Automated Quality of Service Monitoring for 5G and Beyond Using Distributed Ledgers

Tooba Faisal, *King's College London, UK*
Damiano Di Francesco Maesa, *University of Cambridge, UK*
Nishanth Sastry, *University of Surrey, UK*
Simone Mangiante, *Vodafone Group R&D, UK*

Abstract—**The viability of new mission-critical networked applications such as connected cars or remote surgery is heavily dependent on the availability of truly customized network services at a Quality of Service (QoS) level that both the network operator and the customer can agree on. This is difficult to achieve in today's mainly "best effort" Internet. Even if a level of service were to be agreed upon between a consumer and an operator, it is important for both parties to be able to scalably and impartially monitor the quality of service delivered in order to enforce the service level agreement (SLA). Building upon a recently proposed architecture for automated negotiation of SLAs using smart contracts, we develop a low overhead solution for monitoring these SLAs and arranging automated payments based on the smart contracts. Our solution uses cryptographically secure bloom filters to create succinct summaries of the data exchanged over fine-grained epochs. We then use a state channel-based design for both parties to quickly and scalably agree and sign off on the data that was delivered in each epoch, making it possible to monitor and enforce at run time the agreed upon QoS levels.**

I. Introduction

5G has become reality, and its key enabler applications such as low-latency industry control as well as consumer applications are making their way to the market. For instance, in health, service providers demonstrated distant remote-operated surgery [1]. Further, in the automotive industries, the number of 5G-connected cars is expected to be around 100 million by 2025 [2]. As promises are being made, prototypes are show-cased and expectations begin to grow, the real-life adoption of these applications needs some operational and legal assurance. If the Quality of Service (QoS) drops during a remote surgery, what will be the mechanism to pinpoint where the fault lies? How to deal with a connection drop when a remotely connected ambulance [1] passes through an area where there is no connectivity from its usual operator? Such challenges call for radical changes to present service provisioning mechanisms.

Previously, we had proposed the notion of **Accountable Just-in-Time (AJIT)** allocation of network resources [3] as a novel solution to support the guaranteed QoS needs of mission-critical applications. Unlike current arrangements, where customers and operators manually negotiate Service Level Agreements (SLAs) weeks or months in advance and QoS is provided based on *statistical multiplexing*, AJIT allows service levels to be provisioned a few minutes in advance of when connectivity is actually needed using *hard resource reservations*. Because these contracts can be negotiated automatically, the service contract provisions can be for short well-defined periods (even on the order of minutes), making it economically viable for operators to provide connectivity based on hard resource reservations

This *just-in-time* vision is enabled by automated negotiation of service levels via **parameterised smart contracts**. Smart contracts are software codes, installed on Distributed Ledgers and possess properties which are essential to create Monitorable and automated SLAs [4]. Operators advertise guaranteed SLAs as smart contracts with parameters for key performance indicators such as minimum bandwidth or maximum latency. The price charged for the service would depend on these parameters as well as other operator-specific costs. When the smart contract is executed, the operator is first queried whether it has sufficient resources to deliver the requested service. Once the operator indicates it is able to deliver the service and reserves resources, payment is immediately taken from the customer and held in escrow, and the operator is bound to the agreed-upon service level for the duration negotiated. At the end of the service engagement, the smart contract checks whether the operator has upheld the service levels agreed, and transfers the escrow amount to it. The details of the architecture is given in another work [5], [3] and omitted here for space constraints.

Note however that there may be a dispute over whether the service was rendered at an adequate level, so the connection needs to be monitored and records need to be maintained about which packets were delivered and at what time. Doing this at so-called 'line rates' (maximum possible data rates) of today's communication networks is hard. We develop a novel solution based on dividing the overall contract duration into small epochs, and creating records over each epoch (§III).

We evaluate the feasibility of this design by asking the following questions: *(i)* How long does it take to set up such smart contract-based SLAs? Can it truly be "just-in-time"? *(ii)* What is the overhead of maintaining per-packet records, and can this sustain reasonable data transfer rates? We examine these questions in §IV and show that new connections can be set up within approximately 5 minutes in a topology that mimics a national-scale mobile operator. Resources are reserved by dynamically creating dedicated network slices, and this helps sustain deliver the requested quality of service. Even with the overhead of maintaining per-packet records in cryptographically secure bloom filters, data rates of up to 50Mbps can be sustained.

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 [6], [7]. Approaches such as [8] propose mechanisms for dynamic slice reservation, which can be used as drop-in replacements in our implementation for slice reservation. Sciancalepore [9] 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 choice of DLT is dependent on application requirements and resources available; the demands and constraints of using DLT are discussed in [10]. 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 [11], 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 [12]. The work closest to our approach is [13], in which authors advocate the use of DLT for Network slices sharing among the Mobile Network Operators(MNOs) and Mobile Virtual Network Operators(MVNOs). 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 work mentioned above exploit the inherent properties of smart contracts for accountability, nor do they ensure continuous monitoring.

C. Service Level Agreements and Accountability

A proof-of-concept blockchain-based Service Level Agreement (SLA) is presented by [14] conceptualising usage of blockchain for IoT. [15] also discusses SLAs in cloud infrastructure, extracting dynamic service level agreement and translating them to smart contracts. Scheid *et al.* [16] 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. QoS accountable monitoring

Our proposal is based on the architecture detailed in AJIT [5] [3]. Such architecture employs a Just-in-Time Controller [3] to allocate network resources in a *Just-in-Time* fashion. We then exploit a permissioned DLT to create records of resource provisioning and to support service contracts as smart contracts deployed by operators. In our proposed architecture, operators act as nodes and participate in the consensus of the ledger, while customers requesting network resources are limited at sending service request transactions to deployed contracts through a **Distributed Application (DApp)**. Service providers advertise their service contracts, and the customer purchase them through the DApp.

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 can choose among the different offers based on the price advertised for a given QoS needed. As such, it is required to have a mechanism to clearly flag SLA infringements by both parties: operators failing to provide the promised QoS, and customers falsely claiming that 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. In the following, 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.

A. *Monitoring tools*

Each party keeps an independent record of the packets by maintaining a cryptographic accumulator [17] 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. Do note that 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 [17].

For efficiency reasons, we also propose to employ a state channel [18] 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, on the end, 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 in case of payment channels) 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 equivalent to being 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.

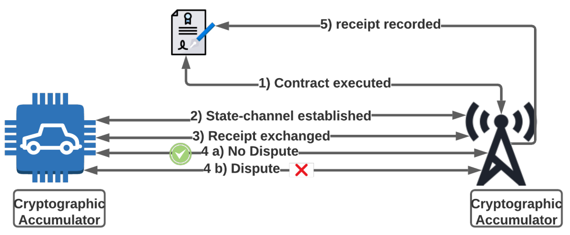


Fig. 1: Steps for state-channel based monitoring of QoS

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 empty 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 a 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 (Figure 1(4a)). 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 (Figure 1(4b)). This is then broadcast by any party to the wider DLT and the state channel is 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 computationally unfeasible 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 be efficiently checked by querying whether the packet 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.

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) [19] as cryptographic accumulators as they outperform traditional accumulators, in terms of operations speed [20] (see §IV).

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

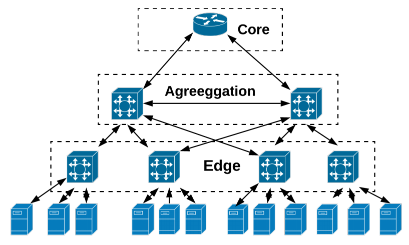
The above protocol can also be strengthened depending on the use case. As an example, Trusted Execution Environments (TEE) [21] can be employed to add the guarantee that customers can only consume packets they admit to have received. If we have a TEE trusted by the operator and deployed on the customer 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 to manage 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 deployability. This limits such examples to application scenarios where the advantages outweigh the costs. However, it 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.

IV. Is Just-in-Time Possible?

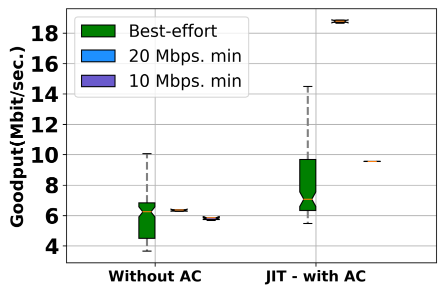
We evaluate the viability of our proposal on top of a topology (Figure 2(a)) that mimics in Mininet [22] a simplified version of the topology of a major mobile operator in the UK. To send instructions to the Mininet-emulated switches we built a ***Just-in-Time Controller*** atop Ryu Controller [23], 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 2(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 careful resource reservation are required to ensure a given service level requirement is met.*

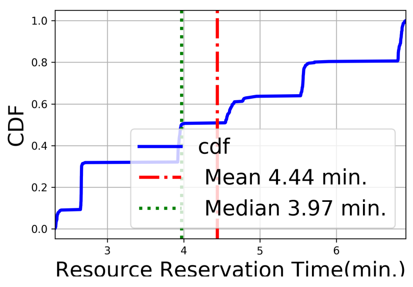
However, setting up a network slice and reserving resources comes with an overhead. Figure 2(c) shows a Cumulative Distribution of the time taken to setup and reserve resources over 350 service contracts with varying background service load. Both the mean and median are below $\approx$ 5 minutes. Conversations with a major UK mobile operator confirm that it is a realistic



(a)



(b)



(c)

Fig. 2: 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

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 operators follow a *"Blue-print"* approach, by grouping similar resource requests together and assigning them to a *pre-configured* set of network functions.

Apart from the time it takes to reserve network resources and set up a custom network slice to meet a given service level requirement, a smart contract has to be executed to record the agreed upon service contract, which will add to the set up time. To test this overhead, we deployed contracts through Hyperledger Burrow [24] on a local Hyperledger Fabric node 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. Figure 3(c) shows that the mean deployment time of a new service contract is 123ms and the average invocation time for a new service request is 119.64 ms, a negligible overhead in comparison with the several minutes taken to reserve network resources and setup network slices. In summary, *smart contracts add negligible (few milliseconds) overheads to the overall time for setting up just-in-time resource reservation and service contracts, and the full service contract can be put in operation on the order of a few minutes.*

After set up, the main constraining factor while the service contract is running is the overhead of maintaining per-packet records in accumulators. We adopt Secure Bloom Filter-based (SBF) accumulators, because of their efficiency as compared to their counterparts (i.e. cryptographic accumulators). We compare the performance of such accumulators with a more traditional RSA accumulator. Both accumulators are evaluated in two different compute infrastructures, meant to represent different kinds of customers. The first is a *server*-based customer, represented by an Intel Xeon CPU E5-2660 (2.60 GHz with 20 cores) and 94G RAM. The second is a more constrained customer (e.g., an IoT deployment), with *Raspberry Pi* Virtual Machine (1.5 GHz processor) with 8G RAM.

The primary constraint for QoS monitoring is the time required by resource limited devices to add packets to accumulators. Hence, we compared the add operations in both the SBF and RSA Accumulators using implementations from [25] for SBF, and [26] for RSA Accumulators. To each accumulator, 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.

We present our results in Figure 3(a) and 3(b). It can be seen that SBF is orders of magnitude faster than RSA accumulators. In fact, SBF only adds a few milliseconds of additional packet processing latency whereas adding each packet to an RSA accumulator takes several seconds, making the RSA accumulator completely infeasible for our use case. We also note that even the SBF benefits from the superior compute power of the *server*, which can add a packet to the SBF in one tenth the time it takes to add a packet in the *Raspberry Pi*.

V. Conclusion and Future Work

In this work, we proposed a method for automated monitoring of Quality of Service in an ongoing connection as a foundation for accountable network services. We believe accountability based on hard resource reservations is a key-enabler for 5G and beyond "mission-critical" services such as remote surgery or connected cars. A primary obstacle to hard resource reservation has always been the expense involved when resources have to be reserved for long periods of time over which a Service Level Agreement (SLA) is in force. We advocate a radically different alternative – to replace manual negotiation of SLAs with automated service provisioning using smart contracts running on Permissioned Ledgers to ensure a sustainable number of transactions. Our evaluation shows that it takes the order of a few minutes to set up a network slice and make hard resource reservations, showing that customers can request resources "just in time", i.e., just before they need the service, and also for a well defined short duration when they actually need to transfer data. This approach allows operators

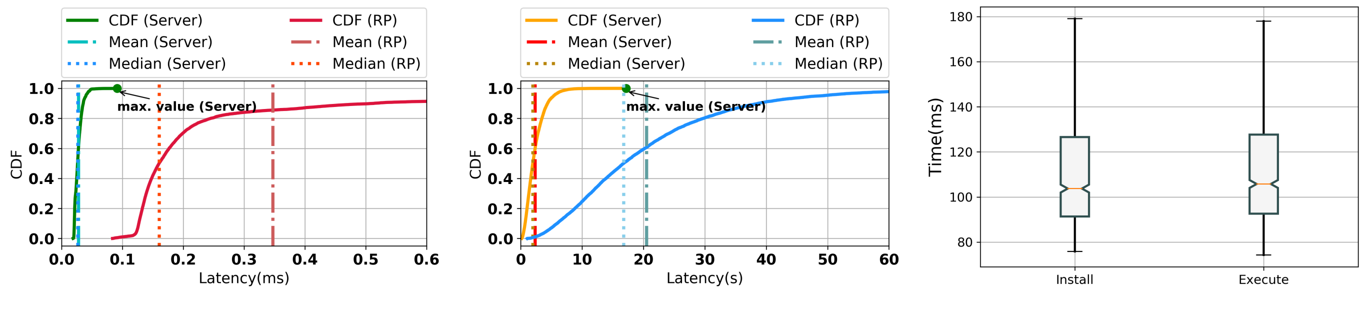


Fig. 3: Accumulator experimental evaluation for 1000 records. Cumulative Distribution Function (CDF) of single add operation on a) Secure Bloom Filter and b) RSA Accumulator. Note that, in a) the data is capped at 0.6 ms and in b) it is capped at 60 s to enhance readability. The maximum value in a) is ≈ 0.09 ms and ≈ 12.05 ms for the Server and Raspberry Pi respectively; and in b) the maximum value is ≈ 17.19 s and ≈ 261.66 s for the Server and Raspberry Pi respectively.c) Installation and Execution time of Smart Contracts on Hyperledger Fabric

to check whether they are able to commit to a particular service level request, and make hard guarantees based on resource reservations and an understanding of currently available capacity. We also showed how the agreed upon service levels can be monitored and enforced at run time, with acceptable overheads, by keeping per-packet records of the data transfer using cryptographically secure Bloom Filters.

This study embarks in 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] S. Baggioni, "Remote Surgery, Robotics and more - how 5G is helping transform healthcare," https://bit.ly/3oyCqKu, 2019, [Online; accessed 06-Oct-2020].
[2] Ericsson, "Connected Vehicles," https://bit.ly/2II4YC4, 2019, [Online; accessed 06-Oct-2020].
[3] T. Faisal, D. Di Francesco Maesa, N. Sastry, and S. Mangiante, "AJIT: Accountable Just-in-Time Network Resource Allocation with Smart Contracts," in *Proceedings of the ACM MobiArch 2020*, pp. 48–53.
[4] A. Sahai, V. Machiraju, M. Sayal, A. Van Moorsel, and F. Casati, "Automated SLA monitoring for web services," in *International Workshop on Distributed Systems: Operations and Management*. Springer, 2002, pp. 28–41.
[5] T. Faisal, D. Di Francesco Maesa, N. Sastry, and S. Mangiante, "How to request network resources just-in-time using smart contracts," in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*.
[6] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega *et al.*, "Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks," *IEEE Communications magazine*, vol. 55, no. 5, pp. 72–79, 2017.
[7] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network Slicing in 5G: Survey and Challenges," *IEEE CommMag*, vol. 55, no. 5, pp. 94–100, 2017.
[8] K. Samdanis, X. Costa-Perez, and V. Sciancalepore, "From Network Sharing to Multi-Tenancy: The 5G Network Slice Broker," *IEEE Communications Magazine*, vol. 54, no. 7, pp. 32–39, 2016.
[9] V. Sciancalepore, X. Costa-Perez, and A. Banchs, "RL-NSB: Reinforcement Learning-Based 5G Network Slice Broker," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1543–1557, 2019.
[10] K. Wüst and A. Gervais, "Do you need a blockchain?" in *2018 IEEE CVCBT*, pp. 45–54.
[11] B. Nour, A. Ksentini, N. Herbaut, P. A. Frangoudis, and H. Moungla, "A Blockchain-Based Network Slice Broker for 5G Services," *IEEE Networking Letters*, vol. 1, no. 3, pp. 99–102, 2019.
[12] 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 *2017 IEEE Internet of Things Business Models, Users, and Networks*, pp. 1–8.
[13] M. A. Togou, T. Bi, K. Dev, K. McDonnell, A. Milenovic, H. Tewari, and G. Muntean, "A Distributed Blockchain-based Broker for Efficient Resource Provisioning in 5G Networks," in *2020 IWCMC*, pp. 1485–1490.
[14] A. Alzubaidi, E. Solaiman, P. Patel, and K. Mitra, "Blockchain-based SLA Management in the Context of IoT," *IT Professional*, vol. 21, no. 4, pp. 33–40, 2019.
[15] R. B. Uriarte, R. De Nicola, and K. Kritikos, "Towards Distributed SLA Management with Smart Contracts and Blockchain," in *2018 IEEE CloudCom*, pp. 266–271.
[16] E. J. Scheid, B. B. Rodrigues, L. Z. Granville, and B. Stiller, "Enabling Dynamic SLA Compensation Using Blockchain-based Smart Contracts," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 53–61.
[17] N. Fazio and A. Nicolosi, "Cryptographic accumulators: Definitions, constructions and applications," 2002.
[18] S. Dziembowski, S. Faust, and K. Hostáková, in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 949–966.
[19] Z. Li and G. Gong, "On data aggregation with secure bloom filter in wireless sensor networks," *Technical Report, Dept. of Electrical and Computer Engineering*, 2010.
[20] A. Kumar, P. Lafourcade, and C. Lauradoux, "Performances of cryptographic accumulators."
[21] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted Execution Environment: What it is, and What it is Not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, pp. 57–64.
[22] Mininet, "Mininet: An Instant Virtual Network on your Laptop," http://mininet.org/, 2020, [Online; accessed 20-Nov-2020].
[23] R. Controller, "Ryu SDN Framework," https://ryu-sdn.org/, 2020, [Online; accessed 20-Nov-2020].
[24] HyperLedger, "Burrow," https://bit.ly/3nmcukT, 2020, [Online; accessed 12 Dec 2020].
[25] A. Kumar, "Python Bloomfilters," https://bit.ly/37Q3s9r, 2014, [Online; accessed 06-Oct-2020].
[26] O. Leiba, "RSA Accumulators," https://bit.ly/3oKEb7d, 2019, [Online; accessed 06-Oct-2020].