This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

JITRA: Just-In-Time Resource Allocation Through the Distributed Ledgers for 5G and Beyond

Tooba Faisal, *Member, IEEE*, Damiano Di Francesco Maesa, Nishanth Sastry, *Senior Member, IEEE*, and Simone Mangiante

***Abstract*— For the viability of future "mission-critical" applications such as remote surgery and connected cars, the customers must trust the network connection and operators must adhere to the Service Level Agreement (SLA). The key to enabling trust between the customer and the operator is the transparency and accountability of the SLA. That is, the operators ensure the transparent and appropriate execution of SLA clauses (e.g., Quality of Service (QoS) and compensations if the SLA is violated). In this work, we argue that today's network is highly volatile. Therefore, it is convenient for the operators to provide service guarantees for short-term rather than traditional long-term methods. Consequently, we advocate short-term and dynamic service contracts considering spatial and temporal characteristics rather than typical long-term contracts. We propose a Distributed Ledger Technology (DLT)-focused end-to-end transparent, accountable and automated resource provisioning system architecture in which are installed as smart contracts. In our architecture, resources can be requested and allocated dynamically and automatically with Quality-of-Service (QoS) monitoring. Our architecture is scalable through a side-channel based QoS monitoring protocol that guarantees data integrity and minimises the Permissioned Distributed Ledgers (PDL) updates. To measure the viability of our proposal, we first evaluate resource provisioning in the context of network slicing. Then we assess the DLT performance for smart contract execution, in both permissioned and permission-less settings. In the end, we evaluate the monitoring tools and compare and contrast sketches and bloomfilters, and justify our choice of bloomfilters.**

***Index Terms*— Quality of Service (QoS), smart contracts, blockchains, Service Level Agreements (SLA).**

I. Introduction

WITH the number of IoT devices expected to reach 30.9 billion [1] and autonomous vehicles expected to rise to 20 million by 2025 [2], we will see an increase in both volume and diversity in network applications. Very soon, such technological advancements will open the doors for new applications and the improved connectivity will enable life-saving procedures such as remote surgery [3]. However, such applications also require stringent Service Level Agreements (SLAs), which bring new challenges for Mobile Operators.

Manuscript received 24 August 2021; revised 12 January 2023; accepted 30 August 2023; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor E. Kalyvianaki. *(Corresponding author: Tooba Faisal.)*

Tooba Faisal is with Nokia Bell-Labs, 91620 Nozay, France, and also with the Department of Engineering, King's College London, WC2R 2LS London, U.K. (e-mail: tooba.faisal@kcl.ac.uk).

Damiano Di Francesco Maesa is with the Department of Computer Science and Technology, University of Cambridge, CB3 0FD Cambridge, U.K.

Nishanth Sastry is with the University of Surrey, GU2 7XH Guildford, U.K.

Simone Mangiante is with Vodafone Research and Development, RG14 2FN Newbury, U.K.

Digital Object Identifier 10.1109/TNET.2023.3318239

In this work, we address two key obstacles operators will face with SLAs in the near future. Firstly, with current network resource allocations methods, it is difficult for operators to cope with the *inflated* and *diverse* network connection demand: the diversity of network devices and applications is accompanied by diverse network requirements. For instance, a remote surgery may require a high-bandwidth low-latency connection, but only between two locations – the surgeon's location and the patient's location. In contrast, a connected car may find a lower bandwidth connection sufficient for its needs, but will need a reliable low-latency connection to transmit critical information from whatever physical location it happens to be in. Thus, network operators will need to support tailor-made connections rather than the traditional one-size-fits-all approach.

Secondly, mission-critical applications require *firm guarantees and accountability* from the network. When a certain service level is agreed, it must be honoured — if service quality drops below the levels agreed upon during a remote surgery for example, it can have catastrophic and potentially life threatening consequences. We argue that traditional service contracts where service agreements are made weeks or months in advance are not fit-for-purpose: Because of changing network loads, a network provider may find it transiently difficult to honour a service level agreed a long time ago. Whereas such glitches in service levels are easily tolerated by users of today's applications such as video streaming [4], they may not be ready to suffer the consequences of service level failures in mission critical applications.

Note that we do not advocate a *fast* and *Universally-Guaranteed-Service* connection for all users at all times and at all locations, but *each user may need a bespoke connection with bespoke service quality guarantees* based on their application's requirements. We argue that such guarantees are difficult for operators to honour over long periods of time. Our prior work has shown how the quality of service offered by operators can drop with increased network load, and different operators can have different performances at different times and different locations even within a major city such as London [5]. Therefore, we advocate *short-term* and *flexible* service contracts with regards to the spatial and temporal characteristics of the network. That is, customers should get network services as per their requirement and operators should offer the services *if (and only when)* they have resources (i.e., capacity) available to meet the SLA.

Our proposal is both practical and advantageous for customers and operators alike. Customers will have the liberty to choose network services and operators that best meet their

1558-2566 © 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: KINGS COLLEGE LONDON. Downloaded on October 04,2023 at 10:07:15 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

requirements at the time and based on their current application. Even for common applications today such as emails or video streaming, users may find that one operator works better in their home location and another might be more suitable at their work place, for example. Similarly, customers can make flexible trade-offs, for example choosing a cheaper connection for emails and a higher bandwidth connection for video streaming. For operators, the advantage is two-fold. Firstly, operators can predict their network behaviour in the short-term better than over a long period. Therefore, they can provide stricter service guarantees over short-term contracts. Secondly, operators can also manage congestion by incentivising their customers to shift their usage to off-peak [6].

However, managing large numbers of short-term dynamic service contracts is non-trivial, and requires a transparent, automated and accountable resource provisioning system which can cope with the future network demand and enable "killer-applications" for 5G and beyond (e.g., remote surgery). These type of mission-critical applications introduce legal liabilities (e.g., insurance claims after a failed remote surgery) which need provable guarantees that the logs have not been tampered with. To address these needs, we propose an accountable QoS monitoring system. Our architecture exploits fundamental properties of Distributed Ledger Technologies (DLT) and proposes **J**ust-**I**n-**T**ime **R**esource **A**llocation **(JITRA)** – an accountable resource provisioning system architecture in which resources can be requested and assigned dynamically with the notice of minutes [7], [8].

JITRA uses Permissioned Distributed Ledgers (PDLs), which are formed by a group of participants generally known to each other and allowed in the network with stringent access control mechanisms. Inherently, in a typical distributed ledger (e.g., PDL), all the participating nodes keep an identical copy of the ledger and particularly, PDLs are managed by a participants' led governance. For these reasons, trust and accountability is inherent to PDLs and hence PDLs are preferred for corporate applications.

JITRA is a collaborative system in which all the participating network operators are part of the same PDL network. Collaboration is indeed beneficial to increase the revenue [9] even when the competitors are involved (e.g. operators) [10]. In JITRA, operators advertise their available service contracts on a web portal (i.e., A Distributed Application (DApp)). These service contracts are backed by corresponding smart contracts. Smart contracts are software codes installed on PDLs and inherit PDL properties such as transparency and immutability.

Customers can choose their suitable network service contract request for the services dynamically. With the service request, the resources are allocated for the customer and the service is initiated. In JITRA, this feature is particularly important, a customer can purchase a service contract within minutes and with only a transaction request sent to the ledger. Moreover, in case of SLA violation, for example, if the promised QoS is not achieved, the penalty clauses will be executed and the affected party will be paid automatically.

However, smart contracts are simply software codes and lack monitoring capabilities. If a customer or operator reports false or incorrect QoS metrics, there is no built-in mechanism to verify the correctness of this data. To this end, in JITRA, we propose a side-channel based monitoring protocol to ensure data integrity. Moreover, JITRA's monitoring protocol also addresses the inherent problem of scalability for PDL and discretises the QoS provisioning by sending reports to the ledger at the only at the start and end of the service provisioning.

The rest of the paper is organised as follows: we review prior work in §II, justify our motivation in §III. We present our architecture in §IV and discussion the monitoring protocol in §V. We discuss the future work in §VII and conclude our work in §VIII.

II. Related Work

This work has three major components, 1) Dynamic Resource provisioning, 2) Smart Contracts/DLT and 3) Service Level Agreements and Accountability. In this section we outline the work close to our research.

A. Dynamic Resource Provisioning

We studied *Dynamic Resource Provisioning* in the context of network slicing. A large amount of work has been done in proposing network slicing solutions, including surveys on challenges and future directions [11], [12]. Approaches such as [13] propose mechanisms for dynamic slice reservation, which can be used as drop-in replacements in our implementation for slice reservation. Sciancalepore [14] also proposes a reinforcement learning-Based 5G Network Slice Broker which uses admission control based on traffic prediction to ensure SLA fulfilment in network slices. This can be adapted to our admission control mechanism. To our knowledge, this line of work does not discuss accountability and monitoring of delivered service in real time, which is our contribution.

B. Distributed Ledger Technology

The concept of DApp is discussed by [15] but this work is focused on components of a DApp only. Smart Contracts live on Distributed Ledgers and there are several flavours of distributed ledgers. The two primary categories are Permissionless ledgers (e.g. Bitcoin and Ethereum [16]) and Permissioned ledgers (e.g. Hyperledger Fabric [17] and R3 Corda [18]). The choice of DLT is dependent on application requirements and resources available; the demands and constraints of using DLT are discussed in [19]. A DLT focused *Network Slice Broker*, that is, a service broker that can act as a mediator between the slice provider and the customer using smart contracts and distributed ledgers is presented by [20], but this work is solely focused on the creation of smart contracts from network slice templates. *Blockchain Slice Leasing Ledger*, a blockchain focused assignment architecture for reducing network slice creation time is presented in [21]. The work closest to our approach is [22], which advocates the use of DLT for sharing slices among the Mobile Network Operators and Mobile Virtual Network Operators. Like us, this work also advocates admission control; however, they are focused towards the usage of DLT in the context of operators bidding for network slices through DLT. None of the works mentioned above exploit the inherent properties of smart contracts for accountability, nor do they ensure continuous monitoring.

Authorized licensed use limited to: KINGS COLLEGE LONDON. Downloaded on October 04,2023 at 10:07:15 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

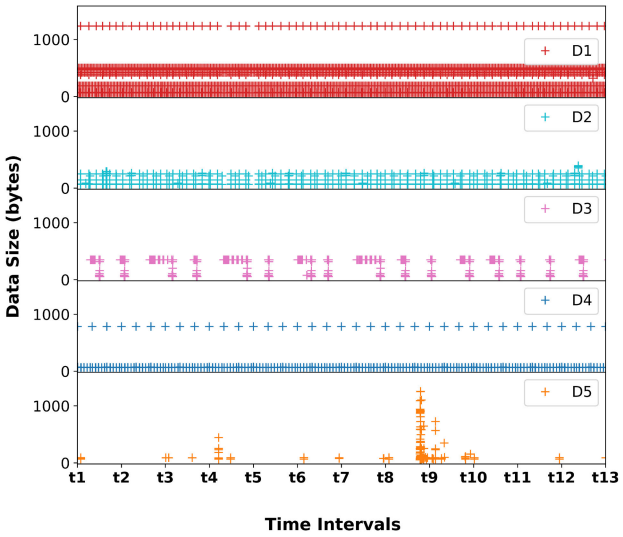


Fig. 1. Data pattern for IoT five different IoT devices. Note that, some devices (e.g., D5) send data periodically and in bursts while D1 and D2 send data continuously and better utilise the spectrum.

C. Service Level Agreements and Accountability

A proof-of-concept blockchain-based Service Level Agreement (SLA) is presented by [23] conceptualising usage of blockchain for IoT. Reference [24] also discusses SLAs in cloud infrastructure, extracting dynamic service level agreement and translating them to smart contracts. Scheid et al. [25] discuss smart contract-based automated compensation mechanisms for SLAs, and also presents a compensation prototype function but it is unclear how the monitoring would work and whether it would cope with today's line rates.

III. Motivation

Our goal is to enable dynamic, short-term and flexible service contracts. That is, the customers should be able to choose service contracts as per their requirements and for a specific duration. Similarly, the operator should advertise the service contracts as per their available capacity. That is to say, their available network resources and ability to fulfil the agreed SLA. This is an ambitious proposal and requires a strong justification. In this section, we justify our arguments with data from previous studies.

A. User Requirement Is Neither Continuous Nor Uniform

We argue that every user has a unique data requirement and customers do not always require an uninterrupted network connection. For example, a smart meter typically will need connectivity only when it will send the readings to the service provider. In another example, during a remote surgery, a surgeon would require continuous and guaranteed network connectivity. Clearly, in both cases, the network requirements are orthogonal. Previous researchers studying users' online behaviour have shown that users' data usage depends on the day (i.e., weekend and weekday) and the time (e.g., morning and evening) [26]. Therefore, to utilise the spectrum efficiently, operators need to identify users' demands and allocate resources accordingly.

As a use case, we analysed the data from [27] to understand the behaviour of IoT devices. The results are presented in Figure 1. Note that devices D1, D2 and D4 are utilising the spectrum well, which means they are sending data continuously and in a regular and predictable way. Yet, some of the devices (e.g., D3 and D5) are sending data periodically, consequently, underutilising the network. During the intervals when such devices don't send any data, the reserved network resources are wasted. But, if many such devices send data at the same time, the network can become congested.

To this end, short-term and flexible service contracts allow the customers to request service contracts *only when they require them*. For example, for remote surgery, a surgeon can acquire a network service contract for the duration of the surgery only. It is equally beneficial for operators because they can sell the peak-time contracts at a higher price and under-utilised resources (e.g., network slice) at a bargain price.

B. Spatial and Temporal Characteristics of Service Providers (i.e., Network Operators)

We strongly argue that, it is neither required nor feasible for operators to provide guaranteed QoS universally: Firstly, congestion in a particular area might be more than other areas. That is, in some areas, operators may have more demand than others. Secondly, every operator has a different coverage demand and policy. Due to many potential reasons such as legal and logistics, it is not feasible for some operators to provide a particular level of service in some regions. For instance, economics may dictate switching off a base station due to low demand.

To study operators' coverage and service quality, we analysed data from [28] and compared the Round Trip Time (RTT) of the UK's four major operators during a day. Figure 2 measures spatial variance in RTT across different base stations. We note high variance in the RTTs seen on all the operators except Op2. However, lower variance does *not* guarantee the best performance — despite Op2 having the lowest variance, it has the highest minimum RTT (36.21 ms) among all operators. Therefore, if a customer requires low-latency, Op2 may not be the best operator for them in some regions, and the customer may prefer another operator.

Next, we investigate operators' temporal characteristics using the same data. In Figure 3, we show the operators' performance at discretised time intervals. We divided the day's data into consecutive intervals of 15 minutes each (labelled t1 – t20). Note that operators' performance (i.e., RTT) is varied across all the intervals. Some of the operators perform well in some intervals but not in others. For example, Op3 has poor performance in some intervals but it does perform better than any other operator in some other intervals (e.g., t10 and t14). Thus Op3 can be the "best" choice in locations and times when it is performing well, regardless of its performance in other times and areas.

On this note, we remark that operators' performance can vary: temporal and spatial constraints impact the performance of the network. Short-term and dynamic contracts allow the

Authorized licensed use limited to: KINGS COLLEGE LONDON. Downloaded on October 04,2023 at 10:07:15 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

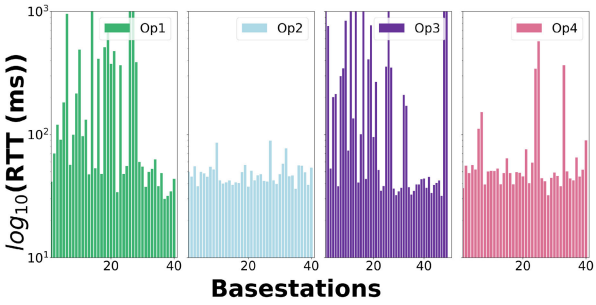


Fig. 2. Round Trip Time (RTT) at base stations of the UK's four major network providers - data is logged and capped at 1000 ms to improve readability. The minimum value for Op1, Op2, Op3 and Op4 is 20 ms, 36.21 ms, 31.71 ms and 32.07 ms, respectively. The maximum values for Op1, Op2, Op3 and Op4 are 2077.64 ms, 89.17 ms, 100,000 ms and 572.07 ms, respectively. Note that all of the operators except Op2 have a higher variance in their performance. Yet, in some particular areas, some operators indeed perform better than others. For example, despite Op3's highest max value (i.e., 100,000 ms), its minimum value is the second lowest (i.e., 31.71ms).

operators to commit to an SLA for a shorter and practically predictable period. We also infer that customers do not always require an uninterrupted network connection. Many applications on today's Internet are also non-urgent. For example, 92.3% [29] of adults in the US used the internet for text messaging. In such cases, short-term and flexible contracts are beneficial for customers equally.

In the next section, we present our system architecture that allows the operators to allocate resources automatically on short notice enabling short-term and flexible service contracts.

IV. System Design

Our goal is to set up a guaranteed Service Level Agreement *just-in-time* between an operator and customer. This is done by executing a smart contract and recording the agreement in an appropriate Distributed Ledger Technology (DLT).

Our system is built atop a specific type of DLT, Permissioned Distributed Ledger (PDL), maintained by a consortium of operators and regulatory authorities. We envision a DLT centered market place where operators can advertise available services, backed by smart contracts deployed on the PDL.

The distributed consensus nature of DLTs guarantees that the system remains honest as long the required threshold of participants (dependent on the particular distributed consensus algorithm used) behaves honestly. This means that even if the network is maintained by the operators, they can not cheat customers without the existence of a big enough cartel of operators. Moreover, the transparent nature of the DLT allows the regulatory authorities to act as watchdogs. This, coupled with the fact that the operator nodes' identities are known in advance, due to the permissioned nature of the DLT, allows for an accurate pinpointing in case of misbehaviour.

Three different type of participants interact with this system: 1) **Customers** can request network services through *Intent* [30]. Customers are the parties who require network services, more precisely, network connection for mobile or desktop applications. They request for contracts - they have limited access to the ledger through a DApp (described further in the next paragraph) and do not take part in the consensus, 2) **Service Operators** advertise their services on a portal and allocate services to the customers upon request. They run the consensus and deploy the service contracts on the ledger 3) **Regulatory Authorities** only observe the whole system for any misbehaviour, and take part in the consensus.

In addition to participants, the system model also contains a Distributed Application or DApp; see Figure 4. While a DApp is strictly not required, as customers can monitor contracts and service directly, employing a DApp relieves the customer from knowing and understanding the underlying technical details of the protocol. The DApp transparently provides to the customer the operations of contracts discovery, contracts comparison, and contracts service monitoring. In other words, the DApp can perform intent translation on behalf of the customer. The DApp can be executed as a mobile or desktop application and will run a discovery service every specified time interval to maintain an updated log of available operators in the vicinity along with their offered contracts and corresponding service quality. Service providers advertise their service contracts, and the customers purchase them through this DApp. When a user wants to buy a service contract, the DApp will list all the available contracts from the operators as per user's preference. The associated offered service metrics will be listed along with the contracts and customers will be able to browse through all the available choices from all available operators to make their choices. For example, if two different operators are providing 20 Mbps with a maximum advertised latency of 20ms, and a customer is looking for at least 10 Mbps with a maximum latency of 50 ms, then both operators' offerings are suitable for the application. The customer may choose the operator with better reputation or perhaps the operator with the better price. Given a set of customer preferences (e.g., a preference for the least cost offering, or a preference ranking for different operators with suitable offerings), the connection can also be negotiated automatically without any intervention from the customer. The DApp will be able to negotiate the service requirements it needs, on behalf of the customer. For instance, an email application can downgrade the connectivity on behalf of the customer as it does not have real-time requirements. In contrast, a video streaming application may renegotiate a different contract for the duration of a TV show [31] or movie, with minimum throughput, jitter specification etc.

As soon as the customer chooses a contract and pays for it, the DApp sends the payment to the contract escrow, where it is kept until the contract is completed, and notifies the operator to start the services. If the service is not provided as agreed in the SLA, the customer can report the degradation of service and the penalties are paid as agreed in the contract. Further details on our monitoring protocol is discussed in §V.

The sequence of actions required to set up connectivity is depicted in Figure 5:

1) Contracts are advertised on a public portal accessible through the DApp, so that a customer can select one as per their requirements. Customers are required to approve an appropriate payment with the contract request. These funds will be held in escrow, to be paid to the operator upon successful delivery of service.

Authorized licensed use limited to: KINGS COLLEGE LONDON. Downloaded on October 04,2023 at 10:07:15 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

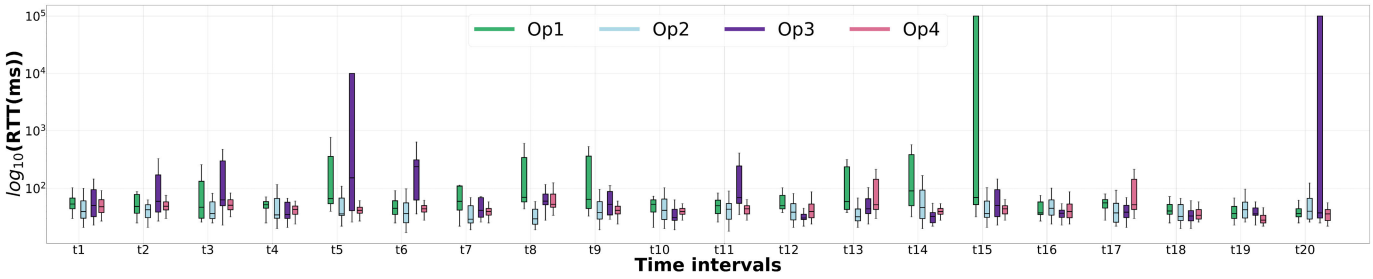


Fig. 3. Round Trip Time (RTT) of UK's four major mobile service providers at 15-minute consecutive intervals. Data is logged (i.e., $log_{10}(RTT)$) on the y-axis to improve the readability. Operator names are anonymised and referred to as Op1, Op2, Op3 and Op4.

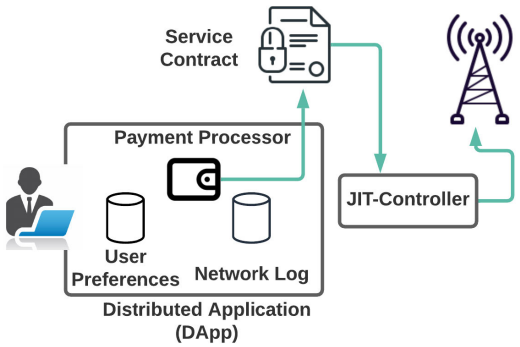


Fig. 4. Architecture of the proposed **D**istributed **App**lication (DApp) - It is run on users' device and acts as gateway to pull/push data from/to the PDL. JIT-Controller [7] performs network level resource allocation.

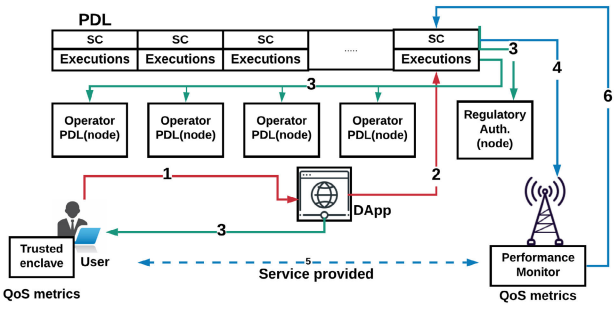


Fig. 5. Just-in-Time Resource Allocation - Architecture.

2) Once the customer requests a service, the DApp sends an activation request, signed by the customer, to the corresponding smart contract residing in the PDL. Note that on top of any payment to the operator if the smart contract executes successfully, the customer incurs a cost simply to execute it (to prevent denial of capability attacks, where a malicious customer requests services and makes an operator reserve network resources without any intention of using and paying for the services).
3) The smart contract is executed, first checking with the operator whether there are available resources that can be reserved in order to provide the requested quality of service. Depending on the jurisdiction and the nature of the contract, regulators may also need to be informed about the potential service contract being agreed upon.
4) Once the operator confirms that it is able to deliver the service (and if needed, regulator approval is obtained), appropriate resource reservations are made on operator hardware such as base stations. Then, initial state is setup to monitor the network connection during run time.
5) At this point, everything is setup, and the operator provides the requested service. Details of the service provided are recorded in a dedicated state channel established between customer and operator.
6) At the end of service, the operator provides proof from the state channel that the agreed upon service has been delivered, and claims the service payment locked in escrow by the contract.

V. QoS Accountable Monitoring

A key goal of our proposal is to allow the customer and operator to agree on a given level of service at set up time, and then to monitor and enforce the quality of the service (QoS) delivered by the operator.

Note that operators will likely charge higher prices for services with higher QoS constraints, and customers at set up time would have chosen an operator based on the price advertised for a given QoS. As such, it is essential as a mechanism to clearly flag SLA infringements by both parties: operators failing to provide the promised QoS, and customers falsely claiming a worse-than-agreed service level was delivered. We say that our QoS monitoring is *accountable* if and only if: both customer and operator can independently provide a trustworthy (i.e., cryptographically unforgeable) certificate showing the correct QoS history of the service provided so far in case of no dispute, and neither customer nor operator can provide a trustworthy certificate showing a false QoS in case of a dispute. We assume that both the customer and operator have an asymmetric cryptographic key pair, whose public key is known to each other, and therefore each can sign messages and verify message signatures for such certificates.

Our model is based on dividing the entire service provision period into discrete epochs. During each epoch, a given number of packets are sent by the operator towards the customer (or vice versa). Both parties can also exchange acknowledgements and other utility messages in both directions. For the sake of simplicity, we consider an SLA in this scenario as the *actual* set of packets that the customer and operator both agree has been sent to the customer by the operator (or vice versa). Note that this construct can capture or approximate common service level indicators. For example, from the actual set of packets, one can compute the number of packets exchanged during a given epoch, from which the achieved throughput can be calculated. By using special 'timer' packets to delineate small periods into different epochs, latency can be approximated.

Authorized licensed use limited to: KINGS COLLEGE LONDON. Downloaded on October 04,2023 at 10:07:15 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

A. Monitoring Tools

Each party (operator and customer) keeps an independent record of the packets by maintaining a cryptographic accumulator that compresses an arbitrary number of packet receipts into a single fixed length value. We require the accumulator to be cryptographically secure to avoid participants from falsely claiming that an element is contained (or not contained) in the accumulator. An accumulator allows to prove whether or not an element is contained in it, without the need to trust the accumulator creator, the drawback is that we can not efficiently list all the elements contained in it. Our system remains modular on the accumulator choice, i.e., as long as they provide the needed security guarantees, different accumulators can be plugged in depending on the use case requirements, e.g., lower costs or increased throughput.

For efficiency reasons, we also propose to employ a state channel [32] for each service agreement. Conceptually a state channel in a DLT is a private channel between two entities that agree on an initial commitment, on incremental updates and finally on a closure commitment. More precisely, both parties open the channel with a set up transaction recorded in the DLT, which specifies the channel *state* (e.g. how much is initially owed to each participant) and its additional parameters (e.g. its expiration time). Once this transaction gets accepted in the DLT, the channel can be used by the participants. To use the channel, all participants cooperate to create a new transaction updating the state of the channel. Once such transaction is well formed, any participant could broadcast it to the DLT, effectively closing the channel with the new state. All participants, however, are incentivised to keep such transaction private. Holding the well formed transaction is a guarantee to the participants that it can be released at any point to enforce the current state of the channel, so the current state is to be considered as good as if it was written in the DLT. Once all participants agree to close the channel, or any participant misbehaves by refusing to cooperate to create an updated state transaction, the honest participants broadcast the last valid state transaction to the DLT, closing the state channel. The advantage of using a state channel is threefold:

- efficiency: the state update transactions are kept private between the participants, so they are not dependent on the slow consensus times of the DLT;
- cost: as only two transactions are ever written in the DLT for each channel, i.e. the first (set up) and the last (closure), the associated fees are paid only for two transactions independently of the arbitrarily large number of private state updates;
- privacy: as all intermediate state updates are kept private, the state of the channel is only visible to outside entities when the channel is closed, and only the final state is visible not any intermediate one.

B. Monitoring Protocol

Recall that the operator and customer(s) involved commit to an SLA for a given price by running a smart contract. The same smart contract code can set up the DLT state channel and requisite initial state to monitor the SLA. The initial state consists of equivalent accumulators on both sides, whose parameters are decided by the service contract. Whenever the operator forwards packets to (or receives packets from) the customer, it records the packets in its accumulator. Likewise, the customer records in its accumulator all packets received from or forwarded to the operator. If there is a link failure (e.g., packet lost due to radio channel interference in a wireless link, or a link flap in fixed-line network) both parties are *expected* to honestly identify this and *not* record the lost packet(s). Thus, the accumulator keeps track of service delivered. For example, the number of packets added to the accumulator over a time period is a measure of bandwidth (number of packets/time period).

Service is broken down into epochs, with a part of the total price and a customisable level of service attached to each epoch. At the end of each epoch, both parties compare their accumulators. If the two accumulators match, the epoch has been successful, and the state is recorded in the state channel, and can be used for future payment. If the accumulators do not match, then they interrupt the service and start a *dispute.*

At the end of the last epoch, if no dispute was launched, the final common accumulator value is used to close the channel. The service contract can then terminate compensating the parties accordingly. If there was a dispute, i.e., accumulators do not match at the end of an epoch, the last valid channel state will be the last common accumulator value at the end of the *previous* epoch. This is then broadcast to the wider DLT and the state channel will be closed. Since the contract is structured with payments and service levels for each epoch, it ensures compensation for the part of service over which there is no dispute.

Note that because the accumulators are cryptographically secure, they can be treated as a trustworthy representation of QoS – it becomes hard for either party to add or remove a packet from their copy of the accumulator whilst still ensuring both accumulators match. When accumulators match at the end of any epoch, it protects both participants from false claims. For example, suppose the customer claims that they never received a particular packet. This can efficiently checked by querying whether it is in either of the two matching accumulators. If the packet is in the accumulator, the customer is lying, as it must have been sent by the operator. If it is not in the accumulator, it was never sent by the operator, so the operator is lying. Note that such claim verification can be performed by any third party, with no further information on the service levels.

C. Considerations

Our monitoring protocol ensures that a dispute can only jeopardise the last epoch of service provision, without invalidating the QoS measurements up to that. Moreover, the service is halted in case of a dispute. This means that both operator and customer can only attempt to get an unfair gain on the last epoch they are willing to receive/serve. This implies that, even if we can not pinpoint responsibility on a misbehaving entity, we can still mitigate the practical effectiveness of misbehaviour in general. Service contracts can employ a decreasing neutral penalty scheme on the length of the service,

Authorized licensed use limited to: KINGS COLLEGE LONDON. Downloaded on October 04,2023 at 10:07:15 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

to disincentive misbehaviour. As all service is verifiably paid as intended for all epochs except at worst the final epoch, operators can price such risk inside their service offers in advance.

The above protocol can also be strengthened depending on the use case. As an example, Trusted Execution Environments (TEE) [33] can be employed to add the guarantee that customers can only consume packets they admit to have received. If we have a TEE trusted by the operator and deployed on the customer device, then the operator can encrypt each packet before sending it to the customer. Only the TEE is capable of decrypting a packet, so the user is forced to send all packets they want to consume to the TEE first. In this scenario it would be the TEE which manages the accumulator on the customer side. The TEE decrypts and adds to the accumulator all packets that it receives before sending them in clear to the customer. This implies that consuming a packet and adding it to the accumulator are considered as a single joint atomic operation from the customer point of view, preventing them from consuming packets without adding them to the accumulator. Such higher level of security for the operator comes at a price. Encrypting and decrypting packets adds to the packet processing latency, and the requirement to deploy and run a TEE on customer devices may hamper applicability. This limits such example to application scenarios where the advantages outweigh the costs. However, this shows how it would be possible to have customisable security levels, for higher premiums, on different service offers depending on the application needs, on top of our universal basic QoS monitoring protocol.

D. Accumulator Choice

We remark that the accumulator choice is a key aspect of the proposal feasibility. Each packet needs to be recorded in the accumulator, and thereby it becomes a limiting factor for overall performance. We use *Secure Bloom Filters* (SBF) [34] as cryptographic accumulators as they outperform traditional accumulators, in terms of operations speed [35] (see §VI).

The main drawback of using SBF-based accumulators lies in their probabilistic nature. Bloom Filters assure a probabilistic false positive rate that is dependent on the filter size (i.e. number of bits) and number of elements added to the filter so far [36]. If the false positive rate is bounded to remain below a threshold to ensure a desired level of security, then the filter size increases with the number of elements. More precisely, given a false positive threshold ϵ and a filter size l, $l \geq n \cdot \log_2 e \cdot \log_2(1/\epsilon)$, where n is the number of elements [36], by adjusting n, l, ϵ, it is possible to tailor the service properties to the service contract needs. For example, ϵ can be increased for higher performance (as filter size decreases), with a trade off of lowered security.

In our scenario, the elements added to the filter are the packets exchanged. This means that the maximum size of the filter is directly related to the maximum bandwidth achievable. In fact, if we consider as bandwidth the number of packets received over an unit of time, then the maximum bandwidth b_i for a given epoch e_i of duration d_i is given by $b_i = |pkts_i|/d_i$, where $pkts_i$ is the multi set of packets recorded during that epoch e_i. We can then compute the minimum size l_i of the accumulator for a single epoch e_i as:

$$l_i = b_i \cdot d_i \cdot \log_2 e \cdot \log_2(1/\epsilon)$$

This formula allows to choose the proper accumulator efficiency (i.e. filter size) depending on the desired security (i.e. false positives rate) and throughput (i.e. bandwidth over a given epoch duration) of any given epoch. The previous formula can also be expanded for an entire service provision expected to run for at most n epochs by considering all epochs $e_i \forall i \in 1, \ldots, n$ and computing the final size L of the filter as $L = \sum_{i=1}^{n} l_i$.

Such parameters determining efficiency, security, and throughput would be specified in the service offering decided by operators, so customers can choose a service accordingly. Do note that traditional accumulators are often of constant size, making them more efficient than Secure Bloom Filters in terms of memory requirements (for example see [35]). However, we chose to use Secure Bloom Filters because their time performance advantage is more important than space efficiency in our considered application. This is because accumulator insertions are done during service provision time, so need to be as fast as possible to have a lower impact on the actual service performances. On the other hand, the accumulator values are only exchanged once at the end of each epoch (including the end of service).

VI. Evaluation

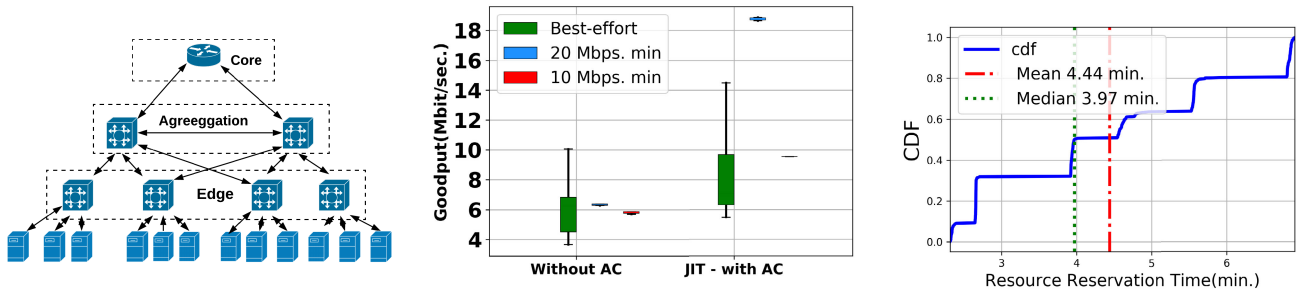
A. Resource Reservation Simulation

We evaluate the viability of our proposal on top of a topology (Figure 6(a)) that mimics in Mininet a simplified version of the topology of a major mobile operator in the UK. To send instructions to the Mininet-emulated switches we built a ***Just-in-Time Controller*** atop Ryu Controller [37], which works in collaboration with a SQLlite database. The Ryu Controller is a Software Defined Networking (SDN) framework, which allows monitoring and commanding network switches. The database keeps a record of the path assignment and available capacity of the network and updates the controller on-demand.

The incoming traffic in both the approaches is divided into three service types: two queues are prioritised by 20 and 10 Mb/s minimum bandwidth, and the rest of the traffic is directed towards a standard queue, that is "Best-effort" without any minimum service level requirement. We present our results in Figure 6(b) This clearly shows that without admission control, all classes of traffic start to contend with each other, and goodput suffers as a result. All three classes achieve only about 6Mbps, regardless of whether they asked for 20Mbps, 10Mbps or Best Effort (different best effort flows achieve different goodput, as shown by the box plot, but the median throughput remains close to 6Mbps. In contrast, with JIT-RA and admission control that does not admit new customers when resources cannot be reserved, the service types which request 20 and 10Mbps obtain close to their requested goodput. This clearly establishes the expected result that just creating a network slice is not sufficient, and *admission control and*

Authorized licensed use limited to: KINGS COLLEGE LONDON. Downloaded on October 04,2023 at 10:07:15 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



(a) (b) (c)

Fig. 6. a) Simulation Topology b) Goodput achieved by 1) Without Admission Control(AC) and 2) Just-in-Time (JIT) with Admission Control(AC) c) Resource reservation with Just-in-Time approach.

careful resource reservation are required to ensure a given service level requirement is met.

However, setting up a network slice and reserving resources comes with an overhead. Figure 6(c) shows a Cumulative Distribution of the time taken to setup and reserve resources over 350 service contracts with varying background service load. Both the mean and median are below $\approx$ 5 minutes. Conversations with a major UK mobile operator confirm that it is a realistic expectation. Thus *"just in time" resource reservation and network slice set up can happen on the order of a few minutes.* Practically, slice-setup can be avoided if operators follow a *"Blue-print"* approach, by grouping similar resource requests together and assigning them to a *pre-configured* set of network functions.

B. Distributed Ledger Comparison

As our study is focused on *Resource Allocation of Network Resources using Smart Contracts*, it is essential to measure the overheads incurred by the employed PDL and smart contracts. The rationale behind adopting a PDL over a permissionless approach is explained in §IV. However, to measure the practical impact, we deployed 200 service contracts on two different permissioned ledgers, Quorum [38] and Hyperledger Fabric [17] and a permissionless ledger, the public Ethereum Ropsten testnet.

The example service contracts are executed on two local nodes of Hyperledger Fabric (Version 1.3.0) through a Hyperledger Burrow [39] running on a Ubuntu 64-bit, 16.04.7 virtual machine with 4.096 GB of RAM, and a 2.3 GHz Dual-Core Intel Core i5 processor. For Quorum we used an identical virtual machine as Hyperledger Fabric and installed two Quorum nodes with RAFT [40] consensus.

Overall, Quorum's Raft protocol is a leadership model, in which a single leader is elected to manage the replication of transactions to other nodes and is ideal for closed groups of nodes where fast block creation is required, ideally at the granularity of milliseconds [40]. In Hyperledger Fabric transactions are sent to endorsers first (in our case we have two endorsers) who endorse the transactions as per the endorsing policy of the ledger. Once the transactions are endorsed, they are sent to orderers (one orderer in our experimental setup), who use Apache Kafka to reach a consensus [41].

This difference in the speed of the two consensus algorithms is reflected in our analysis, in which Quorum outperforms Hyperledger Fabric on average. The mean execution time for Hyperledger Fabric is $\approx$ 91 ms which is a bit higher than Quorum's that is $\approx$ 68 ms. However, the standard deviation is $\approx$ 16 ms and $\approx$ 24 ms for Hyperledger Fabric and Quorum, respectively. That is, although Quorum has lower execution times than Hyperledger Fabric, the standard deviation is higher, meaning that more diversified execution times should be expected with Quorum. This is also visible from Figure 7(b), where Quorum is slower than Hyperledger Fabric in the worst case. This indicates that Hyperledger Fabric could be a more desirable alternative in application scenarios where the consistency of performances is more desirable than speed. In addition to our prior work [7], we deployed another 200 contracts on Ropsten through a node with an Intel Xeon CPU E5-2660 (2.60 GHz with 20 cores) and 94G RAM. The analysis shows that it took an average of $\approx$ 49.3 sec to execute the same contracts on the Ropsten testnet with a standard deviation of $\approx$ 35 sec.

The difference between contracts execution times measured on permissioned and permissionless ledgers is an entire order of magnitude, confirming the performance advantages of permissioned ledgers. Another factor, however, is that the permissionless experiment was conducted on a possibly congested public testnet, while both permissioned experiments were set up on dedicated private testnets.

C. Accumulator Comparison

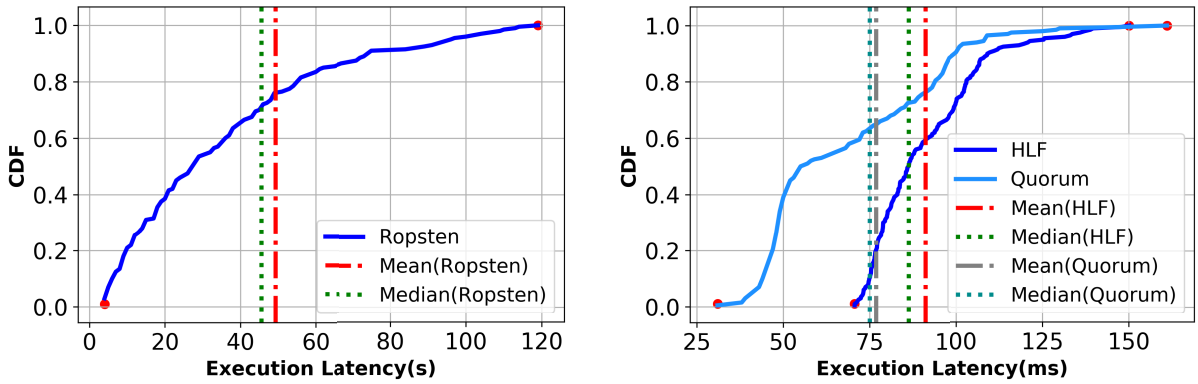
After set up, the main constraining factor while the service contract is running is the overhead of maintaining per-packet records in accumulators. We studied three different data structures and evaluate their viability for JITRA. In a prior study, we compared the performance of two accumulators, RSA-Accumulators and Secure Bloom Filters (SBF). Our evaluations show that SBF have higher performance then their counterparts (i.e., cryptographic accumulators) [42].

In this study, we explore the performance of SBF with sketches [43]. Both the sketches and SBF are compact data structures and provide a summary of big data, for example, IP packets. However, both of them work differently and have trade-offs [44].

In SBF, elements are stored in a single-dimensional array (i.e., a bitarray) which only provides the membership proof. Note that, it is not possible to recover the original elements from a SBF. The elements are hashed and corresponding bits

Authorized licensed use limited to: KINGS COLLEGE LONDON. Downloaded on October 04,2023 at 10:07:15 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



(a) Permissionless Ledger (b) Permissioned Ledgers

Fig. 7. Distributed Ledger comparison of execution latency in a) Permissionless ledger, that is Ethereum Ropsten Testnet and b) Permissioned ledgers, that are, Hyperledger Fabric and Quorum. Note that, permissionless ledgers have high execution latency than permissioned ledgers. Furthermore, within permissioned ledgers, Quorum outperforms hyperledger fabric due to Quorum's leadership consensus.

get set in the bitarray. However, since SBF stores all the elements in one array, there is a possibility of overlapping bits, thus false positives. Of course, false positives can be reduced by increasing the number of hash functions, that is, by calculating different hashes of the same element and setting different bits. Yet, it comes with increased costs, for example, a larger SBF size. To estimate the size of SBF required for 1Gbps connection, we investigated the standard quation (Equation 1) for SBF defined in [43] and list our calculations in Table I.

$$k = (m/n)\ ln2 \tag{1}$$

Here, m is the number of bits of the array, n is the number elements expected to be stored in the bloom filter, optimal number of hash functions k. The size of bitarray (SBF) m is dependent on the tolerable probability of false positives – to reduce the probability of false positives the array size needs to be increased. Interested readers can refer to [43] for details.

Next, we evaluate sketches, multi-dimensional data structures that maintain a counter for each element rather than saving the complete item. In sketches, each hash function corresponds to an array. Similar to SBF, original elements from sketches cannot be recovered. Recall that our goal in JITRA is to measure the QoS rendered. With sketches, every incoming packet can increment the counter, providing a summary of the number of packets received/sent. Therefore, sketches can be a workable candidate for our study.

To compare and contrast SBF and sketches, we added 64-byte packets one at a time for 1000 iterations. Note that for a given amount of data to be transferred, using small (64 byte) sized packets lead to a larger number of packets, and therefore represents the worst case upper bound for overheads.

In Figure 8(a), note that SBF are generally faster than sketches but have higher variance. Recall that in SBF, elements are added by setting bits to 1, but in sketches the counter is incremented which is more expensive than setting bits. This makes sketches slower as compared to SBF.

Next factor we measure is the size of the data structure (Figure 8(b)). Note that, the size of a data structure increases in both sketches and SBF with the increase of hash functions. Yet, in sketches every hash function represents a row, consequently adding a hash function increases a row in the sketch. In SBF, however, all the hash functions correspond to the same array therefore, the size doesn't increase exponentially.

TABLE I
OPTIMUM SBF SIZE AND NUMBER OF HASH FUNCTIONS WITH GIVEN PROBABILITIES OF ERROR

Probability of Error	Bloom Filter size (bits)	Hash Functions
0.0001	37441635	13
0.00001	46802043	17
0.000001	56162452	20
0.0000001	65522861	23
0.00000001	74883269	27
0.000000001	84243678	30
0.0000000001	93604086	33

In JITRA, accumulator filling time is dependent on the epoch size – The sooner the accumulator is filled the finer the epoch can be and finer-grained the QoS monitoring would be. Recall from §V that we add packets to an accumulator data structure and once the accumulator is filled, the receipt is sent to the operator. Note that in Figure 8(c), initialisation time increases exponentially both for sketches and SBF (note the logarithmic scale). Sketches are multi-dimensional, therefore, increasing the hash functions in fact increases the number of rows in the sketch. Clearly, the larger the data structure, the longer it will take to initialise. In case of SBF, however, all the hash functions share the same array and array size depends only on false positive probability. Therefore we note a great difference in initialisation time between the two approaches. We do remark how the different level of optimisation of the two adopted libraries may greatly influence such times as well.

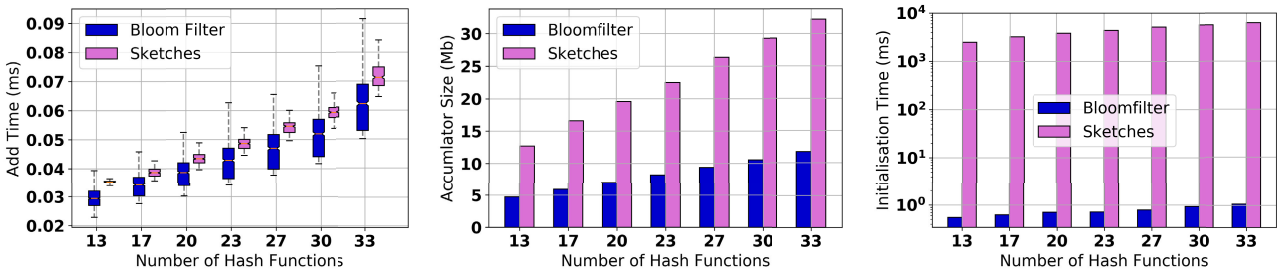
Clearly, we note a trade-off while comparing these two data structures. Indeed, SBFs are more efficient both in terms of space and performance, but they have the limitation of false positives. On the other hand, sketches, indeed, provide a better picture of the data but they are more costly at runtime.

VII. DISCUSSION & FUTURE WORK

In this work, we proposed a novel concept of SLA accountability through short-term and dynamic service contracts. Our architecture is PDL-focused resource reservation architecture

Authorized licensed use limited to: KINGS COLLEGE LONDON. Downloaded on October 04,2023 at 10:07:15 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



(a) Add time in SBF and Sketch

(b) Accumulator size with a given number of hash functions

(c) Accumulator setup time with a given number of hash functions

Fig. 8. Our Analysis shows that SBF outperforms sketches in terms of add latency, size, setup time and connection speed. Note that, this performance is achieved with the cost of error probability in SBF. Details of error probability with number of hash functions are given in Table I.

in which network operators work collaboratively to enable automated and accountable service provisioning.

Indeed, our proposal is ambitious and opens several lines of discussion. We record two types of data in the PDL, 1) the service contract (i.e., SLA) as a smart contract, 2) the execution record of SLA (i.e., executions and QoS receipts). We discuss the consequences of storing both.

Firstly, because data stored in PDLs are immutable, all the transactions and smart contracts become immutable, which aids scalability. However, this also means that the correctness of the smart contracts stored and consequently the service contracts offered by operators is important. Recall from §IV that we record service contracts only once and they can be shared among service providers. Sharing contracts improves the quality of the contracts in terms of correctness. For example, a shared service contract will be scrutinised by several service providers before installation to ensure the correctness and quality of the contract.

For the PDLs' monitoring overheads (i.e., receipts recording), recall from §V that we use a DLT state channel for scalability and the PDL records SLA metrics only in the last and the first epoch. This also keeps the size of the PDL small by decreasing the amount of state needing to be stored. Recall that a key motivation for our architecture is accountability. We intend to design a system, in which records can be maintained immutably and available transparently for audit for a long time. PDLs fits this motive and auditbility is achieved with a small cost of space.

Secondly, the shift from traditional longer contracts to short-term and flexible contracts indeed requires support from our industry partners. Our analysis from §III aligns with prior work [5] and shows that operators' service fluctuates with time and with area. To enable guaranteed service quality, it is essential that the service contracts are allocated with regards to available capacity. This is in fact advantageous for the operators as well, as they can incentivise (e.g., through better price) their customers to shift their usage to off-peak and better manage their congestion.

In our architecture, smart contracts have to be manually coded by operators and installed after checking for correctness and suitability to their network infrastructure. In our future work, we intend to extend this work and design smart contracts which can adapt to network conditions and adjust the SLA metrics accordingly. We also intend to develop a domain specific language for expressing service level requirements in smart contracts in ways that avoid pitfalls of smart contract bugs [45].

VIII. Conclusion

In this work, we advocate short-term and dynamic service contracts and presented JITRA – DLT-focused end-to-end architecture for dynamic and accountable network service provisioning. We believe guaranteed service quality is achievable through customised and tailor-made network services rather than traditional one-size-fits-all. To this end, we installed service contracts (i.e., SLAs) in a permissioned PDL as smart contracts enabling transparency and accountability in the system.

We evaluated three main components of JITRA: firstly, resource provisioning – our results show that it takes on the order of minutes to set up a network slice and make hard resource reservations. Secondly, we looked at the overheads incurred due to smart contracts both in permissioned and permissionless settings and showed that permissioned ledgers indeed outperform permissionless ledgers. Finally, we discussed ways to scalably monitor the QoS provided using two different data structures – secure bloom filters and sketches. Our results showed that bloom filters outperform sketches at the expense of more false positives.

This study enables a new era of network service provisioning, where accountability is of prime importance. We next aim to extend this work and explore the economic aspects in this context. In particular, our approach allows "Just-in-Time" pricing of network contracts to maximise operators' profits and customers' welfare while being cognizant of network neutrality issues.

References

[1] L. S. Vailshery. (2021). *Internet of Things (IoT) and Non-IoT Active Device Connections Worldwide From 2010 to 2025*. [Online]. Available: https://bit.ly/2WtsSHX

[2] G. Maidment. (2021). *In Safe Hands: Connected Cars Will Quickly, Quietly, and Carefully Change Your Life*. [Online]. Available: https://bit.ly/3BO1mVJ

[3] P. Gallagher. *Doctors Carried Out Complex Throat Surgery While Sat at His Desk Using 5G*. Accessed: Oct. 6, 2020. [Online]. Available: https://bit.ly/2Lr5s0k

[4] D. Karamshuk, N. Sastry, A. Secker, and J. Chandaria, "On factors affecting the usage and adoption of a nation-wide TV streaming service," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 837–845.

Authorized licensed use limited to: KINGS COLLEGE LONDON. Downloaded on October 04,2023 at 10:07:15 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

[5] E. Obiodu, N. Sastry, and A. Raman, "Towards a taxonomy of differentiated service classes in the 5G era," in *Proc. IEEE 5GWF*, Jul. 2018, pp. 129–134.
[6] S. Ha, S. Sen, C. Joe-Wong, Y. Im, and M. Chiang, "TUBE: Time-dependent pricing for mobile data," in *Proc. ACM SIGCOMM*, 2012, pp. 247–258.
[7] T. Faisal, D. D. F. Maesa, N. Sastry, and S. Mangiante, "Automated quality of service monitoring for 5G and beyond using distributed ledgers," in *Proc. IEEE/ACM IWQoS*, Jun. 2021, pp. 1–6.
[8] T. Faisal, D. D. F. Maesa, N. Sastry, and S. Mangiante, "How to request network resources just-in-time using smart contracts," in *Proc. IEEE ICBC*, May 2021, pp. 1–5.
[9] P. J. Daugherty et al., "Is collaboration paying off for firms?" *Bus. Horizons*, vol. 49, no. 1, pp. 61–70, 2006.
[10] D. R. Gnyawali and B.-J. R. Park, "Co-opetition between giants: Collaboration with competitors for technological innovation," *Res. Policy*, vol. 40, no. 5, pp. 650–663, 2011.
[11] P. Rost et al., "Network slicing to enable scalability and flexibility in 5G mobile networks," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 72–79, May 2017.
[12] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 94–100, May 2017.
[13] K. Samdanis, X. Costa-Perez, and V. Sciancalepore, "From network sharing to multi-tenancy: The 5G network slice broker," *IEEE Commun. Mag.*, vol. 54, no. 7, pp. 32–39, Jul. 2016.
[14] V. Sciancalepore, X. Costa-Perez, and A. Banchs, "RL-NSB: Reinforcement learning-based 5G network slice broker," *IEEE/ACM Trans. Netw.*, vol. 27, no. 4, pp. 1543–1557, Aug. 2019.
[15] G. S. Ramachandran, R. Radhakrishnan, and B. Krishnamachari, "Towards a decentralized data marketplace for smart cities," in *Proc. IEEE ISC2*, Sep. 2018, pp. 1–8.
[16] Ethereum. *Ethereum*. Accessed: Nov. 23, 2020. [Online]. Available: https://ethereum.org/en/
[17] Hyperledger. *Hyperledger Fabric*. Accessed: Nov. 23, 2020. [Online]. Available: https://bit.ly/3gyHD22
[18] R. Corda. *R3—Corda*. Accessed: Nov. 23, 2020. [Online]. Available: https://bit.ly/377BqXM
[19] K. Wüst and A. Gervais, "Do you need a blockchain?" in *Proc. IEEE CVCBT*, Jun. 2018, pp. 45–54.
[20] B. Nour, A. Ksentini, N. Herbaut, P. A. Frangoudis, and H. Moungla, "A blockchain-based network slice broker for 5G services," *IEEE Netw. Lett.*, vol. 1, no. 3, pp. 99–102, Sep. 2019.
[21] J. Backman, S. Yrjölä, K. Valtanen, and O. Mämmelä, "Blockchain network slice broker in 5G: Slice leasing in factory of the future use case," in *Proc. Internet Things Bus. Models, Users, Netw.*, Nov. 2017, pp. 1–8.
[22] M. A. Togou et al., "A distributed blockchain-based broker for efficient resource provisioning in 5G networks," in *Proc. IWCMC*, Jun. 2020, pp. 1485–1490.
[23] A. Alzubaidi, E. Solaiman, P. Patel, and K. Mitra, "Blockchain-based SLA management in the context of IoT," *IT Prof.*, vol. 21, no. 4, pp. 33–40, Jul. 2019.
[24] R. B. Uriarte, R. de Nicola, and K. Kritikos, "Towards distributed SLA management with smart contracts and blockchain," in *Proc. IEEE CloudCom*, Dec. 2018, pp. 266–271.
[25] E. J. Scheid, B. B. Rodrigues, L. Z. Granville, and B. Stiller, "Enabling dynamic SLA compensation using blockchain-based smart contracts," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, Apr. 2019, pp. 53–61.
[26] E. Bulut and B. K. Szymanski, "Understanding user behavior via mobile data analysis," in *Proc. IEEE ICCW*, Jun. 2015, pp. 1563–1568.
[27] A. Sivanathan et al., "Classifying IoT devices in smart environments using network traffic characteristics," *IEEE Trans. Mobile Comput.*, vol. 18, no. 8, pp. 1745–1759, Aug. 2019.
[28] E. Obiodu, A. Abubakar, A. Raman, N. Sastry, and S. Mangiante, "To share or not to share: Reliability assurance via redundant cellular connectivity in connected cars," in *Proc. IWQoS*, Jun. 2021, pp. 1–6.
[29] J. Johnson. *Most Popular Online Activities of Adult Internet Users in the United States as of November 2019*. Accessed: Aug. 10, 2021. [Online]. Available: https://bit.ly/3CAbC4m
[30] Y. Sharma, D. Bhamare, N. Sastry, B. Javadi, and R. Buyya, "SLA management in intent-driven service management systems: A taxonomy and future directions," *ACM Comput. Surv.*, vol. 55, no. 13s, p. 38, Jul./Dec. 2023.
[31] D. Karamshuk, N. Sastry, A. Secker, and J. Chandaria, "ISP-friendly peer-assisted on-demand streaming of long duration content in BBC iPlayer," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 289–297.
[32] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *Proc. SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 949–966.
[33] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, vol. 1, Aug. 2015, pp. 57–64.
[34] Z. Li and G. Gong, "On data aggregation with secure Bloom filter in wireless sensor networks," Dept. Elect. Comput. Eng., Univ. Waterloo, Waterloo, ON, Canada, Tech. Rep., 2010.
[35] A. Kumar, P. Lafourcade, and C. Lauradoux, "Performances of cryptographic accumulators," in *Proc. IEEE LCN*, Sep. 2014, pp. 366–369.
[36] A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: A survey," *Internet Math.*, vol. 1, no. 4, pp. 485–509, 2004.
[37] RYU SDN Controller. (2020). *RYU SDN Framework*. Accessed Nov. 20, 2020. [Online]. Available: https://ryu-sdn.org/
[38] Quorum. (2021). *Consensus Quorum Blockchain*. Accessed: Mar. 8, 2021. [Online]. Available: http://bit.ly/2OgZGzZ
[39] HyperLedger. (2020). *Burrow*. Accessed: Dec. 12, 2020. [Online]. Available: https://bit.ly/3nmcukT
[40] GoQuorum. (2021). *Raft Consensus Overview*. Accessed: Mar. 8, 2021. [Online]. Available: http://bit.ly/3rA1SAZ
[41] Apache. (2021). *Kafka Documentation*. Accessed: Mar. 8, 2021. [Online]. Available: http://bit.ly/3vmkqXI
[42] T. Faisal, D. D. F. Maesa, N. Sastry, and S. Mangiante, "AJIT: Accountable just-in-time network resource allocation with smart contracts," in *Proc. ACM MobiArch*, 2020, pp. 48–53.
[43] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.
[44] G. Cormode, "Data sketching," *Commun. ACM*, vol. 60, no. 9, pp. 48–55, Aug. 2017, doi: 10.1145/3080008.
[45] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "ZEUS: Analyzing safety of smart contracts," in *Proc. NDSS*, 2018, pp. 1–12.

Tooba Faisal (Member, IEEE) received the B.S. degree in computer engineering and the M.S. degree in telecommunication and networks from Bahria University, the M.Res. degree in security science from University College London, and the Ph.D. degree from King's College, London. She is currently a Research Web3 Engineer with Nokia Bell-Labs, France. She led ETSI ISG PDL report and specifications on smart contracts. Her current research interests include decentralized systems and next-generation networks.

Damiano Di Francesco Maesa received the Ph.D. degree in computer science from the University of Pisa. He specializes in the analysis and study of novel applications of blockchain and distributed ledger technologies (DLT). His past affiliations include the University of Cambridge and King's College London, U.K. He is currently an Assistant Professor with the Department of Computer Science, University of Pisa, Italy.

Nishanth Sastry (Senior Member, IEEE) is currently the Co-Lead of the Distributed and Networked Systems Group, Department of Computer Science, University of Surrey. He is also a Visiting Researcher with the Alan Turing Institute, where he is the Co-Lead of the Social Data Science Special Interest Group. He spent over six years in the Industry, such as Cisco Systems, India, and IBM Software Group, USA, and Industrial Research Labs, such as IBM Thomas J. Watson Research Center. He has also spent time with the Computer Science and AI Laboratory, Massachusetts Institute of Technology.

Simone Mangiante received the Ph.D. degree in computer networks from the University of Genoa, Italy, in 2013, with a focus on carrier Ethernet management using the SDN paradigm. Then, he spent three years with Dell EMC in Ireland as a Senior Research Scientist, where he managed European projects focusing on SDN and network transport. He is currently a Research and Standards Specialist at Vodafone Research and Development Group, U.K. He focuses on MEC architecture, life cycle management and orchestration aspects of MEC, edge cloud to public/private cloud integration, and distributed cloud technologies for 5G. His research interests include edge computing, distributed cloud architecture, and NFV.

Authorized licensed use limited to: KINGS COLLEGE LONDON. Downloaded on October 04,2023 at 10:07:15 UTC from IEEE Xplore. Restrictions apply.