers can reevaluate the request considering the evolution of internal data centre load and economical objectives to then possibly send new MPs addressing the remaining microservices.

Table 4 presents the RE attributes. MR hash refers to the original request, while details of the accepted proposals are carried by MPs hashes. For that reason, new MPs must retrieve the original MR and analyse the MPs already submitted and compromised. A new offer must consider only remaining microservices.

3.3.4. Contract commit and contract finalization

The commit of a contract negotiation must reference the remaining MPs from the last RE or, in its absence, the original

Table 3
Microservice Proposal transaction attributes.

Attribute	Description	Bytes
CP hash	Hash referencing the provider's certificate	32
MR/RE hash	Hash pointing for the corresponding request	32
Encrypted seed	A random encrypted value defined by the service provider	32
Price policy	An index on a predefined pricing policy list	1
Price	The total price to provision the request	4
Number of microservices	Identifies if a request is fully or partially provisioned	2
Microservices	Identifiers to all microservices that will be provisioned defining the QoS SLA	Variable

Table 4
Request Extension transaction attributes.

Attribute	Description	Bytes
MR hash	Hash to point for the corresponding original request	32
Accepted MPs	Indicates the number of MPs accepted	2
MPs hashes	Pointers to all MPs transactions accepted	Variable

MR. This transaction seals the contract between clients and all providers and their MPs selected in the negotiation so far. Once a commit is identified in the blockchain, new microservice proposals for the corresponding request or its extension are not accepted. The effective service provisioning that follows the contract, and its deployment details are not addressed by Dawn. In summary, the contract commit makes the mapping between MR and RE to MPs (Table 5). Finally, committed contracts are atomic entities, which means that all the microservices listed in the original request must be served by a MP for a contract commit transaction to be accepted.

The CF transaction, as its name suggests, exists to allow clients to finish (terminate) their ongoing active contracts. Once again, as Dawn does not deal with service deployment or provision itself, the contracted provider and the client must take the necessary actions to finish it. This transaction can only be sent if there is no existing voting calls for the respective contract, and as it acts mostly as a collective notification of an ending contract, the only individual attribute necessary is the 32 bytes hash of the respective Contract Commit.

3.4. Auditing contracts

The audit phase (blue traced rectangle on Fig. 2) occurs when a user suspects that the QoS indicators are not being respected as agreed. The procedure is based on two fundamental pillars, the first being the SLA contract and its data stored on the blockchain, and the second is a set of monitoring data collected by clients and providers.

Dawn is based on trustful monitoring data to audit a contract (detailed in Section 3.4.1). The monitoring of microservice providers is external and parallel to blockchain management. Data collected is kept privately and only a summary is sent when needed. These data, although having a theoretical owner and eventually being shared, should not be interpreted by any actor other than each user's digital wallet – the term *digital wallet* is loosely employed and it designates the system which allows users to interact with the network. Thus, even the monitoring module owner – user that controls it – cannot understand the data that it produces or receives, and the empirical coherence of these only exists in the context of the auditing phase.

A major process performed on audit phase is the transfer of reputation between users. This phenomenon happens because all users who want to participate in the voting process (described in Section 3.4.4) must bet a part of their reputations, which means that voters are betting on the accuracy and precision of their own monitoring data. Although a minimum amount is required for betting, an upper-bound limit is not defined by the protocol, so it is up to the user to decide how much they want to bet. Consequently, larger bets confer high prizes in case of assertiveness.

The voting process defined by Dawn is based on the concept of focal points, which suggests that in coordination games players tend to take specific common actions if they believe that the rest of the participants will perform in similar ways. Thus, voting users tend to act honestly if they believe that other participants who send votes will do so as well, given that deviations from the collective good sense are punished by the protocol.

3.4.1. Monitoring module

First and foremost, it is essential to credit the development and management of such monitoring module to a trustful entity, responsible for registering users on the network before the submission of any transaction. In this sense, the monitoring module is an appendix to each user's digital wallet, and must be allocated to the infrastructure contracted by the user to collect data. Periodically, the module executes a set of stress processes to acquire performance indicators, which are compared to the QoS values defined on SLA. The monitoring frequency is upfront defined, and all monitoring modules operate in a standalone fashion and roughly at the same frequency. At first glance, the comparison of monitoring data performed on different devices without clock synchronization may seem inconsistent, but with relatively short periods of time and presumably similar monitoring contexts it is unlikely that SLA violations patterns would not be detected in multiple distributed modules.

The data collected by the monitoring module is stored locally in the contracted infrastructure and are only sent to the digital wallet upon an explicit request. The monitoring module has a pair of private and public asymmetric keys, the private key being the same for all modules of all users (stored somewhere in the source code or built secretly within a system of cryptographic enclaves Vaucher et al., 2018). The purpose of the monitoring module's key pair is to ensure the confidentiality of the data it stores. Thus, even if the data is discovered or copied from the disk, the attacker would lack the private key, which is known only by the monitoring module (during execution time). All the collected data remain stored for a fixed period of time independently of whether it has proof of some SLA breakdown or not. Each individual test performed in the infrastructure has its own entry on the database alongside with a unique identifier aimed to prevent reuse of information on votings. The module sends binary alerts to the user when breakdowns are detected. Upon data request for a vote, the module does not simply send the stored encrypted data, which would prevent any user from ever reading the content. Instead, it translates the information to a newly created voting key that guarantees that the data will be secret throughout the audit phase but readable after it ends – in order to enable the verdict creation. This key pair acts like a session key and must be different for every voting process.

Table 5
Contract Commit transaction attributes.

Attribute	Description	Bytes
MR/RE hash	Hash to point for the corresponding request or extension	32
Accepted MPs	Indicates the number of MPs accepted	2
MPs hashes	Pointers to all newly accepted MPs	Variable

3.4.2. Voting keys

Since the distribution of rewards to voters is done through the evaluation of the majority's opinion, the disclosure of voting data should be prevented to avoid the creation of fake votes that place their creators in a point of greater reputation rewards. The voting keys, used to address this issue, must be equal to ensure the consensus over the clear data of all the different users involved. However, since the monitoring modules operate independently and have no communication with each other, the key creation must be synchronized through other means.

Dawn uses equal random seeds to all the participants, which must be fed to the module when a data requisition is made and are used for the keys' setup. There are two necessary seeds to be informed and they determine what could be considered the language of a specific voting process. Then, every user who has meaningful data to that context can translate it through the module and join the voting. Both seeds are 32 bytes random values encrypted by the module's public key. The first one must be generated by the service provider and sent along with the MP transaction, while the second seed must be created by the accusing user and sent in a VC transaction (detailed in Section 3.4.4).

Inside the module, both values are decrypted, concatenated and then again encrypted by the module's private key to create a random, unique and not replicable value, which is then used as the private voting key (the term encrypted is utilized here as a form of shuffling in the original value through the private key, any secure encryption method that relies on a vector of bits could be used). The public key is created through the specific process of the chosen asymmetric encryption system, whose selection is considered an implementation aspect.

3.4.3. Voting process

The voting process, despite mostly focused on the audit phase, has some important steps that happen also in the contract phase. The complete process of a vote is displayed in Fig. 3. Initially, given a certain time in the blockchain, client $C1$ decides to send a MR transaction to the network. This action, although not depicted in the figure, allows provider $P1$ to send a MP, which besides the regular contractual information, comprises an encrypted seed destined to the voting process. The client then, commits the contract and accept $P1$'s service through a CC transaction (identified as $CC1$).

After a service confirmation and the access being granted to the microservices, the client's digital wallet inserts a monitoring module instance inside the infrastructure where the service provisioning is happening. As time advances, at each monitoring period, the module runs a test in the infrastructure, aiming to verify the correctness of service availability and QoS indicators. After each test, a new encrypted data record is written to the local storage. Executing a given test in time and noticing a SLA breakdown, the module sends a notification to the $C1$'s digital wallet informing the occurrence. $C1$ then, aiming the restitution from the break, should start a vote but before, the translation of the monitoring data to a readable form at the end of the process is necessary. In order to achieve this, the user searches the chain to retrieve the encrypted seed from the provider and also creates its own, both used for the voting keys construction.

The request for the module data retrieval is then made using both the provider's and the client's encrypted seeds. In addition, the client's clear seed is appended to the message. The module, receiving the requisition, decrypts both seeds and compares them individually with the clear appended value. In case of a match in one of the comparisons, the module assumes that for knowing the original value, the requesting user intends to start a voting instead of creating a vote — once voter users will not have access to the clear value of any seed.

The module builds the voting keys through the process described in Section 3.4.2. The stored data are translated from a version encrypted by the module's public key to a encryption through the voting private key. A hash is created for the translated data, signed by the module's private key and appended to the response message. The module also appends the voting public key capable of decrypting the information, since the requester is the accusing user, and finally sends the whole set back as a response.

At last, $C1$ receiving the data removes the voting public key and creates a new Voting Call transaction (identified as $VC1$ and detailed in Section 3.4.4) so that the network can contribute with new data, providing means to properly judge the break.

For the creation of votes to a voting call the process is similar. $C2$ then, possessing a monitoring module on an equivalent infrastructure of the same provider and noticing the publication of the previous $C1$'s $VC1$ transaction, can send a vote. In this case, there is not necessarily an alert issued by the module and now this user must retrieve both the provider's and the client's encrypted seeds from the blockchain. As $C2$ does not – and is not meant to – know any of the clear seed values, his request to the module is created solely with the two fetched encrypted seeds. The module receiving this request assumes that as the requesting user does not know any clear value, the data requisition must be for a vote. All the data translation, hashing and signing occur in the same way. However, the response set will not contain the voting public key once the requesting user is not one of the parties allowed to possess it. The voting user can then use the received data to create his vote and send it to the network.

After the voting period is closed and to finish the process, $C1$ sends the $VT1$ transaction containing the verdict and the key needed for the decryption. The same transaction could have been sent by the accused provider in other cases.

3.4.4. Voting call and votes

The start of the audit phase is given by the creation of a Voting Call (VC) transaction. It reveals to the network that an user identified a SLA breakdown and wants repayment for the damage through the network judgement. This transaction references the commit of the broken contract and informs specifically which microservice has been neglected, the violated QoS aspect and the time period of the following proof data, which also must be appended. In case of multiple violations on the same contract, multiple VCs should be issued to deal with different individual aspects.

Alongside with the breakdown information, a bet value is informed. It serves as a warranty of the supposed accusation veracity and dictates the minimum amount that should be alienated by any other voting user's bet. The value sent by any voting user as a bet has no maximum limit and therefore can be as much as he deems interesting, being that in case of correctness, greater bets also receive greater cuts of the remaining final reputation value.

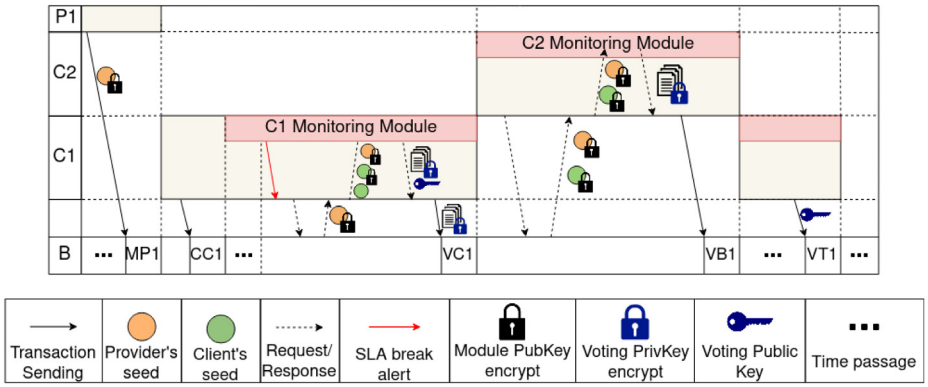


Fig. 3. Detailed voting process.

Voting calls have yet another important field that is the locktime. This value indicates for how many subsequent blocks relative to the current one (*i.e.*, the block where the transaction has been written) new votes will be accepted by the network. In this sense, once the VC transaction is committed in a block x, all valid votes to that call can be accepted, and therefore written in the chain, if the current block height is less or equal than $x + locktime$. Practically then, this field expresses for how long the voting process will be open to new votes. Table 6 shows all the VC transaction fields.

The next transaction type is the Vote and Bet (VB), which can be sent by any user with meaningful data for a voting context. The data appended to the vote transaction must have an equal time period (amount of tests) as the one informed by the accusing user. All votes have the same weight on the result, independent from which group the voting user is in. Table 7 indicates all of the vote transaction fields.

3.4.5. Verdict and transfer of reputation

The last part of the audit phase and hence of a full Dawn cycle is the construction of the verdict about a contract violation and the change in the reputation points of the involved users. After the voting period is over and a certain amount of votes were received, it is possible to calculate through the supplied data the probability of a SLA breakdown. The individual data of each user is used in conjunction to create a verdict that summarizes the concentric and inductive view of the network about the accused break. A statistical inference based on the median and standard deviation is used to judge who is the wrong participant. If the provider is considered guilty, his minimum bet value is taken and divided among the accuser and voting users, otherwise the client loses his/her bet. In the latter case, the value is divided only among the voting users and not with the accused provider for security reasons.

As certain users which did not provided assertive or collective similar data gain less reputation rewards, the ones who did receive more. This redistribution is based on the median, so the distance of the individual probability to it determines how much a user wins or loses. This transaction must be created either by the accusing client or the accused provider. If both send, the first one is committed — given it is correct. Alongside the transaction, also must be sent the voting public key that enables the data visualization and the verdict verification. There is a time limit to send this transaction relative to the VC's locktime.

In order to verify a sent verdict it is only necessary to retrieve the public key and all the listed votes, decrypt the data and mimic the verdict execution. If the distribution result obtained is equal to the one written to the transaction, then it is considered honest and if the other fields are valid, it can be relayed. The Verdict and Transfers transaction attributes are listed in Table 8.

The reputation transfer vector indicates a value of addition or subtraction on the total reputation of each user, in the order of vote reference presented in the fourth field (hashes of the votes). There is no maximum value that can be received, but the maximum loss of a given user is naturally his own bet value.

3.4.6. Verdict calculation

In order to distribute the reputation values among the users, there must be a way to calculate the division based on the users data. This section proposes one possible way of achieving this purpose. It is easy to see that Dawn can be extended to more elaborated and specific proposals (like employing Analysis of Variance (ANOVA) statistical models) without compromising the protocol mechanism.

Once all votes are gathered and verified, the violation data from each vote is extracted, decrypted and individually sorted in order of supplied percentage, so most severe breaks – when there is a low percentage of the accorded provision level – appear first, while the least intense appear in the last positions. This form of sorting can be applied over the dataset because the intensity and amount of breaks are analysed in the set as a whole, and not specifically on the order that these appear. The data dissimilarity is then calculated between each of the votes and the accusation. This process is done firstly by extracting all the break records from the data, while records that do not represent breaks are the ones with supplied percentage over a certain threshold.

Being q_a and q_{vi} the quantity of break records in the accusation and in a vote respectively, two distinct forms of dissimilarity calculation (D_1 and D_2 from Eqs. (1) and (2), respectively) are used to form the final result, in which A is the accusation dataset, and V_i is the dataset of vote i. A_j and V_{ik} are records from the respective sets and D_1 and D_2 are the partial dissimilarity measures. If eventually $q_a > q_{vi}$, then naturally for D_2, records from V_{ik} that do not represent breaks will be taken into the calculation.

$$D_1(A, V_i, q_a, q_{vi}) = \sum_{j=1}^{q_a} \sum_{k=1}^{q_{vi}} \sqrt{(A_j - V_{ik})^2} \tag{1}$$

$$D_2(A, V_i, q_a, q_{vi}) = \sum_{j=1}^{q_a} \sum_{k=1}^{q_a} \sqrt{(A_j - V_{ik})^2} \tag{2}$$

$$D_f(A, V_i) = \frac{D_1(A, V_i, q_a, q_{vi}) + D_2(A, V_i, q_a, q_{vi})}{2} \tag{3}$$

Multiple dissimilarity measures are used in this context because of the distinct ways in which they work. The first, D_1, calculates the dissimilarity of all the breaks in one set with all the breaks in the other. It is interesting as it does not restrict itself

Table 6
Voting Call transaction attributes.

Attribute	Description	Bytes
CC hash	Hash to point for the corresponding Contract Commit	32
Encrypted seed	Encrypted seed to be used in the voting process	32
MP index	Accused MP index	2
Microservice index	Indicates the Microservice/infrastructure where the breakdown was detected	2
Break aspect	Indicates the physical broken aspect	1
Time period	The amount of individual tests present in the data	2
Locktime	Indicates the voting open time in blocks	1
Minimum bet	Contains the value betted by the user which defines the obligatory minimum bet	2
Signature size	The size of the following signature	1
Signature	Module's data signature	70–72
Data size	Indicates the size of the break proof data in bytes	3
Data	Encrypted proof data	Variable

Table 7
Vote transaction attributes.

Attribute	Description	Bytes
VC hash	Hash to point for the corresponding Voting Call	32
CC hash	Hash to point for the voting user's Contract Commit	32
Bet	Amount of reputation points betted by the user	2
Signature size	Size of the module's signature	1
Signature	Module's signature of the data hash	70–72
Data size	Indicates the size of the following proof data	3
Encrypted data	Contains the data proofing the vote and representing the vote itself	Variable

Table 8
Verdict and transfers transaction attributes.

Attribute	Description	Bytes
VC hash	Hash to point for the corresponding Voting Call	32
Voting public key	Indicates what is the public key of the current voting	33
Amount of votes	Contains the amount of votes received in the voting	2
Hashes of the votes	Hashes of all the Vote and Bet transactions received	Variable
Transfers	Vector containing the reputation change for each involved user	Variable

to the amount of breaks pointed by the accusation, taking into account the different numbers of occurrences informed by both participants. On the other hand, since in an all-to-all comparison the summed values scale linearly with the amount of elements, it may overweight certain measures just by the difference in the break quantity. Tackling this problem, the second measure D_2 is used due to its characteristic of just comparing the number of occurrences pointed by the accusing user. Thus, in this form the intensity of the individual breaks have a greater importance over break amounts. The two forms are then combined through a simple average of the resulting values (Eq. (3)), assuring the dissimilarity rises as the difference in the amounts and intensity of the breaks rises as well.

The individual dissimilarity values are calculated for each vote towards the accusation and the median of the results is considered to be the consensual view of the dissimilarity. The median is used instead of the mean since the latter is highly susceptible to big alterations from outliers, which is not interesting for the present scenario. In order to generate the verdict, the final median is compared to a predefined α value of dissimilarity, and if it is lower then the user is considered victorious, otherwise the provider is the winner. This α value can be defined empirically or through analyses of the data itself, assigning the correct threshold based on the votes and accusation relation.

Aiming to create the reputation distributions, the considered aspects are how much that user has betted and how close his results are to the dissimilarity median. The main reward division comes from the loser user. However, the individual quotient is pondered in a way that a user loses part of what he/she should receive as reward according to the lack of precision in his data and receives a fraction of what other users have lost due to the same reason. The loss inflicted over an user's reward is proportional to the percentage represented by his data deviance in relation to the sum of all voting users deviance. The reward fraction that a user receives from other users losses is proportional to the percentage of his bet value in relation to the sum of all bets from voting users. Hence and ideally, if all the users converge, they equally receive proportionally to the amount they invested, otherwise the furthest from the overall consensus the less a user earns.

As the accusing user has no value of dissimilarity associated with his data, the earnings are just the fractions of other's losses and he does not receive a share of the loser reputation division. This restriction is set to balance the fact that he cannot lose reputation due to imprecision. Therefore, being D_f the final dissimilarity between the accusation and a vote; $\tilde{d}$ the median of all the dissimilarities; D_t the sum of the deviances from the individual dissimilarities to the median; u the amount of voting users; v the amount of votes; b_i the bet of an user i; b_p the bet of the loser user equivalent to the minimum bet and t the total reputation of the voting, the distributions are given by Eqs. (4) to (8).

$$D_t = \sum_{i=1}^{v} D_f(A, V_i) - \tilde{d} \tag{4}$$

$$L_i = \left(b_i + \frac{b_i b_p}{t - 2b_p}\right)\left(\frac{D_f(A, V_i) - \tilde{d}}{D_t}\right) \tag{5}$$

$$TR_i = b_i + \frac{b_i b_p}{t - 2b_p} - L_i \tag{6}$$

$$R_i = TR_i + \sum_{j=0, i\neq j}^{u} \frac{L_j b_i}{(t - b_p - b_j)} \tag{7}$$

$$R_a = b_a + \sum_{i=0}^{u} \frac{L_j b_a}{(t - b_p - b_j)} \tag{8}$$

The reputation loss of a user given by its deviance is represented by L_i, while TR_i is the temporary reputation of a user, after the division of the reward and the discount of its own loss.

R_i is the final reputation of a user; R_a is the reputation of the accusing user, in case of his win. In case the provider wins the voting, the R_a calculation is not performed and the b_p operand inside the R_i sum must be $2b_p$, as the provider is not a part of the reward divider. The value is given in the form of a new reputation value, that must be transformed into a reputation variation by $R_i - b_i$ to be put in the VT transaction.

This is an initial and basic form of reputation distribution, which could be hardened or softened depending on specific scenario demands. Finally, as Dawn presents itself in a very modular way, it can actually work independently of the verdict calculation form used.

4. Use case

Fig. 4 presents a small-scale example of Dawn protocol based on the execution scenario from Fig. 2. A client $C1$ submits a microservices request through transaction $MR1$ detailing all processing and communication requirements. Specifically, the client requests for 1 vCPU (2.5 GHz), 4 GB RAM, 64 GB storage, network bandwidth of 15 Mbps, maximum latency of 150 ms and 99.997% availability. The client sends its request to the network and once it has been inserted in the blockchain, providers can send their proposals. In this example, providers $P1$ and $P2$ send MPs that meet the request's requirements. The first proposes to supply only microservice 1, while the second proposes to provision the entire request. Details on pricing policies were omitted as they do not interfere with the protocol operation, and are just used to support client's decisions. Following, $C1$ selects the $MP2$ (from provider $P2$) which includes all of the microservices requested. In this example, a RE is unnecessary. Finally, a Contract Commit (CC) transaction is created by the client referencing $MP2$.

In addition, during the contract negotiation and establishment described so far, providers $P1$, $P2$, and a third provider $P3$ already perform the monitoring of QoS metrics using the monitoring mechanism described in Section 3.4.1.

Eventually, $C1$ receives an alert from its monitoring module stating that a QoS indicator was violated. After collecting the data which supported the alert, $C1$ creates a voting call transaction. In this transaction, $C1$ specifies the contract that is under analysis, the encrypted seed ($49a7b83c8$), the $MP2$, the specific microservice and the monitoring period used to compose a dataset of 6 entries. The minimum bet that must be paid to participate in the voting session is also defined in the transaction, as well as the locktime to indicate how long the transaction can be referenced (set to 2 blocks after the corresponding one). Votes sent after the locktime are rejected.

Once $C1$ voting call is registered on blockchain, participants who have monitoring data about the defendant provider can submit their votes on whether the accusation is true or not. In the example of Fig. 4, $P1$, $P3$, and a customer $C2$ – who has an active contract with $P2$ – decide to send votes. In the vote of $P3$, which is taken as an example, the minimum bet value was used, and levels of violation similar to those of the $C1$ were found in the last 6 monitoring events.

As soon as the voting period closes, the accusing client or the accused provider can send a verdict to finalize the process. In this example, it is assumed that $P3$, $C2$, and $P1$ bet only the minimum value and have similar data, which attest to the veracity of the $C1$ accusation and therefore consider $P2$ as guilty. The accusing client sends the VT transaction and informs that the provider $P2$ was defined as guilty and all voting participants ($C1$, $C2$, $P1$ and $P3$) receive in addition to the initial bet, the proportionally shared bet of $P2$. Thus, the contract audit phase ends, in which the culprit is punished by losing reputation and the voting participants are rewarded by increasing their reputation indicators.

The reader will notice that almost all the interaction of the protocol processes described so far could be automated by the user. For instance, MPs could be issued automatically in response to predefined sets of request informations, VCs and votes in a similar manner, could be sent as an automatic response to a break detection in the former and to a newly received VC in the latter. A similar rationale could be applied to most of the transaction types, in a way that through the analysis of the chain and recent transactions, the human interactions with the system could be lowered to a minimum.

5. Threats analysis

In this section we study how Dawn protocol reacts to a set of threats and dishonest participants. Initially, all the characteristics and threats that are outside the scope of the protocol are highlighted and discussed (Section 5.1). Latter, threats related with blockchain network consensus are analysed (Section 5.2). Finally, clients and providers generated threats are discussed (Sections 5.3–5.5)

5.1. Assumptions and requirements

The first important aspect is the representation of microservices requests. The selection of description languages, models, and data structures to represent directed graphs is out-of-scope for Dawn. Based on this principle, Dawn is not strongly dependent on how the data is represented, turning the protocol extensible.

Another task exempted from Dawn's scope is the initial certification of microservice providers joining the network. However, we argue that the certificates could just be imported into the protocol through recycling of existing ones (*e.g.*, X.509), which were already previously issued by trustworthy organizations and that many service providers already possess. This would make entering the network fairly simple for existing microservices provision business. Other options include a community agreement to choose a third-party organization responsible for issuing the certificates in a partnership mode, or the institution of a dedicated certifying organization. Certification would be the only centralization entity of the network. However, it does not affect in any way the decentralization of Dawn transactions. A disfunction in the certification entity (*e.g.*, a possible fault event) during a certain time would only prevent new microservice providers from joining in, not affecting the operation of the network itself and the activities to establish contracts and audits.

In addition, Dawn is not responsible for controlling the microservices or other functions performed internally by microservice providers. The same prerequisite is applied for the instantiation of the monitoring module in the contracted infrastructure. Finally, Dawn does not include the design and implementation of the monitoring module, although it makes extensive use of its functionality. The protocol assumes that virtual private communication networks and safe channels are used to transfer data between monitoring modules and protocol actors.

5.2. Discussion on network consensus

The selection of a definitive or ideal form of consensus for the protocol is not sought by the present paper. If the basic operating characteristics of the blockchain are maintained, the protocol is successful regardless of the underlying form of consensus.

A characteristic usually related to Proof of Work (PoW) is the low flow of transactions, created by the difficulty in inserting the

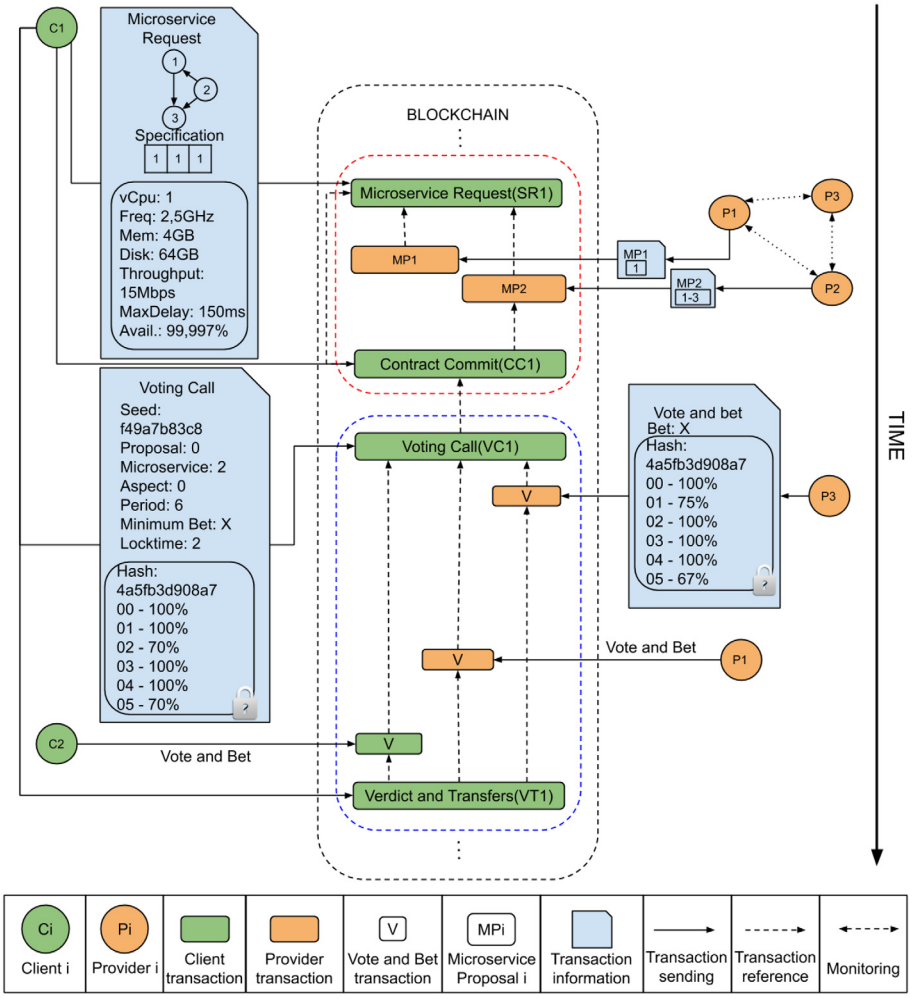


Fig. 4. Example of use of Dawn protocol.

blocks and the recommended wait of six subsequent blocks to guarantee confirmation. This disadvantage does not present an obstacle to the protocol since short times between subsequent transactions are not necessary. However, the main threat of PoW in the context of Dawn is the important difference in computational power between clients, brokers and providers, which can lead to interference with the protocol by domination in the blocks creations.

Regarding scalability, the network is not expected to reach the same number of nodes present in completely public networks, specifically general purpose cryptocurrencies such as Bitcoin and Ethereum. It is estimated that the groups of microservice providers and clients will have at most hundreds and thousands of users, respectively. Therefore, the size limits indicate that Byzantine fault tolerance could be used, employing algorithms such as Practical Byzantine Fault Tolerance (pBFT) (Castro et al., 1999) derivatives or Tendermint (Buchman, 2016), for example.

On any case, some form of incentive should be provided so that miners or voters are willing to engage on the block creation process. The first and more intuitive way is to distribute small reputation rewards as gratifications for generating new blocks, either directly to the winning miner, under PoW consensus, or to all contributing voters, on Byzantine Fault Tolerance (BFT) based ones. In order to explicitly assign these rewards, blocks could have a list of reputation additions similar to the ones employed on the Verdict and Transfers (VT) transaction, which would be utilized in a complementary way when calculating the reputation of a user following the algorithm presented on Section 3.1. The reward for a block however should not be on the same order of magnitude that the value received through contract establishments and audits, since this would allow certain users to achieve high statuses only by contributing to the block creation, and not by actually participating on contract related processes, which reputation is intended to stand for.

If a non extrinsically or non physically based form of consensus was to be employed, such as Tendermint (Buchman, 2016) or pBFT (Castro et al., 1999), then there would be no significant addition of monetary costs for a node to actively participate on the consensus process. Thus, although rewards are needed in the beginning of the network's life, their value could be possibly reduced as time passes since the existence and correct functioning of the service itself, paired with the mild operational costs, could serve as sufficient motivation for users to engage in the block creation routines.

This discussion although important, is far from the focus of the present paper and further possibilities of miner rewards as well as selection of one among the several existing consensus algorithms are regarded as implementation directed subjects and left for future work.

5.3. Common threats

Common threats consist of a set of possible actions that would be harmful to the protocol and that can be performed by any user,

regardless of their participation group. The first and most easily identifiable threat is the falsification or fraud of transactions. This threat is fortunately addressed by the very functioning of the blockchain. Although any user can write a transaction with biased data and try to register it on blockchain, an honest user will identify the fraud by executing the verification algorithm and will not allow the registration of such fraudulent data. The only possible scenario to register an invalid transaction on blockchain occurs when the consensus mechanism (*e.g.*, computing power, votes) is controlled by a majority of dishonest participants following the same illegitimate objective, which is commonly impracticable. With regard to the same spectrum of threats is the attempt to use false keys or certificates, so that the user attaches such false information to a transaction, but again the simple verification by the receiving node is already effective to prevent it.

A complex threat to be faced by Dawn is the discovery of valid keys by dishonest users, which can unfold into other attacks. However, this threat cannot be considered a protocol vulnerability because the effective protection of private keys is an inherent task to their owners and the security against a possible cryptanalysis process that will result in the discovery of keys depends on the level of robustness of the encryption method employed.

5.3.1. Contracts negotiations

Although the contract establishment phase does not have a specific set of harmful vulnerabilities, the biased manipulation of the seed attribute in the MP transaction can bring problems. Nothing would prevent a provider from instead of following the correct process – which is choosing a random number and encrypting it with the module's public key – simply publishing any clear value in the transaction. This inconvenience cannot be solved with signatures, since the corresponding module's private key is necessary to verify the data. However, this is not in fact a vulnerability of Dawn and does not operationally harm the protocol. Even if any unencrypted seed attribute is published, the decryption algorithm will generate an error due to the inconsistency of the data reported. The only negative effect arising from such action is that a provider would not be able to obtain the public voting key used to visualize client's data in a vote. Thus, it is assumed that there is no real advantage in sending an incorrect encrypted seed, as the service provider is just abdicating the visualization of client's data. The same reasoning applies to the seeds provided by clients.

5.3.2. Auditing contracts

In the audit phase there is a relatively larger set of possible threats to Dawn, mainly due to data sensitivity and restrictive interactions between participants. Initially, regarding the concept of focal points for obtaining collective conclusions, the direct exchange of sensitive information between users should not be admitted, as it would allow the emergence of collusion, in which the participants could tend to generate inaccurate or incorrect responses in order to increase their rewards or avoid punishment.

However, Dawn does not have the ability to prevent deliberate communication of voting data between users, because even with data being encrypted in different ways, there must inevitably be a form to view them to generate the appropriate collective verdict. From this analysis, dishonest users can quite simply have access to their own data, but it is not possible to have access to honest third-party users voting data regarding the security of the protocol. Such a statement is valid because in the case of sending an honest vote there is no way to view the data, since they are encrypted by the voting key, known only by the accusing user and by the accused provider.

A user can exhibit dishonest behaviour to increase reputation or damage the reputation of other users. Two strategies can be employed by the user to obtain (or generate) false monitoring data at the voting phase. The first is the artificial generation of data by observing the writing patterns of the monitoring module. In this case, a dishonest user creates a new dataset and populates it with data that meet the biased objectives. The second way of creating artificial data is by emulation, inserting a valid monitoring module in an infrastructure similar to the original request. Latter, the biased data are generated by external and artificial disturbances, making the data unreal even though originated from a valid monitoring module.

The introduction of invalid forged data of the first form is handled in a simple way by Dawn through the use of signatures and encryption in general. First, the private key used to vote is built secretly using the seeds provided by the accused provider and the accusing user. Both values are concatenated and then encrypted with the module's private key generating a resultant set of data that cannot be constructed in polynomial time by any participant outside the module.

In turn, the use of emulation to create biased data has complex remediation, as there is no easily verifiable evidence to indicate this attack. Dawn makes it difficult to perform such emulations, but has no methods of correcting and checking for occurrence. In short, the emulations are divided into two main types according to the implementation.

- Active Emulation: this type occurs when a user initiates a monitoring process in an adulterated infrastructure without having any previous monitoring data from any other user. The only plausible justification for this attack is to use the fake data through a second address that would purposefully lose the voting, while the primary address providing assertive data receives the reputation lost by the second one. Another alternative is the formation of coalitions of several dishonest users who agree on the profile of the emulation that is carried out. If the number of attackers is large enough compared to the total number of voters, the resulting verdict will be closer to the false data than to the honest one.
- Reactive Emulation: in this emulation, the attacker inserts the module into an adulterated infrastructure and uses real monitoring data from other users (discovered in some way) to direct the emulation. The reactivity of the process is due to the need of a predecessor breach of confidentially about other's monitoring data. For the emulation to happen, there must be first a disclosure of the public voting key of a client or service provider before the end of a voting session. By discovering the public voting key before the end of the established locktime, a malicious user can use it to decrypt all the votes already entered in the blockchain, and estimate – by a precalculation of the verdict – the median vote that defines the result up to that moment. Thus, the user directs the emulation to coincide with the found median, consequently increasing the final reward.

To address the feasibility of such emulations and other individual attacks the analysis is divided between clients and providers groups in the next sections.

5.4. Clients generated threats

For the client, theoretically the individual active emulation does not represent an advantageous attack since for obtaining a reputation gain at the main account is also necessary a second one, both with active contracts. It is then needed to spend time on an already contracted infrastructure to generate false data, so that the main account can receive a reputation fraction.

Through an initial analysis, this form of attack implies in a doubled cost due to the need of two similar active contracts

that demand payment. There is also the underutilization of the resources that better applied could be generating profit. At last, the main account receives just a fraction of the reputation lost by the second one, while all the other divisor users also receive a piece without any costs.

A second possibility is the individual multiple emulation, which consists of a single user possessing a group of fake accounts to endorse his victories. This form of attack is also considered highly costly because the user would need to contract a large set of infrastructures to be able to have some form of advantage in the blockchain network, which in comparison with a honest behaviour is a lot less rewarding given the expenses.

The collective multiple emulations are a lot less costly and technically less complex to be executed. The biggest challenge in this attack is to reunite a sufficiently large number of users that have the same form of contract and are willing to influence the voting. As if the group is not numerous enough the participants will lose reputation instead of gaining it, because their fake data will be away from the correct majority. Again, thanks to the concept of focal points, users tend to be honest if they believe that the majority of the network is honest as well, and the majority is considered to be in fact honest, once this assumption is also in some level a prerequisite of the protocol and the blockchain itself. Therefore this form of attack is not trivial and even upon finding means to achieve it, it is highly risky.

As for reactive emulations, there are no related high costs, since, as already pointed out, a honest set of monitoring data can be used as basis for appending emulated tests. The execution of reactive emulations depends on two main factors that are information disclosure and time availability. If information is disclosed by a dishonest attacker to a honest user, than there is no security mechanism offered by the protocol since the safety of sensitive information is a task of the user himself. Nonetheless, if the disclosure is achieved through a voluntary revelation of the voting public key, then its assumed that the revealer seeks some form of advantage by doing so. However, as the interests of an emulator user are always different from the ones of an accusing or accused user, the latter ones only will disclose the key upon belief of losing the voting since in a winning position there is no point for doing so.

Revealing the key however, would generate the opposite results for the users in such condition. The emulator users, to receive more rewards, will approximate their data to the median, which would just reconfirm the supposed loss. In order to obtain a convergence of interest through these actions, the key revealer would have to convince a large number of voters to create data on his behalf, which as already discussed in this section, is not trivial.

The second factor, time availability, refers to how much time a user has to execute the emulation, once the disclosure of the key and hence of the data has already happened. Upon the possibility of emulation, the more time a user has to do it – or the closer to the time period pointed by the accusing user – the easier it is to accurately approximate the results to the median. However, the shorter the available time period the harder it is to obtain good approximation results. The protocol uses this rationale through the VC's locktime, in a way that if the voting period is considerably smaller than the time necessary to gather the monitoring data used as proof, the impact of the emulations, if they ever occur, is very small.

5.5. Providers generated threats

In the same manner as for the clients, data forging is easily verified and its not considered feasible for providers. However, for emulations there are a set of distinct aspects that demand a new analysis. The first is the easiness to create fake client accounts, since there is no need for advanced forms of authenticity as in the inverse case. The second is the reduced amount of costs for certain attacks, as there is no need for an actual allocation or resource usage on a contract with a fake client.

Obviously, for a provider to be able to perform emulations, it is necessary an infrastructure in the form of the ones described in the contract, and although there are no inherent contract costs since the resources are owned by the provider, there is its underutilization. Despite that, the simple fact of only needing one infrastructure without contracting costs is a great facilitator for active emulation attacks, where a main account absorbs the reputation of a secondary one.

This form of attack, although substantially cheaper for providers, does not present any gains for them. This is concluded because even if a second account starts a voting with fake data against the provider, aiming at his victory, the accused user does not receive cuts of the lost reputation if he wins – as pointed in Section 3.4.5. This means that only the honest voting users would gain reputation from the fake account attack, while the attacker besides receiving nothing, would contribute to a reputation raise in a subset of the network participants, which is inherently bad if his reputation stays the same. The inverse path through individual multiple emulation, in which a provider emulates a set of data and impersonates a group of clients in his defence, is considered highly costly compared to the reputation losses possibly avoided. This is because with the network and established contracts growth, the number of necessary emulation instances grows similarly.

Still on the provider's client impersonation, when the contract is established and a voting starts, each of the actors must inform a encrypted seed value known secretly and individually by each of them. However, as the dishonest provider is behind both accounts he/she also has the knowledge of both clear seeds. If the monitoring module only used the clear values in the creation of the voting private key, then it would not be hard for the provider to reconstruct it. Possessing the referred key, the data forging would be possible and any counterfeit data could be encrypted as authentic. A horde of fake clients without any creation or monitoring cost could be used thereafter to steer the voting with the forged data. Nonetheless, as aforementioned, the module processes the clear values with its private key, so it is not possible to recreate the keys even knowing the seed values. The module's signature is also a strong additional defence against this attack.

The execution of collective active emulation assumes the same character as the discussed for the clients in Section 5.4, needing the formation of a sufficiently large group to be effective. Correspondingly, the reactive emulation presents itself in the same way in the two groups.

5.6. Analysis

Through this section a broad and general list of threats was identified and the mechanisms that the protocol utilizes to treat them and protect the users were exposed. Although the present threat and security analysis is made via theoretical lenses, its accuracy and amplitude are believed to be high, specially considering the examination of all the possible dishonest actions that could be taken in the distinct phases.

In the realm of protocol development and information security as a whole, new possible vulnerabilities can be found at any given time, which then may demand further security discussions. Nevertheless, Dawn is considered to be safe in all of its component processes.

6. Theoretical limits, issued messages and data volume

The Dawn protocol uses and trusts the blockchain mechanism to disseminate reliable data among participants. However, several transactions and data are generated by Dawn, being agnostic to the consensus mechanism used by the blockchain. In this sense, we have employed analytical modelling to investigate the minimum and maximum number of issued transactions to support Dawn, as well as the storage data volume and network traffic.

The number of issued transactions represents the total amount of unique transactions that are created by the users on each process of the protocol. This metric counts every transaction as having the same weight on the final value, with disregard to its type, size and distributed replication. Next, the storage data volume is simply the product of the number of unique transactions by the expected size for that type, representing the total amount of information that must be stored to preserve the whole history of the chain. Finally, the network traffic is an approximation of the total expected traffic that should be generated to send all the unique transactions to all the users.

To discover the theoretical limits of the metrics, we consider for each case a varying number of total clients and providers, then the necessary amount of unique transactions that must be created for all clients to establish a contract is counted. In order to make the analysis feasible and yet representative of the protocol processes, we consider some simplifications about the behaviour of both clients and providers: *(i)* all MRs sent by clients demand the same amount of microservices, denoted m; *(ii)* all microservices demand the same type of infrastructure, which is also available for purchase from all the providers; *(iii)* MPs are always sent attending to all the remaining microservices in the targeted MR or RE; and *(iv)* all clients will send votes to any new VC that accuses a provider with which they have active contracts.

From these simplifying assumptions we can derive best an worst cases for all three metrics. For each case, two equations are presented: the first is a long version that has terms representative of each protocol process and the second is its corresponding simplified version. In all of these equations t represents the total number of unique transactions, p the number of providers, c the number of clients and β the percentage of contracts that experience a SLA break.

6.1. Best case

The best case scenario happens when only the minimum number of transactions would be created to commit and audit a contract for each client. Eqs. (9) and (10) represent this metric and along this section we explain the logic behind the construction of the former, while the latter is just a direct product of mathematical manipulation.

$$t = p + 3c + \beta c + \beta c\left(\frac{c}{p} - 1\right) + \beta c + c \tag{9}$$

$$t = p + c\left(\frac{\beta(p+c)}{p} + 4\right) \tag{10}$$

The first term of Eq. (9) is relative to the CP transaction and since each provider must publish its own, p new transactions are created. For the contract establishment in the best case, the client sends a MR which is replied by only one arbitrary provider supplying all the microservices, finally the client accepts this sole proposal and commits the full contract with that provider. We assume further that MPs for each request in this case will be sent by a random provider in an uniform distribution. The process of committing a contract is represented by the second term in Eq. (9) considering that for each client three new transactions are created.

Since βc contracts will break, there will be βc VCs (third term). If the amount of clients is smaller or equal to the amount of providers, then the protocol is by definition unable to audit contracts because there would be at most one client contract for any given provider and thus there would be no votes. If however the amount of clients is at least twice the providers, then audits can occur and since the contracts are uniformly distributed among providers, there will be theoretically c/p active contracts with any given provider. Therefore, as all microservices utilize the same infrastructure, there will be $\frac{c}{p} - 1$ votes to each of the βc voting calls (fourth term). Finally, there will be βc VT (fifth term) and c CF (sixth term) transactions.

6.2. Worst case

The main difference from the best to the worst case is that in the contract establishment phase all providers will send MPs to any MR or RE from a client which has not already accepted a previous proposal from the same provider. Additionally, instead of blindly accepting the first proposal, clients will purchase only one microservice from a single proposal and will issue a RE calling for new proposals. These changes beside affecting both contract committing and auditing phases also demand to take into consideration the number of requested microservices. To better structure the analysis we divide the worst case in two forms: the first where the amount of microservices requested by any user is larger or equal to the total amount of providers ($m \geq p$) and the second where it is smaller ($m < p$). For the first form of the worst case, where $m \geq p$, the number of issued transactions is given by Eq. (11) and its simplified form in Eq. (12).

$$t = p + c + c\left(\frac{p^2 + p}{2} + p\right) + \beta c + \beta c(c-1) + \beta c + c \tag{11}$$

$$t = p + c\left(\frac{p^2 + 3p}{2} + \beta(1+c) + 2\right) \tag{12}$$

The first and two last terms of Eq. (11) represent the same processes as their parallels in the best case, namely the CPs, VTs and CFs, respectively. The second term represents the MRs, one per client and thus c in total. As the number of microservices is larger than the number of providers and a client contracts only one microservice from each MP (except when there is only one provider left), any client will end up committing a contract involving all the providers. In this sense, for each MR there will be a number of new MPs that is equal to the sum of the first p naturals, since for each new round of proposals started by the REs one additional provider will not send new transactions because his proposal was accepted. For each MR, a client will also send a RE to accept every single microservice and a CC transaction in the end of the process, resulting in a total of p additional transactions. All these transactions are summarized in the third term of Eq. (11) and again the number of VCs is expressed by βc in the fourth term. Since all clients establish a contract with all the providers, any client can vote to any voting call and hence there are $c - 1$ votes to each of the βc VCs.

$$t = p + c + c\left(\sum_{i=0}^{m}(p-i) + m\right) + \beta c + \beta c\left(\frac{qc}{p} - 1\right) + \beta c + c \tag{13}$$

$$t = p + c\left(\frac{m(2p - m + 3)}{2} + \beta\left(1 + \frac{qc}{p}\right) + 2\right) \tag{14}$$

For the second form of the worst case, where $m < p$, the number of issued transactions is given by Eq. (13) and its simplified version in Eq. (14). The only differences of this form to the first one are the third and fifth terms of Eq. (13) in relation to their parallels in Eq. (11), both corresponding to contract establishment and the votes, respectively. As the number of microservices is smaller than the number of providers and a client accepts only one microservice from one single proposal at a time, instead of committing contracts involving all the providers, a client will commit his contract with a random $\frac{m}{p}$ fraction of the total. Again, we assume clients will choose proposals to accept in a random uniform manner and therefore any provider will have active contracts with roughly $\frac{m}{p}$ clients, since that is the probability of a given provider to be included in a new contract. The amount of proposals sent by providers and the amount of REs are also similar to the total of the previous form, the difference is that now the amount of proposal rounds – and thus the amount of REs – is limited by the number of microservices, yielding for all of these transactions the third term of Eq. (13). Finally, since $\frac{m}{p}$ clients have an active contract with any given provider, for each βc voting call there will be roughly $\frac{mc}{p} - 1$ votes (fifth term).

6.3. Parameters and scenarios

To effectively quantify the amount of new transactions, we have plotted in Fig. 5 the simplified equations for both best and worst cases with varying number of clients and providers. The same was made for the total storage data volume in Fig. 6, except we have utilized the extended versions of the equations multiplying each of the terms by the associated transaction type size. With the exception of the best cases, in all of the plots the number of clients varies from 1000 to 20 000 on steps of 100, while the number of providers varies from 100 to 2000 with steps of 10. The difference for the best case plots is that the number of clients starts at 4000 to always maintain the condition of $c \geq 2p$, as discussed in Section 6.1. For the worst case, graphs were plotted for both the forms discussed in Section 6.2 and also an additional one with m fixed in 500, so the first form was employed for $m < 500$ and the second for $m \geq 500$. This seemingly arbitrary number was chosen because Uber has circa 500 microservices to support their urban mobility platform (Haddad, 2015).

6.4. Results and discussion

Initially, it is noticeable that the number of unique transactions issued varies on several orders of magnitudes from the best case on Fig. 5(a) to the largest worst case scenario on Fig. 5(b). This is mostly due to the different behaviours of the equations, since the first is quadratic on c and grows restricted by p, while the second is quadratic both in c and p. The best case also exhibits a reduction in the number of issued transactions for a given amount of clients when the number of providers rise. This can be observed because the modelling assumes that client contracts will be evenly distributed among providers, thus with more providers there are less clients contractually involved with each one of them, which in turn reduces the amount of total votes sent — mathematically, this can be visualized by the fourth term in Eq. (9) being divided by p.

For all the forms of the worst case scenario the plots have a similar behaviour, as all of them grow with a direct relation to c and p. Analysing all the three forms in Fig. 5(b–d), it is possible to note the impact of the number of demanded microservices. From the case where the number of microservices is always larger then the amount of providers to the mix case with $m = 500$ there is a reduction of 2.7x in the total transaction count. From the case with $m = 500$ to the case with $m = 50$ – where p is always larger than m – the 10x reduction in the demanded microservice total yields a 37.5x reduction in the number of issued transactions. This results present themselves this way because as pointed in Section 6.2 the demanded number of microservices dictates the maximum amount of proposal rounds sent to a client and naturally, with fewer microservices, fewer total proposals will be sent.

Table 9
Network traffic generated to transport all the data.

Case	Smallest traffic	Largest traffic
Best	43.7 GB	9.8 TB
Worst ($m \geq p$)	1.2 TB	167.6 PB
Worst ($m = 500$)	1.2 TB	73 PB
Worst ($m = 50$)	884.3 GB	8.2 PB

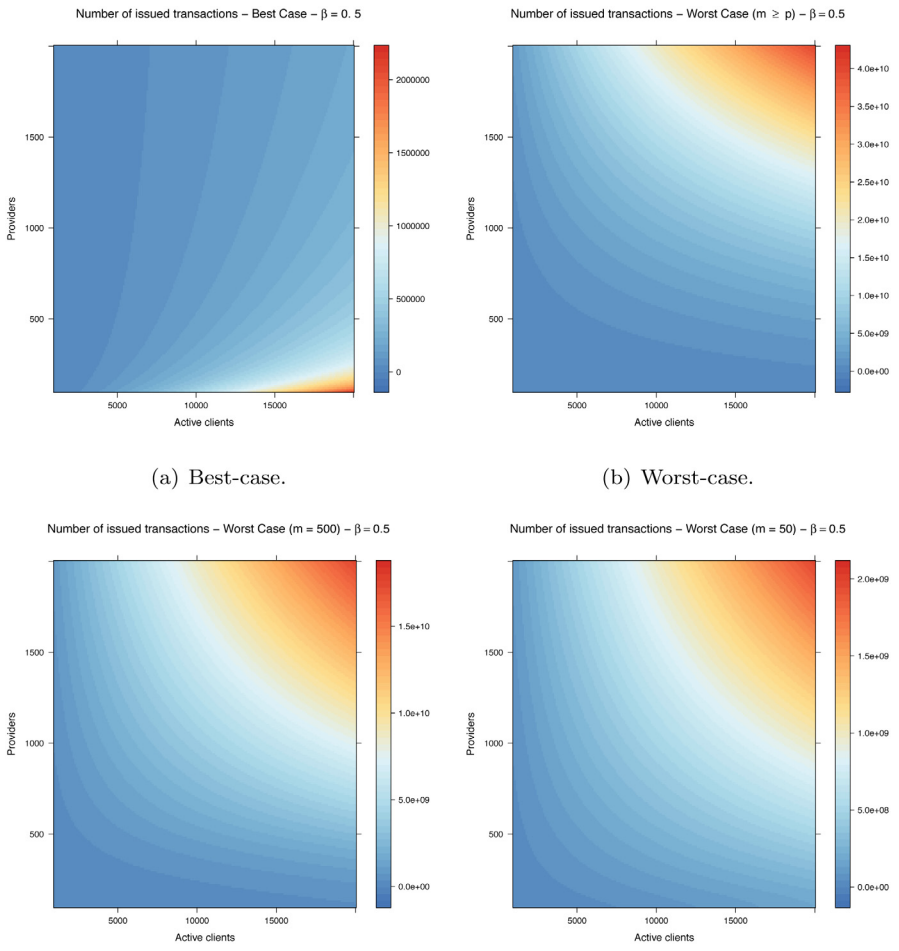
From the analysis of the total storage data volume presented in Fig. 6 it is clear that the curves follow roughly the same growth pattern as the curves for total issued transactions, that is because although the coefficient of each term has increased – from the addition of transaction sizes – the asymptotic behaviour of the equation remains the same. For the largest of the worst cases in Fig. 6(b) with the maximum number of clients and providers, there would be needed 7.8 TB of disk space to store all the protocol's history. Again as observed on the number of transactions, the total storage volume drops on several orders of magnitude from the worst to best case, where on the latter there would be needed at most 512 MB of storage space. The influence of the amount of microservices can also be spotted in the storage volume with a drop from around 3.4 TB on $m = 500$ to 393 GB on $m = 50$.

The network traffic generated to transport all the data to all the users is showed on Table 9. The values are calculated multiplying the smallest and largest data volumes for each case by the number of users minus one. These values are an estimate of the amount of traffic needed to distribute the data, however the actual traffic might be higher or lower depending on network topologies, message relaying algorithms, *etc.* As the network traffic is closely related to the data volume that must be transmitted, similar magnitude differences can be observed in this metric as well.

The need for storage and network in the best and largest worst case have very distinct resource demands. Although both these forms model real possible situations of the protocol, it is unlikely that in a practical execution the history of transactions would completely fit in one or another. This is due to the simplifications employed that create a standardized behaviour for the actors, which in turn pushes the models towards the limits of the protocol. However in an actual real scenario there would be much more fluctuations in the processes, taking the results of the metrics to a middle ground between best and worst cases. Even if some hypothetical execution fitted perfectly on the largest worst case, still it would not be strictly necessary for all users to store all the transactions, leaving to them the option to store only transactions that they deem relevant while the chain as a whole is distributed among several users or kept replicated as a single unit by a small group of users.

7. Related work

By the time of doing this research, there has not been identified other related works that address microservices contracts and their audit in a fully decentralized and trustful way as in the present paper. However, the specialized literature on selecting providers and auditing SLA, as well as on applying blockchain to support large-scale distributed systems has solid contributions.



(a) Best-case. (b) Worst-case.

(c) Worst-case mix. $m = 500$ (d) Worst-case mix $m = 50$

Fig. 5. Issued messages.

7.1. Selecting providers and auditing SLAs

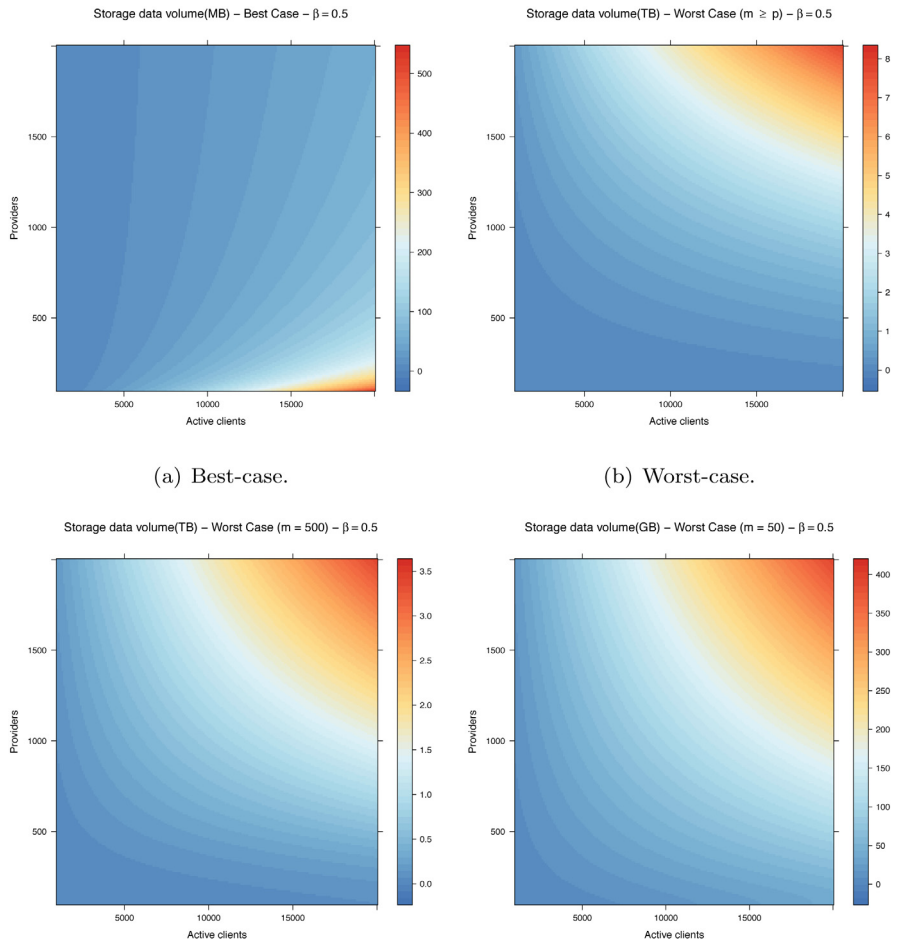
A third-party auditor for verifying cloud SLA was proposed by Zhang et al. (2014). Similarly to Dawn, the auditor mechanism does not fully rely on monitored data from service providers and supports multiple hardware performance indicators. In turn, the work of Sauvanaud et al. (2018) proposed an anomaly detection system to identify preliminary symptoms that might lead to SLA breaks. In short, the systems rely on traditional hardware counters and operating system indicators such as memory load and CPU consumption. The proposal is tailored to virtualized environments, gathering data from hypervisors and virtual machines. In Section 3.4.1 we argue that monitoring systems must be provider's agnostic and data should be verifiable for final users and competing providers.

Regarding the selection of providers, the work (Yazir et al., 2010) decomposed the management of cloud computing resources in different tasks executed by autonomous agents. This approach ensures for the providers flexibility, by varying criteria weights, and by adding or removing a criteria rather than change the cloud provider settings. In turn, a QoS-architecture for deploying services across clouds (CloudPick) (Dastjerdi et al., 2015) was proposed to rank and select providers to host virtual infrastructures (composed of multiple networked virtual machines). Finally, Multi-Criteria Decision Making (MCDM) algorithms are traditionally applied for ranking and selecting providers (Ergu et al., 2013; Rodrigues et al., 2019). The ranking method used by traditional approaches only consider criteria publicly available or directly informed by providers to tenants. Dawn deepens the discussion by including the reputation from completed votes on criteria set (as presented in Section 3.4.6).

7.2. Distributed systems management based on blockchain

The work of Uriarte et al. (2018) proposes an Ethereum blockchain network based on smart contracts for allocating and monitoring IaaS in general terms, aiming at the creation of a public infrastructure market. It used a second network destined to the deployment and operation of services, negotiations and monitoring data, while the blockchain stores smart contracts itself and violation decisions. The decision about breaks is done in a centralized manner and possibilities of decentralization are briefly discussed. An oracle is used to retrieve monitoring information from the side network into the main chain and there is no reputation control in the aid of contract establishment.

The work of Pascale et al. (2017) aims at an automation of Small Cell as a Service (SCaaS) contract establishments through the use of Ethereum smart contracts, in which small companies and home locations could rent part of their network infrastructure to mobile internet providers. The equipment is concessed through the mutually accorded smart contract and the payments are periodical. The monitoring is only a provider's task, which can apply penalties or cancel the contract.



(a) Best-case. (b) Worst-case.

(c) Worst-case mix $m = 500$. (d) Worst-case mix $m = 50$.

Fig. 6. Storage data.

In turn, Backman et al. (2017) proposed a blockchain network that in conjunction with other technologies acts as a 5G mobile Internet slicer, providing infrastructures through smart contracts to a variety of companies, specifically industrial automation devices. The allocation is made on-demand and there is need for an oracle that provides monitoring information for the appropriate decision making.

The work of Alowayed et al. (2018) proposes a blockchain based system to assign different scores to internet providers, relying on a oracle to assure information of the provided services. A two-actor conference is used to assure consensus over the data received by the oracle and a third participant can be evoked in case of fraud suspicion.

The proposal of Herbaut and Negru (2017) is a blockchain network based on Hyperledger Fabric, to help Content Delivery Network (CDN) content provision contract establishments. Multiple chains are used to deal with negotiation, distribution and monitoring, the latter being just briefly discussed. The actors are clients, providers and technical facilitators which create contracts defining searches and preferences, service provision and technical terms, respectively and in this order, which establishes a full service provision.

On Franco et al. (2019), the authors propose an Ethereum based model and implementation for a Network Function Virtualization (NFV) infrastructure auction system. The auctions are held on a Smart Contract (SC) which contains information about client's Virtual Network Functions (VNFs) and infrastructure preferences, which can be accessed by infrastructure providers in order to gather information and place bids. Multiple configuration possibilities are provided for client's definitions and provider's bid formulation, while monitoring integration is only briefly discussed.

The present paper differs from the previous ones for not only defining a protocol capable of aiding in the microservice service contract establishment but also to systematically rank the providers according to their credibility in the service provision. Our proposal also presents a solid technical form of monitoring and most importantly utilizes the monitoring data to punish dishonest users, allowing the verifiable audit of SLA and converging to a dynamic mapping of the provider's service quality.

8. Conclusions

The lack of guarantee of SLA compliance by the providers is a problem inherent in microservice provisioning contracts signed with cloud and edge providers. Dawn mainly uses blockchain technology to enable the creation and auditing of contracts. A protocol incorporating several different transactions was proposed to publish requests on the network and through them carry out processes such as contracting and auditing services. Moreover, the protocol defines each user a value in reputation points that quantitatively reflects the QoS or behaviour on the blockchain network. Monitoring processes are used to collect the services status and to support the audition of SLA contracts. Each possible outage is verified through votes based on both focal

points and data from similar contexts. The use of votes allows the verification of verdicts on the guilty and innocent, as well as the possibility of redistribution of reputation. Finally, the reputation-based negotiation used in Dawn can be used in the real world for the most diverse purposes in guaranteeing service quality and choosing new business partners.

The theoretical analysis of all possible threats highlighted that Dawn is resilient and robust in the face of the different threats, dealing well with the possible existence of dishonest or malicious users. In the future, we foresee further development of the monitoring module and the user's wallet, which endorse the proper functioning of the protocol. Another interesting consequent work is an attempt of realization of some attacks listed in the security analysis as well as new found forms to confirm the conclusions of the protocol's robustness.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The research leading to the results presented here has received funding from UDESC, Brazil and FAPESC, Brazil.

References

Alowayed, Y., Canini, M., Marcos, P., Chiesa, M., Barcellos, M., 2018. Picking a partner: A fair blockchain based scoring protocol for autonomous systems. In: Proceedings of the Applied Networking Research Workshop. In: ANRW '18, ACM, New York, NY, USA, pp. 33–39. http://dx.doi.org/10.1145/3232755.3232785.

2020. AWS CloudFormation: Model and provision all your cloud infrastructure resources. https://aws.amazon.com/cloudformation/, Accessed: 2020-09-01.

Backman, J., Yrjölä, S., Valtanen, K., Mämmelä, O., 2017. Blockchain network slice broker in 5G: Slice leasing in factory of the future use case. In: 2017 Internet of Things Business Models, Users, and Networks. pp. 1–8. http://dx.doi.org/10.1109/CTTE.2017.8260929.

Buchman, E., 2016. Tendermint: Byzantine Fault Tolerance in the Age of Blockchains (Ph.D. thesis). University of Guelph.

Castro, M., Liskov, B., et al., 1999. Practical byzantine fault tolerance. In: OSDI, Vol. 99. pp. 173–186.

Clack, C.D., Bakshi, V.A., Braine, L., 2016. Smart Contract Templates: foundations, design landscape and research directions. CoRR abs/1608.00771. arXiv:1608.00771.

Comi, A., Fotia, L., Messina, F., Pappalardo, G., Rosaci, D., Sarné, G.M.L., 2015. A reputation-based approach to improve QoS in cloud service composition. In: 2015 IEEE 24th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises. pp. 108–113. http://dx.doi.org/10.1109/WETICE.2015.28.

Dastjerdi, A.V., Garg, S.K., Rana, O.F., Buyya, R., 2015. Cloudpick: a framework for qos-aware and ontology-based service deployment across clouds. Softw. - Pract. Exp. 45 (2), 197–231. http://dx.doi.org/10.1002/spe.2288.

Di Francesco, P., Lago, P., Malavolta, I., 2019. Architecting with microservices: A systematic mapping study. J. Syst. Softw. 150, 77–97. http://dx.doi.org/10.1016/j.jss.2019.01.001, URL http://www.sciencedirect.com/science/article/pii/S0164121219300019.

Ergu, D., Kou, G., Peng, Y., Shi, Y., Shi, Y., 2013. The analytic hierarchy process: task scheduling and resource allocation in cloud computing environment. J. Supercomput. 1–14.

Franco, M.F., Scheid, E.J., Granville, L.Z., Stiller, B., 2019. Brain: Blockchain-based reverse auction for infrastructure supply in virtual network functions-as-a-service. In: 2019 IFIP Networking Conference (IFIP Networking). pp. 1–9. http://dx.doi.org/10.23919/IFIPNetworking.2019.8816843.

Garcia Lopez, P., Montresor, A., Epema, D., Datta, A., Higashino, T., Iamnitchi, A., Barcellos, M., Felber, P., Riviere, E., 2015. Edge-centric computing: Vision and challenges. SIGCOMM Comput. Commun. Rev. 45 (5), 37–42.

Gill, S.S., Buyya, R., 2018. Failure management for reliable cloud computing: A taxonomy, model and future directions. Comput. Sci. Eng..

Haddad, E., 2015. Service-oriented architecture: Scaling the uber engineering codebase as we grow. disponível em:< https://eng.uber.com/soa/>. Acesso em: 23 fev. 2019.

Heorhiadi, V., Rajagopalan, S., Jamjoom, H., Reiter, M.K., Sekar, V., 2016. Gremlin: Systematic resilience testing of microservices. In: 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS). IEEE, pp. 57–66.

Herbaut, N., Negru, N., 2017. A model for collaborative blockchain-based video delivery relying on advanced network services chains. IEEE Commun. Mag. 55 (9), 70–76. http://dx.doi.org/10.1109/MCOM.2017.1700117.

Koslovski, G.P., Primet, P.V.-B., Charão, A.S., 2009. Vxdl: Virtual resources and interconnection networks description language. In: Vicat-Blanc Primet, P., Kudoh, T., Mambretti, J. (Eds.), Networks for Grid Applications. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 138–154.

Lins, S., Schneider, S., Sunyaev, A., 2018. Trust is good, control is better: Creating secure clouds by continuous auditing. IEEE Trans. Cloud Comput. 6 (3), 890–903. http://dx.doi.org/10.1109/TCC.2016.2522411.

Mashayekhy, L., Nejad, M.M., Grosu, D., 2019. A trust-aware mechanism for cloud federation formation. IEEE Trans. Cloud Comput..

Nakamoto, S., 2009. Bitcoin: A peer-to-peer electronic cash system. URL http://www.bitcoin.org/bitcoin.pdf.

2020. OpenStack heat documentation. https://docs.openstack.org/heat/latest/, Accessed: 2020-09-01.

Pascale, E.D., McMenamy, J., Macaluso, I., Doyle, L., 2017. Smart contract SLAs for dense small-cell-as-a-service. CoRR abs/1703.04502.

Rodrigues, L.R., Cardoso Jr., E., Alves Jr., O.C., Redígolo, F.F., Pillon, M.A., Miers, C.C., Koslovski, G.P., 2019. Cloud broker proposal based on multicriteria decision-making and virtual infrastructure migration. Softw. - Pract. Exp. 49 (9), 1331–1351. http://dx.doi.org/10.1002/spe.2723.

Sauvanaud, C., Kaâniche, M., Kanoun, K., Lazri, K., Da Silva Silvestre, G., 2018. Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned. J. Syst. Softw. 139, 84–106. http://dx.doi.org/10.1016/j.jss.2018.01.039, URL http://www.sciencedirect.com/science/article/pii/S0164121218300256.

Shi, W., Zhang, L., Wu, C., Li, Z., Lau, F.C.M., 2016. An online auction framework for dynamic resource provisioning in cloud computing. IEEE/ACM Trans. Netw. 24 (4), 2060–2073. http://dx.doi.org/10.1109/TNET.2015.2444657.

Tuli, S., Mahmud, R., Tuli, S., Buyya, R., 2019. Fogbus: A blockchain-based lightweight framework for edge and fog computing. J. Syst. Softw. 154, 22–36. http://dx.doi.org/10.1016/j.jss.2019.04.050, URL http://www.sciencedirect.com/science/article/pii/S0164121219300822.

Uriarte, R.B., De Nicola, R., Kritikos, K., 2018. Towards distributed SLA management with smart contracts and blockchain. In: 2018 IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2018, Nicosia, Cyprus, December 10-13, 2018. IEEE Computer Society, pp. 266–271. http://dx.doi.org/10.1109/CloudCom2018.2018.00059.

Vaucher, S., Pires, R., Felber, P., Pasin, M., Schiavoni, V., Fetzer, C., 2018. Sgx-aware container orchestration for heterogeneous clusters. In: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). pp. 730–741. http://dx.doi.org/10.1109/ICDCS.2018.00076.

Veríssimo, P., Bessani, A.N., Pasin, M., 2012. The TClouds architecture: Open and resilient cloud-of-clouds computing. In: IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN 2012, Boston, MA, USA, June 25-28, 2012. IEEE Computer Society, pp. 1–6. http://dx.doi.org/10.1109/DSNW.2012.6264686.

Yazir, Y.O., Matthews, C., Farahbod, R., Neville, S., Guitouni, A., Ganti, S., Coady, Y., 2010. Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis. In: IEEE 3rd International Conference on Cloud Computing (CLOUD), 2010. IEEE, pp. 91–98.

Zhang, H., Jiang, H., Li, B., Liu, F., Vasilakos, A.V., Liu, J., 2016. A framework for truthful online auctions in cloud computing with heterogeneous user demands. IEEE Trans. Comput. 65 (3), 805–818. http://dx.doi.org/10.1109/TC.2015.2435784.

Zhang, H., Ye, L., Shi, J., Du, X., Guizani, M., 2014. Verifying cloud service-level agreement by a third-party auditor. Secur. Commun. Netw. 7 (3), 492–502. http://dx.doi.org/10.1002/sec.740, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/sec.740, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.740.

Wilton Jaciel Loch Master student at Santa Catarina State University (UDESC) in Joinville/SC – Brazil and received his bachelor's degree in computer science from UDESC. He carried out research activities related to scheduling and management on cloud computing resources at LabP2D (Laboratory of Parallel and Distributed Processing) of UDESC.

Guilherme Piêgas Koslovski Professor of Computer Networks and Parallel Programming at Santa Catarina State University (UDESC) in Joinville/SC - Brazil. He received his doctorate from École Normale Supérieure at Lyon/France, the master's degree from Federal University of Santa Maria (UFSM), and his bachelor's degree from the UFSM in Computer Science. Currently, his research is related to virtual infrastructures, scheduling, SLA specification, software defined networks, and virtualization of computational and communication resources. He

is the coordinator of LabP2D (Laboratory of Parallel and Distributed Processing) at UDESC in which there is a private OpenStack cloud.

Maurício Aronne Pillon Professor of Distributed Systems, and Parallel Programming at Santa Catarina State University (UDESC) in Joinville/SC - Brazil. He received his doctorate from the Institut National Polytechnique of Grenoble at France, the master's degree from Pontifical Catholic University of Rio Grande do Sul (PUC-RS), and his bachelor's degree from the Regional University of the Northwest of the State of Rio Grande do Sul in Informatics. Post-doctorate at the Federal University of Rio Grande do Sul (UFRGS), conducted in the Research Group - Parallel and Distributed Processing Group (GPPD). He currently conducts research on cloud computing, computer networks, and energy efficiency. Currently, he assists on the coordination of the LabP2D (Laboratory of Parallel and Distributed Processing) of UDESC which has a private OpenStack cloud.

Charles Christian Miers Professor of Computer Networks Security, and Computer Networks at Santa Catarina State University (UDESC) in Joinville/SC. He received his doctorate from University of São Paulo (USP) in Computer Engineering, the master's degree from Federal University of Santa Catarina's (UFSC) Department of Computer Science, and his bachelor's degree from the Santa Catarina State University in Data Processing. Security consultant at LockNet Security Solutions, in the software area, between 1999–2003, having worked on projects of national companies (private and public) and multinationals. At the Polytechnic School of the University of São Paulo (USP) he coordinated several research projects with Ericsson Research and worked on projects with RNP and the European Community (FP7). He currently conducts research on cloud computing, computer network security and ICT sustainability. Currently, he assists in the coordination of the LabP2D (Laboratory of Parallel and Distributed Processing) of UDESC which has a private OpenStack cloud; and also, is the coordinator of COLMEIA (Open Source/Hardware Research Group).

Marcelo Pasin Researcher in the University of Neuchâtel (Switzerland) and an associate professor in the Engineering School Arc of the University of Applied Sciences and Arts Western Switzerland. He holds diplomas of Doctor in Computer Science from the National Polytechnic Institute of Grenoble (France, 1999), Master in Computer Science from the Federal University of Rio Grande do Sul (Porto Alegre, Brazil, 1994) and Electrical Engineering from the Federal University of Santa Maria (Brazil, 1988). He is member of IEEE, ACM and SBC (Brazil). He currently conducts research on resource management for a high-performance publish/subscribe system, and locality optimized indexed storage for highly distributed cloud applications.