SECURING CONFIGURATION, MANAGEMENT AND MIGRATION OF VIRTUAL NETWORK FUNCTIONS USING BLOCKCHAIN

Igor Drummond Alvarenga

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Otto Carlos Muniz Bandeira Duarte

Rio de Janeiro
Março de 2018

SECURING CONFIGURATION, MANAGEMENT AND MIGRATION OF VIRTUAL NETWORK FUNCTIONS USING BLOCKCHAIN

Igor Drummond Alvarenga

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

Prof. Otto Carlos Muniz Bandeira Duarte, Dr.Ing.

Prof. Miguel Elias Mitre Campista, D.Sc.

Prof.ª Fabíola Gonçalves Pereira Greve, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2018

Alvarenga, Igor Drummond

Securing Configuration, Management and Migration of Virtual Network Functions Using Blockchain/Igor Drummond Alvarenga. – Rio de Janeiro: UFRJ/COPPE, 2018.

XII, 60 p.: il.; 29, 7cm.

Orientador: Otto Carlos Muniz Bandeira Duarte

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2018.

Referências Bibliográficas: p. 55 – 60.

1. Blockchain. 2. Network Function Virtualization. 3. Security. I. Duarte, Otto Carlos Muniz Bandeira. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

"To absent friends, lost loves, old gods, and the season of mists; and may each and every one of us always give the devil his due."
— Neil Gaiman, Season of Mists

Agradecimentos

Primeiramente gostaria de agradecer à minha família pelo seu apoio incondicional. Agradeço em especial aos meus pais, sempre presentes. Agradeço aos amigos pela motivação e companheirismo.

Agradeço também ao meu orientador, professor Otto, por todo apoio, dedicação, compreensão e paciência dedicados a mim desde que me uni ao GTA.

Agradeço especialmente a todos os amigos e professores do Grupo de Teleinformática e Automação pelo apoio e incentivo durante todo período que estivemos juntos. Também agradeço à Universidade Federal do Rio de Janeiro e a todos os demais professores que contribuíram para minha formação.

Por fim, agradeço ao Governo Brasileiro, em especial às agências de fomento CAPES, CNPq, RNP, FAPERJ, FAPESP e a fundação COPPETEC por terem investido em mim.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

PROTEGENDO CONFIGURAÇÃO, GERENCIAMENTO E MIGRAÇÃO DE FUNÇÕES DE REDE VIRTUAIS UTILIZANDO BLOCKCHAIN

Igor Drummond Alvarenga

Março/2018

Orientador: Otto Carlos Muniz Bandeira Duarte

Programa: Engenharia Elétrica

As tecnologias de virtualização de funções de rede e de encadeamento de funções de serviço de rede aumentam a agilidade na provisão de serviços e acrescentam inteligência no núcleo da rede. No entanto, a programabilidade do núcleo da rede e a oferta de serviços por múltiplos fornecedores provocam novas vulnerabilidades neste ambiente. A necessidade de provisão de funções virtuais de serviço de rede (VNFs) seguras torna-se ainda mais crítica, uma vez que uma simples modificação no núcleo da rede pode afetar múltiplos usuários. Este trabalho propõe uma arquitetura baseada em correntes de blocos para gerenciamento seguro, configuração e migração de VNFs. Esta arquitetura garante a imutabilidade, não repúdio e auditabilidade da configuração de VNF e do histórico de gerenciamento de VNFs. Além disso, a arquitetura proposta preserva o anonimato das VNFs, dos inquilinos e das informações de configuração, a fim de evitar que estes se tornem alvos de ataques. Foi desenvolvido um protótipo concebido para a plataforma OPNFV (Open Platform for NFV) e foi avaliado o desempenho em relação ao custo benefício de parâmetros e aos gargalos da arquitetura proposta.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

SECURING CONFIGURATION, MANAGEMENT AND MIGRATION OF VIRTUAL NETWORK FUNCTIONS USING BLOCKCHAIN

Igor Drummond Alvarenga

March/2018

Advisor: Otto Carlos Muniz Bandeira Duarte

Department: Electrical Engineering

The current technologies of network functions virtualization and network service function chaining increase service provision agility and add intelligence at the core of the network. However, the network core programmability and the provision of services by multiple providers brings new vulnerabilities to this scenario. The need for secure provisioning of virtual network service functions (VNFs) becomes even more critical, since simple modifications at the network core can affect multiple network users. This work proposes a blockchain-based architecture for secure management, configuration and migration of VNFs. This architecture ensures the immutability, non-repudiation, and auditability of VNF configuration and the management histories. In addition, the proposed architecture preserves the anonymity of VNFs, tenants, and configuration information, to mitigate the possibilities of targeted attack. A prototype designed for the OPNFV (Open Platform for NFV) platform was developed, and the proposed architecture performance was evaluated in terms of parameters trade-offs and bottlenecks.

Sumário

Lista de Figuras

Lista de Tabelas

Chapter 1

Introduction

Network function virtualization (NFV) and service function chaining (SFC) appear as alternative software-based technologies to allow commercial off-the-shelf hardware to perform functions previously delegated to proprietary hardware-specialized middleboxes [1]. The software-based approach reduces operational expenditure (OPEX) and capital expenditure (CAPEX), enabling multiple infrastructure providers, such as Internet service providers (ISPs), to offer customized end-to-end communication services. These services are composed by specific virtual network functions supplied by multiple virtual network function (VNF) vendors. Network function vendors also benefit of reduced time to market in order to offer specific virtual network functions, when compared to the long development cycle associated to hardware-based middleboxes.

In a network function virtualization (NFV) scenario, an Internet service provider (ISP), a user or an application may request an end-to-end network service with specific functionality. Then, this ISP establishes a customized chain of VNFs that provides specific functionalities for this user or application demand, selecting the appropriate VNF from several vendor implementation alternatives. These VNFs, actuallly, make part of a pool of VNFs that reside in NFV data centers near the network core, for which the ISP is a tenant. The use of programmable technologies at the network core, such as NFV and SFC, exposes the Internet service provider (ISP) to an increased number of vulnerabilities. Therefore, it is of major importance to reduce the possible VNF attack vectors and to provide secure and reliable configuration management [2]. It is worth to note that a threat in the network core affects a considerably greater number of traffic flows when compared to a localized threat at the border of the network, potentially affecting much more victims [3, 4]. Hence, the main challenge faced by this new service paradigm lies on ensuring VNF security. We can assume that, at the worst-case scenario, a compromised virtual network function at the network core endangers all traffic forwarded through this VNF [5].

To identify a faulty or compromised VNF, and how it came to be, the ability to audit VNF configuration and management history is mandatory and, as a consequence, non-repudiation and immutability of all previous VNF history are required. In addition, there is no simple solution for the immutability requirement using conventional databases [6]. Moreover, a proposed solution must consider that the network core and adjacent infrastructure providers are composed of high-demand carrier grade networks. Therefore, to be feasible in these networks, a solution must be able to adhere to strict quality of service guidelines, as carrier grade networks are expected to have at least 99.999% uptime, while still being able to handle tremendous amounts of system hand-offs, managing massive amounts of data and enforcing service level agreements (SLAs).

In order to achieve these goals, the dominant model employed for telecommunications network management the is fault, configuration, account, performance, and security management model (FCAPS) [7], as shown in Table 1.1. This model was developed by the International Organization for Standardization (ISO) and further promoted by the International Telecommunications Union telecommunication standardization sector (ITU-T) in the telecommunications management network recommendation on management functions (TMN M.3400) [8].

Tabela 1.1: FCAPS telecommunications network management model.

Area of function	Description
Fault Management	Defines methods and processes for fast recognition, isolation, logging and remediation of faults when they occur in a network.
Configuration Management	Covers processes for storage, recovery, change, management, and future planning of networks.
Administration Management	Identifies processes for tracking usage statistics and network utilization for billing of metered resources, as well as tracking roles and actions of credentialed users.
Performance Management	Ensures that networks perform as intended, validating compliance to assurance agreements.
Security Management	Defines guidelines and processes to ensure data, network, and access integrity.

In the last thirty years, FCAPS has become a proven approach for network management in a centralized, single-provider environment [9]. However, current networks function management paradigms, such as NFV, demand distributed workloads pertaining to multiple tenants across several shared data center environments. This raises the difficulty of providing a satisfactory VNF management security solution that tend to those requirements. We argue that blockchain-based repositories with specialized transaction schemes are a good solution to implement configura-

tion, administration, and security management in conformance with FCAPS model. Furthermore, a blockchain-based proposal should be able to facilitate distributing FCAPS capabilities across multiple shared domains [9, 10], by coupling its immutability property with suitable consensus and provenance tracing mechanisms.

This work proposes a two-fold blockchain-based architecture for the secure configuration management of VNFs, as well as to provide transparent and reliable interservice auditability to datacenter VNF management services. A configuration repository blockchain (CRB) is devised for tackling configuration management challenges, and a service management blockchain (SMB) is devised for storing management services requests and responses. Besides the immutability and integrity features provided by the use of blockchain, consistency of transactions is ensured by a proposed consensus protocol, which validates every transaction before registering it and keeps consistency intact even under byzantine faults. Finally, the adoption of an asymmetric encryption key identification scheme confers anonymity to VNFs and tenants, while still allowing access management, action tracking and sensitive information encryption when desired by the rightful parties.

The proposed architecture enables secure VNF configuration state migration, defining a trust mechanism between different infrastructure providers and VNF vendors, which may distrust each other. Furthermore, this architecture intends to cope with compromised systems, which could act maliciously. The proposed VNF configuration management and service logging mechanisms are compliant with the good practices found in the literature [7, 8, 11, 12]. In addition, the proposed architecture is designed to avoid any changes in the network function virtualization (NFV) and service function chaining (SFC) orchestration platforms, and it is not restricted to a predicted subset of VNFs. While the architecture accounts for signed configuration templates to be shared by VNF vendors as a configuration starting point, sensitive tenant configuration and management information is still kept private using encryption. More importantly, the proposed architecture eliminates the need for listening services in a VNF, thus allowing the definition of policies, which automatically enforce closure of all ports, reducing VNF exposure to possible attack vectors.

1.1 Contributions and publications

As a direct and indirect result from the contributions of this work, papers were elaborated on the subjects of blockchain, network function virtualization, service function chaining and data analysis.

1.1.1 Blockchain related papers

The following papers are direct results of the current work, and present early architectures for improving aspects of network function-virtualization security.

1. Alvarenga, I. D., Rebello, G. A. F., and Duarte, O. C. M. B. "Securing Configuration Management and Migration of Virtual Network Functions Using Blockchain", to be published in IEEE/IFIP Network Operations and Management Symposium - NOMS 2018, April 2018. English, A4 size, 9 p.

2. Alvarenga, I. D., Sanz, I. J., Rebello, G. A. F., Mattos, D. M. F., and Duarte, O. C. M. B. - "Gerenciamento, configuração e migração seguros de funções de rede virtualizadas utilizando corrente de blocos", Technical Report, Electrical Engineering Program, COPPE/UFRJ, July 2017. Portuguese, A4 size, 14 p.

3. Rebello, G. A. F., Alvarenga, I. D., Sanz, I. J., Andreoni Lopez, M., Mattos, D. M. F, and Duarte, O. C. M. B. - "SINFONIA: uma Ferramenta para o Encadeamento Seguro de Funções Virtualizadas de Rede Através de Corrente de Blocos", Technical Report, Electrical Engineering Program, COPPE/UFRJ, December 2017. Portuguese, A4 size, 8 p.

The first two blockchain related papers propose a blockchain-based architecture for the secure management and migration of virtual network function (VNF) configuration, while still relying on the network virtual function (NFV) platform to perform VNF management operations. These papers present a new approach to allow for auditability through immutability of configuration of VNFs in multi-cloud-vendor and multi-tenant VNF environment. The immutability is achieved by using blockchain with a consensus protocol acting as a mediator between the configuring tenants and the configured VNFs. In particular, the specific structure and method applied to the NFV environment are presented.

The paper entitled "Gerenciamento, configuração e migração seguros de funções de rede virtualizadas utilizando corrente de blocos"proposes a blockchain consensus protocol based on an election-less variant of the Raft protocol [13, 14] that is able to achieve fast transaction effectuation time, while still resistant to consensus member failures. The major shortcomings of this approach are the vulnerability to collusion and malicious behavior attacks, which are inherent to Raft-based protocols, and non-deterministic number of necessary message exchange rounds to reach consensus in case of major failure of consensus participants. The paper entitled "Securing Configuration Management and Migration of Virtual Network Functions Using Blockchain"improves on this shortcoming by proposing a practical byzantine fault tolerance (PBFT) [15] inspired blockchain consensus protocol that accounts

for collusion attacks and other types of malicious behavior, and is able to complete consensus in a fixed number of message exchange phases in all supported failure or attack scenarios. While the former demonstrates its proposed consensus mechanism through mathematical validation, in the latter the proposed architecture and PBFT-based consensus mechanism were prototypically implemented and evaluated regarding blockchain transaction performance.

The last paper proposes SINFONIA, a first attempt of a blockchain-based system that provides security to virtualized networks, ensuring auditability, non-repudiation, and integrity of VNF and SFC orchestration operations. SINFONIA is a modular stateless architecture that allows the orchestration of VNFs in a simple and agile way, offering a web interface front-end for intuitive manipulation of architectural facilities. A prototype of the proposed system for the Open Platform for Network Function Virtualization (OPNFV) was developed, with the implementation of a specific operation logging blockchain and a collusion-resistant PBFT-based consensus protocol. The results show that SINFONIA provides security with low overhead on the performance of the cloud orchestrator. The current work builds upon SINFONIA contributions to devise a blockchain-based mechanism to enable auditing of all NFV data-center management-service interaction.

1.1.2 NFV and SFC related papers

The following papers are indirect results of the current work, and present contributions on NFV security, NFV performance evaluation and security provision through service function chains (SFC). The studies on those papers were of utmost importance to the identification of the security and configuration challenges pertaining to NFV environments that were tacked in the present work.

1. Sanz, I. J., Alvarenga, I. D., Andreoni Lopez, M. E., Mauricio, L. A. F., Mattos, D. M. F., Rubistein, M. G., and Duarte, O. C. M. B. - "Uma Avaliação de Desempenho de Segurança Definida por Software através de Cadeias de Funções de Rede", published in Anais do XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg'17, Brasilia, DF, Brazil, November 2017. Portuguese, A4 size, 14 p.

2. Mauricio, L. A. F., Alvarenga, I. D., Rubinstein, M. G., and Duarte, O. C. M. B. - "Uma Arquitetura de Virtualização de Funções de Rede para Proteção Automática e Eficiente contra Ataques", in XXII Workshop de Gerência e Operação de Redes e Serviços (WGRS'2017) - SBRC'2017, Belém, PA, Brazil, May 2017. Portuguese, A4 size, 14 p.

3. Andreoni Lopez, M., Lobato, A. G. P., Mattos, D. M. F., Alvarenga, I. D.,

Duarte, O. C. M. B., and Pujolle, G. - "Um Algoritmo Não Supervisionado e Rápido para Seleção de Características em Classificação de Tráfego", in XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2017, Belém, PA, Brazil, May 2017. Portuguese, A4 size, 14 p.

The first paper presents a performance evaluation of different software-defined security-oriented service function chain topologies, and was fundamental for the understanding of emerging configuration and security issues on a NFV platform environment, highlighting the necessity of service and inter-service communication logging. This paper also investigates and proposes the use of intrusion detection systems (IDS) as virtual function. To this end, real time threat detection in live traffic is made possible by fast feature selection algorithms, such as the one we proposed in the last paper. While the proposed IDS architecture is effective in leveraging cluster power outside of service function chain to cope with real time traffic analysis, it has shortcomings in exposing an additional communication channel between the real time IDS cluster and the IDS VNF. Attackers through a denial of service (DoS) attack could potentially target this channel in an attempt to deny the correct operation of the IDS. The second paper furthers this understanding by proposing modifications to OPNF architecture in order to provide more efficient security, and specially raised awareness of issues that could arise if the attacker is able identify and target specific VNFs or systems. The improved architecture on the proposed work mitigates these issues by anonymizing VNF and services identity, as well as providing a secure indirect configuration mechanism that reduces VNF attack vector exposure. In special, DoS attacks are mitigated by the increased availability of configuration sources.

1.1.3 Data analysis related papers

The following papers are indirect results of the current work, and were paramount in ascertaining traffic behavior, attack distribution, and the increased threat reach as threats move from near access networks to the network core. Nevertheless, these research efforts unveiled a high necessity of providing fast-updating security network services near threat origin, to cope with newly developed threats before they could reach their intended targets. As NFV is a technology that employs software based network devices, the time to market to update security solutions is far shorter than the one for hardware middleboxes. Hence, NFV is a technology suitable to enhance security in the present threat landscape, providing flexible rapid-reacting real-time protection against network attacks even close to the network core.

1. Andreoni Lopez, M., Silva, R. S., Alvarenga, I. D., Mattos, D. M. F., and Duarte, O. C. M. B. - "Coleta e Caracterização de um Conjunto de Dados

de Tráfego Real de Redes de Acesso em Banda Larga", in XXII Workshop de Gerência e Operação de Redes e Serviços (WGRS'2017) - SBRC'2017, Belém-Pará, PA, Brazil, May 2017. Portuguese, A4 size, 14 p.

2. Andreoni Lopez, M., Silva Souza, R., Alvarenga, I. D., Rebello, G. A. F., Sanz, I. J., Lobato, A. P., Mattos, D. M. F., and Duarte, O. C. M. B. and Pujolle, Guy - "Collecting and Characterizing a Real Broadband Access Network Traffic Dataset", in 1st Cyber Security in Networking Conference (CSNet'17) - Best Paper Award - Rio de Janeiro, Brazil, October 2017. English, A4 size, 8 p.

1.2 Organization

The remainder of this work is organized as follows. Chapter 2 presents de Network Function Virtualization concept, the main aspects of blockchain data structure and associated technologies, then discusses the main consensus algorithms applicable to distributed blockchains. Chapter 3 presents the proposed architecture. Chapter 4 presents the implemented prototype evaluation results for the proposed architecture. Finally, Chapter 5 concludes the work.

Chapter 2

Network Function Virtualization, Blockchain, and Consensus Mechanisms

2.1 Network Function Virtualization

Network function virtualization (NFV) and service function chaining (SFC) appear as alternative software-based technologies to enable commercial off-the-shelf hardware to perform and replace functions previously delegated to proprietary hardware-specialized middleboxes [1]. The software-based approach reduces operational expenditure (OPEX) and capital expenditure (CAPEX) enabling multiple infrastructure providers to offer end-to-end communication services, and multiple virtual network function vendors to offers specific virtual network functions.

2.1.1 Virtual Network Function Security challenges

Previous works [1, 4, 5, 11] investigate the problem of security vulnerabilities due to co-location of multiple tenant virtual network functions (VNF) on the same service functions chaining (SFC) platform. They state that compromising a single VNF at the core of the network endangers entire service function chains and their network users. Firoozjaei *et al.* and Lal *et al.* investigate security threats and propose taxonomies for vulnerabilities in NFV [12, 16].

Several solutions are proposed in the recent literature for the problem of VNF secure configuration. Coughlin *et al.* propose the use of secure hardware through a Trusted Platform Module (TPM) to protect privacy of VNF configuration [17]. While achieving its purpose, this approach relies on specific hardware and centralized remote attestation, impacting up to 50 % of the available bandwidth. Massonet *et al.* propose an architecture for global configuration of security VNFs in federated

networks with automated deployment [2]. This architecture, however, is dependent on a centralized controller and on agreement between clouds, resulting in a single point of failure and configuration delays when in situations of great competition. A variation of this architecture is proposed by Massonet *et al.* in which the configuration of a security virtual network function (VNF) is performed by the cloud platform VNF orchestrator and configuration parameters are encoded in the service function description file in [18]. This approach requires modification of various components of the orchestrator and is restricted to pre-established types of VNF. Additionally, one cannot modify the security configuration of the VNF after it is booted by the same mechanism. Pattaranantakul *et al.* propose a similar mechanism to control access to VNFs through service function description files [19]. Reynauld *et al.* find, however, that such approaches render VNF configurations susceptible to cloud platform information access vulnerabilities [20].

Firoozjaei *et al.* and Lal *et al.* discuss the challenge of transferring configurations and states securely during VNF migration. Reynaud *et al.* point out that VNFs are built on software from different vendors, with different vulnerabilities, and mentions attacks that can be carried out against VNFs through privileged terminal access on cloud platforms[20]. The architecture we proposed is capable of performing VNF configuration and migration through a secure configuration repository, while preserving information confidentiality. Besides, the proposed configuration update mechanism eliminates VNF listening services, thus mitigating threats from terminals and from the network.

2.2 Blockchain

Blockchain related technologies are heavily reliant in cryptography, and first surfaced to enable anonymous, trustless monetary transactions [21], because even if parties do not trust each other, they may make deals. The usual path to secure a deal with one that is not trusted is to ask a common trusted party for intermediation. Nevertheless, this signifies the concession of some liberties, because the intermediation of a third party means that one may lose anonymity, privacy and time, moreover, one definitely loses control. Blockchain-enabled systems are trustless, which means there is no trusted party or intermediary to a transaction between two unknown and untrusting entities.

Several works explore the state of the art regarding the application of blockchains in communication network problems [22–24]. These works focus on the use of blockchain as a replicated incremental data repository, in which all past transactions are signed and recorded with asymmetric cryptographic keys. Xu *et al.* present blockchain as a mechanism of communication and service coordination th-

rough transactions while demonstrating its applicability as a repository of distributed information [6]. Boudguiga *et al.* present a solution for updating IoT devices through blockchain-stored information [25]. The architecture proposed in the present work uses similar mechanisms to configure VNFs. However, our proposal takes into account the needs for confidentiality, anonymity and auditing, not addressed by previous works. Hence, we propose a slightly modified blockchain to be used with a proposed practical byzantine fault tolerance (PBFT)-based algorithm.

2.2.1 Blockchain data structure

Blockchain is a term used broadly in the current literature to refer to applications, distributed systems, networks, databases, transaction schemes, and so forth. In fact, blockchain can be leveraged to create all of those technologies. However, ultimately, blockchain is a data structure in its original form, proposed by Nakamoto [21], a pseudonym, as part of his Bitcoin cryptographic currency system proposal. A blockchain is a linked list of smaller data structures, known as blocks. Each block is linked to its predecessor by pointing to its cryptographic hash, as depicted in Figure 2.1.

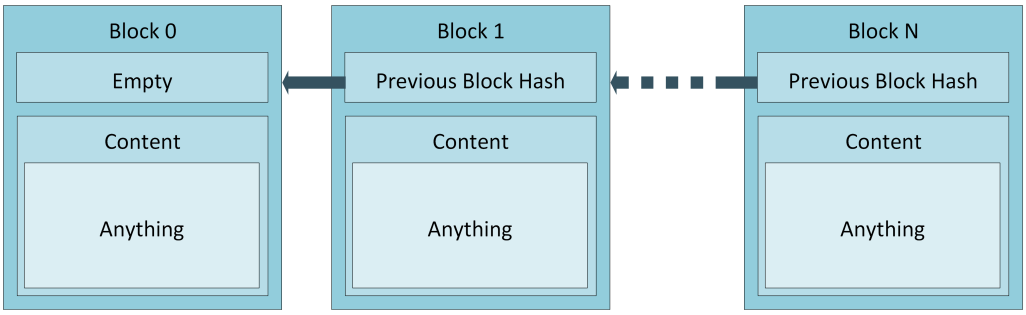


Figura 2.1: Standard blockchain data structure. This data structure functions as a linked list, in which each block is linked to the previous block by the previous block hash.

Besides the hash of the last inserted block, a block in a blockchain may have any content that makes sense to an application. There are many examples in the literature, for instance: Azaria *et al.* propose the use of blockchain to register and control access to medical records [26]; Frantz *et al.* [27] and Wood *et al.* [28] propose the utilization of a blockchain to store and execute contracts; Fujimura *et al.* propose the utilization of blockchain content block to store digital rights management (DRM) information [29]; Zyskind *et al.* propose the utilization of blockchain to store files [30]; and, Bozic *et al.* [23], Mukhopadhyay *et al.* [22] 2016, and Christidis*et al.* [24] propose storing virtual machine or network related resource information on blockchain.

The main type of blockchain content is the transaction, that represents an action that goes from one entity to another [31]. The original type of transaction, was the cryptocurrency exchange from Bitcoin [21]. But most of new blockchain application deal with some kind of exchange, that is registered on the form of a transaction. A transaction has four main parts: a sender, a recipient list, the content that is being transferred, and a signed hash of all preceding transaction fields. When the immutability property of blockchain is combined with a signed transaction, a system that works like a ledger is created. A ledger is an ordered immutable sequence of transactions that describe all deals made in a system from the last one, until the first in its conception. Signed hashes provide authenticity to each transaction claim, as well as non-repudiation and integrity to a transaction. Anonymity is obtained without sacrificing other properties, by utilizing the public counterpart of an asymmetric key pair as means of identifying the sender and recipient portion of a transaction. If the content of a transaction is classified and meant to be exchanged only between the sender and the recipient, the same key pair may be used to encrypt the transaction content, hence conferring privacy to the transaction.

After a block is validated and added to the chain, the preceding block becomes immutable, as any modification to a block other than the last breaks the chain of hashes that links the blocks. Consequently, the only operations allowed on the immutable part of a blockchain are to add a block, and to read the data stored in blocks. It is worth noticing that at this point one can remove blocks starting from the newest one or create a fork in the chain by pointing another blockchain to a block that already has a successor. The solutions to these pitfalls are replication and consensus. Replication ensures, besides eliminating single point of failure issues and providing availability of the blockchain contents, that there is a shared truth between blockchain system participants. This shared truth mitigates the possibility of unilateral modification of blockchain contents. As discussed by Saito *et al.* and Vukolić *et al.*, blockchain replication is a distributed consensus problem due to the need of consistent transaction ordering in a trustless public environment [32, 33]. Consensus is the mechanism that is used so that this shared truth may be reached, hence it is used to establish an agreement on the contents of a new block and their order, to prevent every participant in a blockchain network adds the exact same new block on their blockchain copy at the same time. A set of system rules is agreed upon by every participant of a blockchain-based system, in order to avoid that transactions may be validated in an unambiguous fashion. All the information on a blockchain can be verified and trusted solely relying in blockchain contents, without assistance or confirmation of any third party. This is a major advantage of blockchains when compared to other data storage systems, such as versioning systems or distributed databases [6, 24].

2.2.2 Blockchain-based systems

Blockchain-based systems implement a replicated state machine for the consistent preservation of a shared global state between distributed peers on a network. However, as discussed before, blockchain data structures themselves do not meet all requirements for its proposed usage scenarios. Systems built based on blockchain need to consider carefully all features and configurations to be implemented to the base blockchain data structure, as well as all available supporting technologies to complete a full featured blockchain system. Based on the taxonomy for blockchain-based systems for architecture design [31], and main researched blockchain applications [6, 24] we can pinpoint the main decisions defining an adopted blockchain architecture: centralization and configuration.

While there is no benefit of employing blockchain in a fully centralized network, there are varying degrees of possible decentralization. Centralization is the main defining aspect of a blockchain system, and can be classified in three main types:

- **Partially centralized**: This type of centralization is also known as permissioned blockchain with fine-grained permission. Every operation is controlled, including who can write to the blockchain, who con read from the block chain, what types of information one is authorized to read, what types of information one is authorized to write, and who can participate in the consensus and write blocks. In some cases, a special party may be elected to sign, authorizing, specific types of transaction.

- **Partially decentralized**: This type of centralization is similar to the partially centralized blockchain in regards to consensus and, sometimes, writing permissions. This type of blockchain, however, is permission-less in regards to reading operations.

- **Fully decentralized**: This type of centralization is also known as permissionless blockchain, which means no authorization is required to perform any operation or role.

Configuration of a blockchain system defines finer aspects of its management and operation. The main configuration decision are on the blockchain scope, anonymity and incentive. Regarding blockchain scope, a blockchain can be:

- **Public**: A public blockchain can be accessed by everyone in the Internet, and anyone is free to join the network and fully participate in all available system features.

- **Federated**: A federated blockchain defines its rules and participation decision by a managing consortium or federation. A consortium is defined as group

of collaborating organizations, a federation is defined as a specially created management organization to oversee a blockchain.

- **Private**: A private blockchain work in the confines of a single organization and the general public has no access to stored information.

Anonymity is a configuration decision that defines the manner a user of a blockchain system is identified. Most blockchain implementations use the public counterpart of an asymmetric key pair as a form of identification. This kind of identification allows one to authenticate transactions while retaining anonymity, as well as to replace the last identity should it be compromised. Nevertheless, there is the possibility to tie a subset of public keys, or even all of them, to real identities by means of public certificates, or identity to public key dictionaries maintained by a federation or consortium in case of a federated blockchain. Such an approach is essential when there is the need to avoid duplicated actions by the same individual on a system, such as in blockchain assisted elections [34].

Incentive is a configuration decision with the main objective of attracting participants to aid in blockchain replication, validation and consensus that are essential to maintain a blockchain-based system. Incentive is necessary when these activities, or the utilization of the system, would not otherwise provide any direct benefit to the ones performing them, such as in cryptocurrency blockchains [21, 28, 35]. Public permission-less systems usually are based on incentives, while private or federated blockchains do not need incentives as the system existence in itself offer benefits to the ones managing it [24, 30, 31, 36].

2.3 Blockchain consensus mechanisms

Regarding the consensus protocols utilized by blockchain, there are two main classes of consensus protocols, eventual consistency protocols and quorum based consistency protocols [24, 31, 37]. Eventual consistency protocols aim at providing a secure consistency mechanism that works without knowing the total available number of consensus participants, while quorum based consistency protocols requires a list of participants to enable consensus. Hence, eventual consistency protocols are well suited for public permission-less networks, while quorum based consistency protocols are well suited for private and federated networks [37]

2.3.1 Distributed agreement challenge

Consensus is the process by which a group of communicating systems reach distributed agreement. The main challenge lies in that there is no distributed computational

system that is completely safe from failures or misuse [33, 38, 39]. The consistency-availability-partition (CAP) theorem [38] proves that one can not have consistency, availability and a network partition at the same time. Consistency means that the same information will be retrieved for the same request to any distributed system participant. Availability means that one distributed system participant will always answer a request. Partition tolerance means that the system stays working even if the distributed system participants are split in disconnected groups due to a network failure. As the theorem states, when designing a distributed system, for when a partition occurs, one must decide between consistency or availability. When deciding on consistency, the entire system must time out and not respond, as it will not be sure of the last state when answering due to loss of communication to all its peers, hence forgoing availability. When deciding on availability, any system participant will answer a request with the most recent known data, even if it is not sure if there is a newer version, hence forgoing consistency. In the absence of a partition, both consistency and availability are satisfied.

The partition then availability-consistency else latency-consistency (PACELC) theorem [40] states that when there is a network partition, the CAP theorem applies, else the distributed system designer must decide between latency and consistency. This happens because replication is not instantaneous. Hence, if a system must respond promptly at all times, it must forgo consistency, else a received request must wait consensus to be reached before answering, therefore, creating latency.

2.3.2 Eventual consistency

Eventual consistency means that, without any new transactions, any participants eventually maintains the same blockchain in its storage [35]. Eventual consistency protocols do not rely in any information about total participating members of consensus, they are based on forwarding their decided known truth to other participants, while keeping track of possible truth and deciding between them based on a set of agreed upon rules. In blockchain case, it means generating or receiving a block that carries current known transactions, validating it against the network agreed upon validation rules, and forwarding the validated block to other participants of the network. When there is more than one valid block, this is called a fork in the chain, and the participant keeps track of all possible chains until a new block that solves a tiebreaker arrives. The most common rule to decide which block is correct is the first known block rule, and the most common tiebreaker rule is that the longest chain is the correct one [21, 24, 28]. Which means that an eventual consistency blockchain participant will append the first block they see to their local blockchain, consider it correct and forward it, but will keep track of other possible chains. When a chain

is longer than all other by a a number of blocks, the participant discard other chain known and keep the decided one as their current chain.

In blockchain eventual consistency systems, the most common type of block creation algorithm is the proof-based [31, 35]. A proof-based algorithm relies in a derived proof of expending a local resource to find a valid block. A valid block is one that conforms to the agreed upon blockchain rules, points to the last known valid block in the chain, has a valid list of not previously recorded transactions, and solves a lottery based resource spending challenge [37]. This challenge does not depend on any external information besides the last version of the blockchain. Blockchain eventual consistency systems usually set a desired mean consensus time, and adjust the difficulty of the necessary challenge based on the recorded solution times of an agreed upon number of blocks. Participants need to increase or decrease the challenge difficulty in a deterministic way to reach the target value. Once the challenge is cleared, the created block is submitted together with the related proof to all locally known participants of the network. Any participant may attempt to generate a new block at any time. Eventual proof-based consistency mechanisms usually reward the participant that succeed in creating a block appended to the chain with some sort of cryptocurrency, to account for the expended resources and incentive continued participation in the system consensus. This cryptocurrency value can be fixed, or complemented by transaction emitting participants as a way to offer incentive for their transaction inclusion on the current block. This incentive is necessary as the block size has a maximum value, and the consensus participant has the autonomy to decide which transactions to include.

Most of the proposed challenges in the literature for blockchain proof-based consistency mechanisms are based on the original challenge from Bitcoin proposal [37], the proof-of-work challenge [21]. Proof-of-work consists in adding a nonce field to block data, and then trying to find a random string, which results in a block hash starting with a set number of zeros. This number of zeros is the target difficulty. This is a brute force process deemed to be completely aleatory, hence the lottery comparisons, and gets extremely costly as more nodes compete for rewards. Other proof-based algorithm build upon this idea with the objective of reducing energy consumption by decreasing challenge difficulty according to a new expended or possessed resource, or replacing the challenge completely with the new resource. A non-exhaustive list of new proposed resources can be found on Table 2.1. There is often more than one challenge type proposed for each resource in the literature [41].

Eventual consistency is the predominant consensus algorithm for public permission-less blockchains, because proof-based consensus does not care about participant number, as such, in relation to PACELC theorem, it is not possible to determine when there is a partition on the network. Hence proof-based consensus results

in an always available system that is consistent only eventually and has no response delay.

Tabela 2.1: Proposed resources for new proof-based protocols.

Protocol	Resource	Description
Proof-of-Stake	Currency	The amount of currency a participant has on the blockchain. A common security measure to avoid abuse, such as producing blocks for different forks at the same time, is to remove part of the staked currency in case of misbehavior.
Proof-of-Deposit	Currency	Just as proof-of-stake, but the currency is locked while the participant is partaking in consensus.
Proof-of-Burn	Currency	Just as proof-of-stake, but instead of depositing currency, this resource is measured by the total amount of currency destroyed for this purpose. Currency is destroyed usually by sending it to an invalid or null address.
Proof-of-Coin-age	Wighted currency	Just as proof-of-stake, but currency possession is weighted by their age.
Proof-of-Capacity	Storage	The amount of disk space dedicated by the participant to allocate chain information. It is suited to systems where the blockchain points to associated large data sets, which are not necessarily copied to all participants.
Proof-of-Elapsed time	Computational	Uses Intel software guard extensions (SGX) hardware-enabled devices and processors in order to generate a wait-time-proof. The smallest the wait-time-proof, the better. Intel proof generating mechanism is designed to be tamper-proof[1].

2.3.3 Quorum-based consistency

Quorum based consistency is defined by the necessity of reaching an agreement of a certain percentage of consensus participants before they all modify the local replicated copy of the system data. Which means that every consensus participant is known by all others, and each one votes to agree or to refute an all proposed modifications to the system data. Quorum based consistency is designed to provide sequential consistency at standard operation, which means that the registering of information happens at the same time and in the same order for all replicas, hence all replicas will respond the same way to the same request at all times. In case of a failure, or malicious behavior of a consensus participant, the sustained consistency type depends on the consensus algorithm.

[1]Hyperledger Sawtooth. https://intelledger.github.io/introduction.html

Common fault tolerance

Paxos [42] and Raft [13] are equivalent protocols that aim to manage a replicated log on data entries. The main idea of those protocols is to perform a leader election, so that all system data-modification proposals are sent to this leader, who shares the modification with all others and ask for their opinions. Then the leader shares the collective decision with all other consensus participants and, everyone performs the write operation at the same time. Both protocols account for all possible instances of participant or network failure, in order to ensure consensus, and are resilient to N failures in $2N + 1$ total participants [13]. To this end, the protocol assumes a controlled environment where the following set of assumptions hold true:

- Each participant operates at arbitrary speed.
- Each participant may experience failures.
- All participants can exchange messages.
- Messages are asynchronous and may take arbitrary time to be delivered.
- Messages may be lost, reordered or duplicated.
- Messages are delivered without corruption.
- All participants act on their best effort to perform consensus.
- All participants are honest.

Fault-resistant consensus protocols are designed to be used in trusted environments, as they account for consensus participant failures, but not to consensus participant malicious behavior. Hence, they are only suited to private blockchains where the consensus participants belong to the same organization and are trusted at all times [37]. In relation to PACELC theorem, Paxos and Raft based blockchain systems prioritize consistency above all else.

Byzantine fault tolerance

The Byzantine generals problem, is an agreement problem proposed by Lamport *et al.* in which a group of generals, each one responsible for the command of a portion of the total Byzantine army prepare to attack a city [42]. Each general has the option to attack or retreat, and the group of generals must reach a common decision, as an uncoordinated attack or retreat is the worse outcome. The problem must take into account that some of the generals could be traitorous, working alone or in groups. They may selectively lie to each other in order to disrupt the decision process. Furthermore, the problem is complicated by the fact that all communication is

performed by messages delivered by a courier, who also cannot be trusted. This problem translates to computer science as a consensus problem in which, assumptions on the environment are changed so that [15, 39, 43]:

- Each participant operates at arbitrary speed.
- Each participant may experience failures.
- All participants can exchange messages.
- Messages are asynchronous and may take arbitrary time to be delivered.
- Messages may be lost, reordered or duplicated.
- **Messages can be delivered with corruption or arbitrary modification by a third party**.
- **Any participant may act maliciously and lie in a selective way**.
- **There is no way to ensure that a participant is honest**.

Bizantine fault tolerant (BFT) protocols solve this in a way that is similar to common fault tolerant protocols in their default case, but the leader role is diminished as all consensus participants send their confirmation to each other, and then share all received responses with all consensus members [15]. Moreover, all exchanged messages are cryptographically signed. BFT protocols also require that all participant network addresses, as well as their public keys, are known to each other. Castro and Liskov proposed a practical byzantine fault tolerance (PBFT) protocol, an asynchronous BFT protocol suited for state machine replication [15], which provides high efficiency in the default case when compared to eventual consistency protocols. PFBT is expected to perform hundreds of write operations and thousands of read operations per second, while reaching consensus with hundreds of participants in a few seconds [33]. PBFT protocol is able to tolerate Byzantine failures of N nodes from a total of $3N+1$ nodes, because, even if N nodes are malicious, it is still possible to reach a quorum between the remaining $2N+1$ consensus participants. The PBFT default operation case is depicted in Figure 2.2, and the following steps are performed:

1. **Request**: A new request is sent by a client to a consensus participant. If it is not sent to the current consensus leader, it is forwarded to the leader.
2. **Pre-prepare:** The leader, known as primary participant, informs all replicas of the request.

3. **Prepare**: Each participant answer the request based on local knowledge and inform their answer to all other participants using signed messages, that includes the received request.

4. **Commit**: When more than two thirds of participants agree to the new request and answer, they send a signed commit message including all received signatures to all participants.

5. **Reply**: When more than two thirds of all signed commit messages are verified, the request and answer are appended to the consensual replicated request database. Then every consensus participant sends the request answer to the client.

6. Post-consensus, the client accepts the result if more than two thirds of consensus participants return the same result.

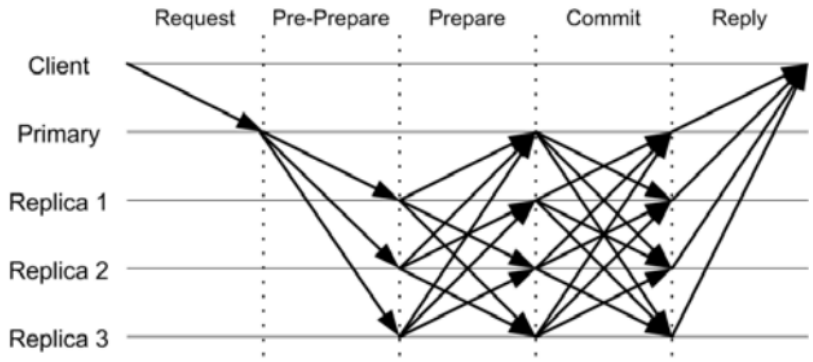


Figura 2.2: Sequence of messages/phases, from left to right, for the default case of the Practical Byzantine Fault Tolerance (PBFT) consensus protocol.

As federated blockchains are designed to be under the control of a consortium or a federation, they have means to identify the consensus participants, and to control writing rights of its permissioned users, hence federated blockchains need to address many of the common problems of distributed computing systems. As the target environment is trustless, the most prominent category of problems related to federated blockchais lie in the BFT research field, that has been active over a decade [37]. Federated blockchains can use many well known techniques to reach consensus, broadcast transactions or replicate states in asynchronous error prone environments. This suitability between federated blockchains and BFT protocols has sparked new research, and BFT protocols are specially well suited to leverage the asymmetric key infrastructure commonly found in blockchain architectures [33]. Schwartz *et al.* propose the Ripple protocol for distributed consensus in federated blockchains [36].

This protocol is byzantine fault tolerant (BFT) and achieves robustness against collusion attacks by creating trust zone subsets where collusion is not expected to occur. However, the use of Ripple presents scalability problems regarding the number of nodes designated to reach consensus [37]. Miller *et al.* propose a novel asynchronous BFT protocol that achieves low-latency consensus with higher scalability [44], but it still lacks a compliant computational implementation [37].

The consensus protocol adopted in our proposed architecture is based on the PBFT protocol, with modifications to account to blockchain properties. We also propose slight modification to the traditional blockchain data structure, to better leverage PBFT protocol capabilities and to confer auditability of past consensus decisions. The modified blockchain data structure is depicted in Figure 2.3. The main difference of the proposed blockchain data-structure and the traditional one depicted in Figure 2.1 is that the blocks are now linked by a signed hash of the content part of the structure, performed by consensus participant identified by its public key in a hashed content field. The default operation case phases of the proposed PBFT-based blockchain consensus protocol are the following:

1. **Request**: Transactions are sent by blockchain clients to a consensus participant. If it is not sent to the current consensus leader, it is forwarded to the leader. These transactions are accumulated by the consensus leader in a pending transactions list. Different from PBFT, the request phase in our PBFT-based consensus happens asynchronously in relation to the other phases.

2. **Pre-prepare:** After waiting a fixed time since the last concluded consensus, if there are pending transactions, the consensus leader creates a block with all valid transactions received. This block points to the signed hash of the current blockchain last block. The leader sends the new proposed block to all consensus participants.

3. **Prepare**: Each participant validates the transactions within the block, as well as the block structure locally, and then inform their decision to all other participants, using signed messages containing the signed hash of the received block content part.

4. **Commit**: When more than two thirds of participants agree to the new block, they send a signed commit message including all received signed hashes and decisions to all participants.

5. Post-consensus, if more than two thirds of all signed commit messages are verified, the commit messages content is registered on the block header proof

of acceptance field, and the the new block is appended to the local blockchain of consensus participants.

Our modified blockchain PBFT-based consensus eliminates the reply phase. The block just needs to be locally stored in the blockchain, and leverages the new introduced signed hash scheme as a means of tracking accepted block consensus history and reducing data exchanged in pre-prepare phase. Just the signed content hash is enough to ensure that everyone received the same block. Later, when a transaction is retrieved by a client, this client can be sure if the transaction is correct and authentic without further confirmation, because the transaction is signed by its sender.

In case of a network partition, if the partition does not segregate more than a third of participants, the consensus protocol can still be performed. When the segregated participants come back, the request of a blockchain copy from any consensus participant is sufficient for all previous validation to be performed locally. Moreover, as soon as the returned participants receive the first block proposal, they can verify if their current blockchain is correct. Regarding PACELC theorem, the proposed system always decides on consistency, introducing a little delay to transaction registration due to consensus. Nevertheless, leveraging blockchain immutability property, even on network partitions which partially stopped consensus, there is the guarantee that if a read request can be answered, then the answer is always consistent. On partition that stop the consensus on every remaining connected group, the system responds consistently to all requests, even if it cannot write.

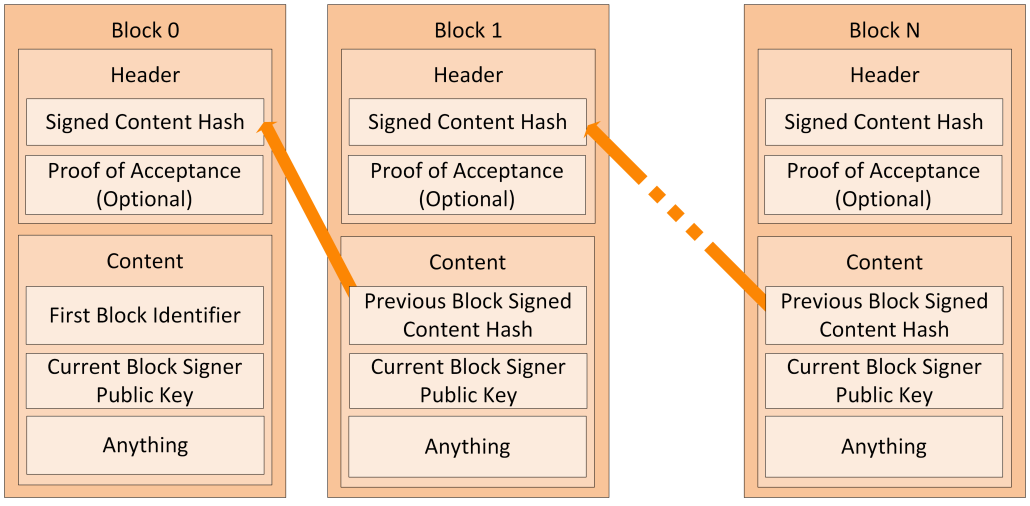


Figura 2.3: Proposed signed blockchain data structure. The main difference is that signed content hashes now link the blockchain. The header field may carry a proof of acceptance of the block by other consensus participants than the current block signer.

Chapter 3

The Proposed System Architecture

This work proposes an architecture that focuses on secure virtual network function (VNF) configuration and inter-service communication logging. We consider a scenario in which a user or application requires a network service to a network service provider, an Internet Service Provider (ISP), in a network function virtualization environment. This tailored specific service is composed, on the fly, by a chain of virtual network functions (VNF), using service function chaining technology. The virtual network functions (VNF) are offered by multi-vendors and reside on data centers. Our proposal makes assumptions and defines requirements.

3.1 Assumptions and requirements

Our architecture assumes that the network function virtualization (NFV) data center provider, in which the virtual network functions (VNF) reside, is not malicious. Actually, we can not provide security when an owner of a data center is malicious and compromise the virtual network functions that are hosted. This is a common assumption for cloud systems. Nevertheless, an attacker could compromise the NFV-data-center management services or other systems under that provider. It is also assumed that the infrastructure providers, tenants of the NFV data center provider, build their computational structure in a shared, but isolated, virtual environment.

In order to the proposed architecture confer the desired security and features, each virtual network function (VNF) configuration update request and each NFV management service request is required to be:

- confidential, to not expose nor reveal confidential information or facilitate the exploitation of known vulnerabilities by an informed attacker;
- authenticated, to enable verification of sender request permissions;

- anonymous, to not expose VNF or tenant identity that can facilitate targeted attacks;
- permanent, to keep historical records to enable auditability in case of an attack or dispute;
- immutable, to guarantee that the subject data has not been tampered, beacause auditability is only meaningful if there is verifiable proof;
- traceable, to pinpoint an event of interest in the overall sequence of events, as well as the ability to identify the related parties involved.

In short, we need a management system with a generic data repository that presents the above mentioned desired properties. Cryptographic signature and cypher schemes provide the confidentiality and authentication requirements. The anonymity requirement can be achieved using a private counterpart of an asymmetric key pair as an identifier decoupled from party identity, while still maintaining the first two requirements. The permanence requirement implies disabling deletion operation over registered, and, analogously, immutability requirement implies disabling updating operations over the same data. Traceability over an immutable and permanent data repository is achievable by the careful recording of the time, the order and the ownership of each recorded request. Hence, in a first analysis, one can conclude that a generic data repository with only the data inclusion and data reading operations, coupled with a careful request recording scheme is enough to fulfill these requirements. Nevertheless, even though common data repositories such as distributed databases or versioning systems could be properly configured and administrated to confer permanency and immutability properties, that implies trust in a third party to maintain this data repository. That also means trust in that third party that when a record is retrieved from this data repository it is the unadulterated original record, and when it is missing, it never existed. Hence, trust is essential for these common data repositories, as there is no way to verify that the required properties are met at ones discretion without believing in the third party [6, 24]. This result in a series of serious security risks. For instance, this third party is able to remove records at its discretion, or to allow an specific author of a request to modify previously recorded information. Those modifications would go unnoticed by any user of the system. If a third party is found to be compromised or misbehaving, all trust in previous recorded information would be lost.

This work solves the trusted party problem by completely removing the necessity of trust. Hence, a trustless approach using blockchain-based data repositories is proposed. Blockchain is a data structure that is able to solve immutability and permanency requirements in an easily verifiable way to anyone that can read the

data structure. Furthermore, this work proposes a transaction model that employs asymmetric keys to provide anonymous authentication [21] of tenants, VNFs and services, as well as confidentiality of configuration data through encryption. When combined to a blockchain data structure, this signed transaction context intrinsically provides traceability, as well as non-repudiation, of all recorded information. The proposed architecture takes into consideration that some parties need to have their identities disclosed, such as the NFV data center management services and configuration template proponents.

The proposed blockchain data repositories are linked lists of blocks that act as a log of ordered entries, best known as transactions. The signed result of a hash function uniquely identifies a block and guarantees integrity of its contents, which includes the preceding block signed hash. The blockchain data structure achieves immutability of past stored block order and block contents due to the use of cryptographic hash linking. Thus, this blockchain-based proposal guarantees the non-repudiation property by combining the immutability property with signed transactions, which ensures traceability and accountability requirements. To guarantee availability, this work replicates the blockchain in several locations. Moreover, to provide resiliency against faulty systems and collusion attacks, a consensus protocol is used to validate every transaction before storing it in a block of the blockchain, keeping consistency intact even under byzantine faults.

In order to mitigate threats to the VNFs, all VNF listening services are disabled, requiring a proactive VNF configuration model, i.e. the VNF initiates all requests for configuration update. This feature eliminates open ports, avoiding connection from attackers to a VNF. Furthermore, VNF access terminals are also completely disabled or used solely as unclassified anonymous displays of simple information. Therefore, there is no possibility of direct configuration input and all configuration updates must be retrieved by the VNF at fixed intervals from a trusted configuration repository. This secure repository must accommodate both configuration update requests and configuration templates.

The adopted consensus algorithm defines criteria on which consensus participant closes a block at each round and the manner transactions are distributed and validated. Public blockchains, such as the one for Bitcoin virtual currency, usually employ proof-of-work (PoW) based approaches [21], which rely on verifiable expenditure of computational power to solve a cryptographic challenge. This approach is ideal for public networks with an unknown number of participants and complete lack of trust. Proof-of-work (PoW) based approaches, however, imply too much computational power and may take a long time to finally register a transaction [32, 35]. Moreover, PoW approach is based on rewarding the consensus participants that closes a block for its computational power expenditure, as a means of sustaining interest

in voluntary consensus participation. Our proposal is based on a federated private blockchain model, as such, it does not rely on virtual currency, rewarding participants is not necessary. In addition, a number of blockchain applications requires strict transaction registering latency and, then, a different consensus mechanism is needed. Private and federated blockchains (i.e. blockchains owned by a single party or multiple collaborating parties) may relax trustlessness requirements by assuming its set of participants is known [24].

The proposed architecture assumes and implements a federated consensus model where, consensus participants are known, and a certificate issuing party, previously agreed upon by all the federation members, certifies their asymmetric key pair. The literature on the aforementioned topic has shown that byzantine fault tolerant (BFT) protocols enable low-latency consensus for applications up to a few hundred validating participants [33, 44] and accept erratic or malicious behavior from up to one third of the consensus participants. PBFT consensus protocol uses a list of agreed upon neighbor addresses and public keys, which is provided by certificate issuing party. The cost of neighbor list modification depends on federation governance rules. The implemented consensus protocol is inspired by the PBFT (Practical Byzantine Fault Tolerance) protocol and is modified for the blockchain consensus case. A PBFT-based consensus enables a blockchain-based system to be robust against collusion attacks performed by up to one third of consensus participants. In addition, the PBFT protocol leverages the consensus leader participant key pair to sign closed blocks in a way that is easily verifiable by other participants. For the blockchain use case, all transactions sets received by all participants during a given interval are forwarded to the current elected leader, which closes, signs and proposes a new block. There is no reply phase in blockchain consensus.

3.2 Attacker model

This architecture considers a Dolev *et al.* attacker model, that is, the attacker is able to `read`, `send`, and `discard` a transaction addressed to the blockchain, or any packet of the network [45]. The attacker can act passively, by connecting to the network and capturing all message exchanges, or actively, by injecting, repeating, filtering or exchanging information. Tenants, VNFs, the blockchain itself, and the network can be attacked.

Blockchain attacks are an attempt to prevent a legitimate transaction or block from being embedded in the blockchain. In order for a blockchain attack to succeed, the attacker must control a significant portion of the network to affect the consensus algorithm. This type of attack is mitigated by the adopted consensus mechanism. Attacks that try to modify or corrupt a transaction are not possible because a

correspondent signed hash accompanies every transaction.

Attacks on tenants or VNFs consist on attempting to obtain either configuration information or personification of the target. Personification attacks are not possible because every transaction sent to the blockchain is signed by its issuer. Attacks that seek to obtain configuration information are mitigated by sensitive information encryption, where the attacker needs to obtain the private key of the targeted recipient. This work does not address the case where a tenant or VNF was compromised through tenant terminal invasion, or tenant stored key pair hijacking. However, the proposed architecture mitigates attack vectors by avoiding any active listening services in a VNF, and by only allowing use of a terminal in read-only mode [46].

In addition, the proposed architecture permits auditing of all past transactions at will. Therefore, if an attacker tries to modify the blockchain using stolen key pairs, the attempt will be logged. Upon discovery of an incident, the stolen key pairs can be easily replaced by the tenant, reestablishing security and preventing further damage.

Network attacks represent the attempt to isolate a single tenant, a group of tenants or a group of VNFs from the network, thus preventing the network from performing transactions or reading content from the blockchain. This attack category contemplates classic network attacks which can be mitigated by establishing redundant paths between the distributed blockchain and VNFs or tenants. The proposed architecture assumes a redundant public network, e.g. the Internet, interconnects all participants. The aforementioned assumption hardens single entity targeting if the attacker is not in its adjacent network. Complete mitigation of network attacks is outside the scope of this work. The proposed architecture focus on blockchain attacks and anticipated transactions. However, by eliminating listening services in VNFs, this architecture eradicates application-layer denial-of-service attacks, which are a common threat in shared cloud environments [5].

3.3 Proposed architecture modules

The proposed architecture is composed of two types of modules:

- **Blockchain modules**, responsible for building a replicated trusted repository fabric of historical information pertaining their roles.
- **Client modules**, that enable the retrieval of information from blockchain modules, and the issuing of transactions that add information to a blockchain.

Considering module interaction, the proposed modules could be organized in two pairs, one for each proposed blockchain, and a client module that connects to both

blockchains. Each module pair has a specific transaction scheme tailored to the functional needs of the pair, and the client module is compatible with both transaction schemes. There is the configuration-repository blockchain (CRB) module, that is paired with the VNF client module to perform the configuration management roles in the proposed architecture. And there is the service-management blockchain (SMB) module, that is paired with the service client module to perform NFV data center management and auditing roles in the proposed architecture. Finally, the tenant client module holds the tenant facilities necessary to interact with both blockchains and perform the tenant administrative roles.

The proposed modules are composed of smaller components, such as functional blocks, data stores, layers and services. Functional blocks perform a specific function set. Data stores are used to preserve data, such as databases or blockchains. Layers perform data encapsulation, decapsulation, conversion, and transmission. Services are structures that exposes discrete units of functionality that can be accessed by other layers and acted upon independently. All modules share a set of common components:

- **RPC/TLS communication layer**: This layer performs inter-module communication using remote procedure calls (RPC) via transport layer security (TLS) protocol.

- **Encoding/Decoding layer**: This layer performs encoding and decoding of data sent via the RPC/TLS communication layer. Data is exchanged using binary JavaScript object notation (BSON) encoding, that is suitable for signing and hashing operations because it always has the same exact binary representation.

- **Blockchain query service**: This service exposes blockchain querying functionalities that enable other layers to access blockchain-stored information. When associated directly to a blockchain and index database, it will answer the queries promptly, otherwise it will forward the request to a known blockchain module.

- **Key pair management**: This functional block manages the key pairs associated with the current module instance, and performs all hashing and signing operations. The architecture is planned in such way that this is the only component that needs to be adapted if there is the need to use physical security like trusted platform modules or smart cards.

3.3.1 Blockchain modules

The configuration-repository blockchain (CRB) and the service-management blockchain (SMB) modules have the same structure, illustrated in Figure 3.1. A blockchain module is responsible for hosting the blockchain; for receiving, validating, and propagating transactions of client modules; and for responding to client module requests for stored information. These modules are connected to other modules of the same type in order to perform these tasks while maintaining consensus on the content of their replica of the blockchain. Prior to the initialization of this module, a list of addresses of neighbor blockchain modules of the same type is provided. During module initialization, it requests the latest blockchain version from its neighbors, and then updates its local copy with any pending approved blocks. Upon initialization completion, a local relational database containing indexes is constructed to facilitate the search for stored blockchain content. This database is always rebuilt at the module initialization, so that there is the guarantee that it has not been tampered with. Once fully initialized, each blockchain module responds to requests for information from a client module based on its local copy of the blockchain, taking into account only consensus-validated information.

Each blockchain module has to verify three conditions for each transaction received:

- whether the transaction format corresponds to the identified type;
- if the transaction has no recipient, whether the transaction signature is correct;
- whether there is no transaction duplication.

If any of these checks fail, the transaction is promptly discarded. Once the conditions are cleared, the transaction is sent to the neighbor blockchain modules through the consensus mechanism. Each time a round of the consensus algorithm is completed, the new block is appended to the local blockchain and the information pertaining to the transactions contained in the closed block is inserted into the local relational database. The blockchain modules are executed by their federation authorized members at the NFV data centers at its data centers, in a number of instances sufficient to meet the requests of client modules.

Blockchain modules feature a blockchain management layer that expose services enabling client modules to send transactions and to query information, as well as other blockchain modules to perform consensus. This layer houses the key pair management functional block and blockchain query service common components in a blockchain module, and contain five new components besides the common ones:

- **Blockchain**: This component maintains blockchain data in permanent storage devices.

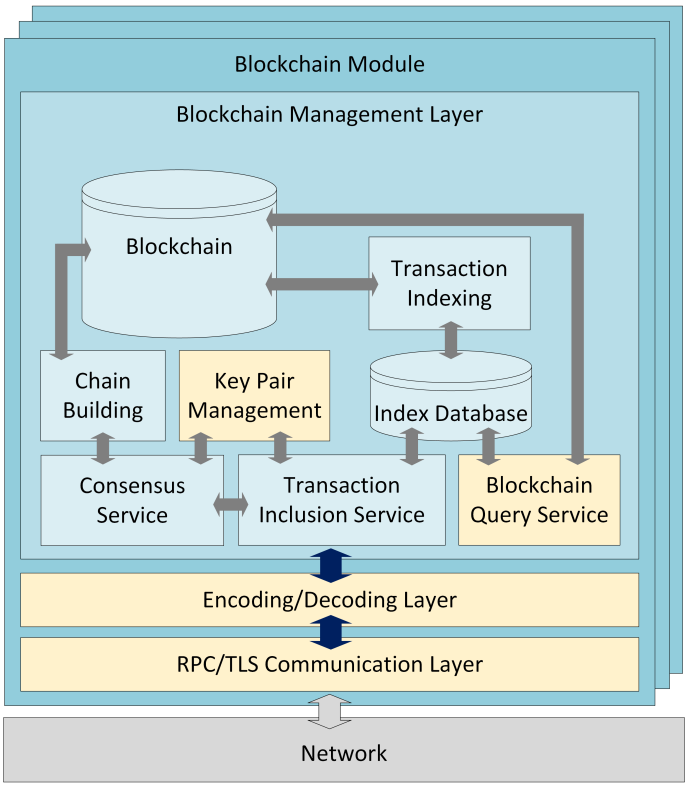


Figura 3.1: Blockchain module architecture. The blockchain module hosts a consistent replica of the blockchain and is responsible for reaching consensus with other blockchain modules, as well as answering client requests.

- **Chain building**: This functional block is the only component that is able to write to the blockchain, and writes only blocks agreed upon by the consensus service.

- **Transaction indexing**: This structure is responsible for building and updating the indexes and transactional database that is used at run time to facilitate access to blockchain information. It updates indexes every time it detects blockchain.

- **Consensus service**: This service exposes the necessary functionalities to interact with other blockchain modules and perform all consensus related operations. Upon reaching a consensus, this service decides if a block is to be sent to the chain building service, or discarded.

- **Transaction inclusion service**: This service exposes the functionality that enables client modules to propose new transactions addition to the current block. Each proposed transaction goes through policy and format validation. An approved transaction is then forwarded to the consensus service if the current blockchain module is the current consensus leader. If not, it is forwarded to the transaction inclusion service of the consensus leader.

CRB and SMB modules are differentiated by two key aspects: their main interested party, and the distribution of the modules in relation to location and ownership. The CRB is conceived to tend primarily to tenant interests in security and auditability, hence the main interested party in the security and correctness of stored information is the tenant. In addition, the main adversary of tenants are other tenants, that may as well compete with each other. Therefore, to establish a distributed consensual blockchain with reduced risk of collusion, a federation must be built and operated by a collective group of tenants. This federation will authorize tenants to own and manage a single CRB module instance, which tenants will allocate inside, or near, a convenient NFV data center, as depicted in Figure 3.2. This federation will manage an authorized CRB module public key list, but will not hold any CRB module private key. Federation policies may manage CRB module placement.

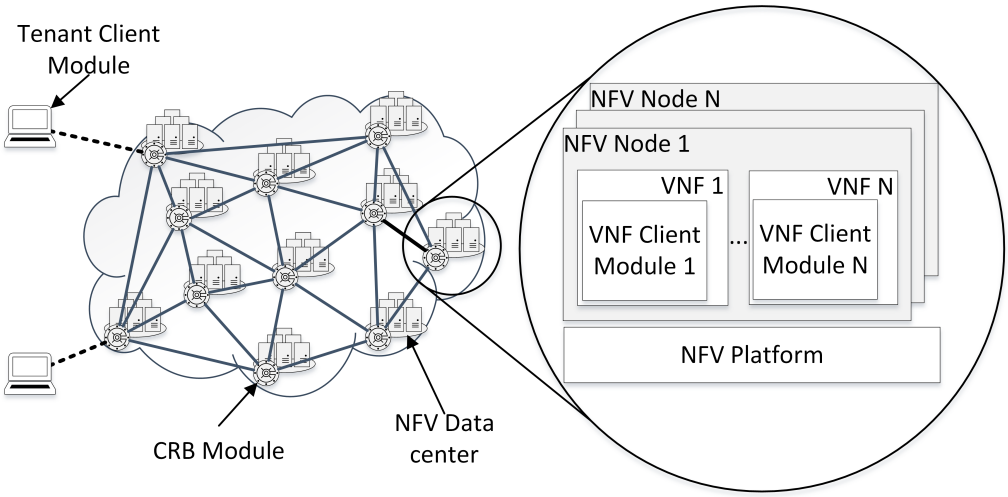


Figura 3.2: CRB modules are located in NFV data centers and are interconnected in a way that allows connection from any other CRB module. Both tenant and VNF client modules are able to connect to one or more CRB modules. No VNF created in the proposed architecture accepts external connections and its configuration state is manage solely by a VNF client module that requests configuration stored in the CRB by an allowed tenant.

SMB modules are conceived to guarantee auditability of inter-service communication inside a single management domain of a NFV data center. This domain could be composed of one or more NFV data centers under a federated administration. This auditability is important to identify the causes of security breaches, or solve disputes with tenants and other federation members. In this case, the main interested party in the correctness and availability of stored information is the owner of the NFV data center. The management services of a NFV data center are autonomous programs or systems that are executed in the different machines that are members of the NFV platform, those services will only misbehave if an attacker

compromises them, or purposely modified by an authorized administrator. So the main adversaries of consensus are these administrators subordinated to the NFV data center domain and possible attackers that can steal their credentials or obtain access to their managed systems. Hence, for the SMB, this work envisions each NFV management domain as the federation responsible for issuing certificates to operate, as well as placement rules, for SMB modules associated to their infrastructure. This federation will manage an authorized SMB module public key list, but will not hold any SMB module private key. These SMB modules would be under different administrative credentials and installed in independent physical machines. Therefore, there is a SMB for each management domain of a NFV data center. SMB modules would be deployed in machines inside this management domain, as depicted in Figure 3.3, and in quantity and locations sufficient to answer to the internal NFV service management requests.

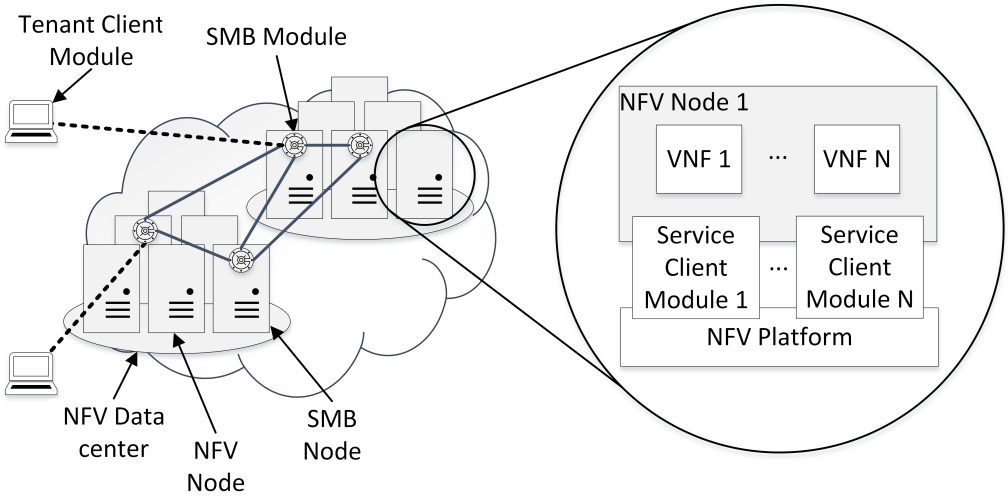


Figura 3.3: SMB modules are located in specialized NFV nodes inside NFV data centers that share the same management domain. These SMB modules are interconnected in a way that allows connection from any other SMB module. Both tenant and service client modules are able to connect to one or more SMB modules.

The use of the CRB and SMB parts of the proposed architecture is designed to be conjoined, however their use can be independent. In fact, the CRB part of the architecture does not depend on the NFV provider for endorsement or provisioning, nor requires any modification to the NFV platform used by VNFs.

3.3.2 Client modules

Client modules enable the retrieval of information from blockchain modules, and the issuing of transactions that add information to a blockchain. There are three client modules in the proposed architecture, from which two of them, the VNF client module and the service client module, are autonomous and interact with a

specific blockchain; and one of them, the tenant client module, is tenant operated and interacts with all blockchains. All clients share a transaction management layer that houses the key pair management functional block and blockchain query service common components. This layer also houses the new transaction management service that exposes to other layers the functionality to validate queried transactions in relation to format and policies, as well as the functionality to construct transactions to be sent to blockchain modules. The blockchain query service is able to access the transaction validation functionality internally.

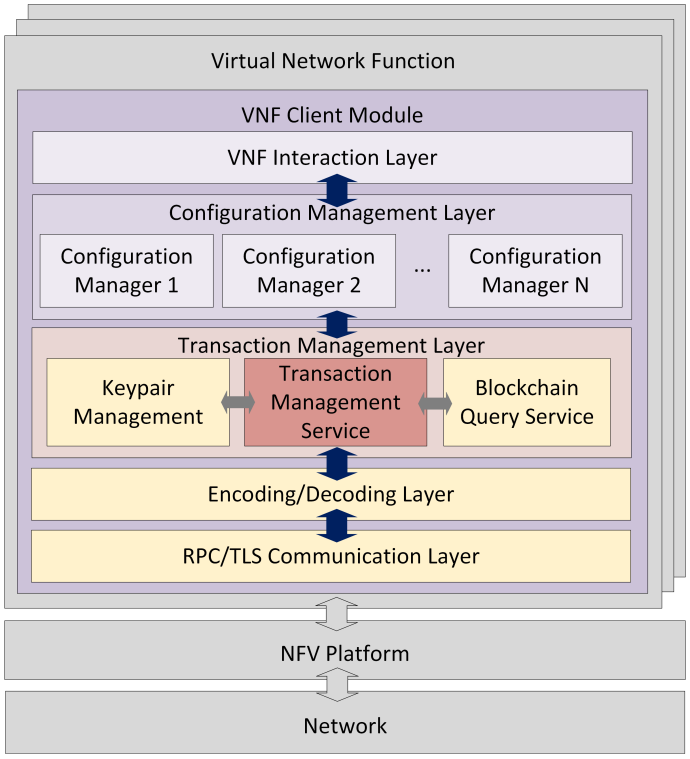


Figura 3.4: VNF client module architecture. The VNF client module connects to a CRB module to send and retrieve transactions regarding the configuration state management of the VNF in which it is installed.

The VNF client module, shown in Figure 3.4, is responsible for sending transactions originating from a VNF to a CRB module; carrying out periodic VNF configuration requests for the VNF that hosts the module; applying the received configuration to the VNF; and managing local VNF key pairs and logical VNF group key pairs. This module connects to one or more CRB modules, depending on their configuration. A VNF client module has the following specific components in addition to the common ones for its type:

- **Configuration management layer**: This layer contains different configuration managers. Each of these managers targets a different software or service installed in the current VNF with functions to retrieve and send configuration

information to its target, and to the blockchain via the transaction management layer.

- **VNF interaction layer**: This structure is responsible for enabling communication and performing the appropriate data format conversions, between the configuration management layer and current VNF software configuration interfaces.

Prior to the initialization of the VNF client module, some variables must be configured through the cloud orchestration platform, such as tenant public keys, addresses of CRB modules and the configuration request interval. During the initialization of this module, the VNF client module generates a pair of asymmetric keys for the current VNF in which the module is installed. The generated public key pair is sent to the client via an encrypted configuration transaction. Afterwards, the VNF client module will issue a request to a registered CRB module for the latest configuration assigned to the local VNF public key by a tenant whose public key the VNF recognizes. This configuration request is performed periodically at the interval specified in the module configuration, and is answered by a transaction set, even if it is an empty one. Upon the retrieval of a set of transactions, the VNF client module will perform local validation of each one. If a configuration transaction is retrieved and passes validation, the configuration transaction is applied, and the transaction timestamp and header are recorded. If a configuration request transaction is retrieved and passes validation, the VNF client module sends the current configuration state to a CRB module through a configuration transaction. Transaction validation consists of:

- checking whether the received transaction is addressed to the current VNF's client module instance public key or a VNF logical group public key for a group the current VNF is a member of;
- verifying that the sender of the transaction corresponds to a tenant public key registered in the local VNF client module;
- verifying that transaction signature is correct and belongs to transaction sender;
- verifying that the transaction header is different from an already processed transaction;
- verifying that the transaction timestamp is newer than the last applied transaction.

The use of the VNF client module requires modifications to the VNF software, so that it is able to manage the local VNF components configuration state. These modifications could be performed by the tenant, or ideally, are implemented by the VNF vendor. A VNF client module runs on each VNF that adopts the proposed configuration management architecture.

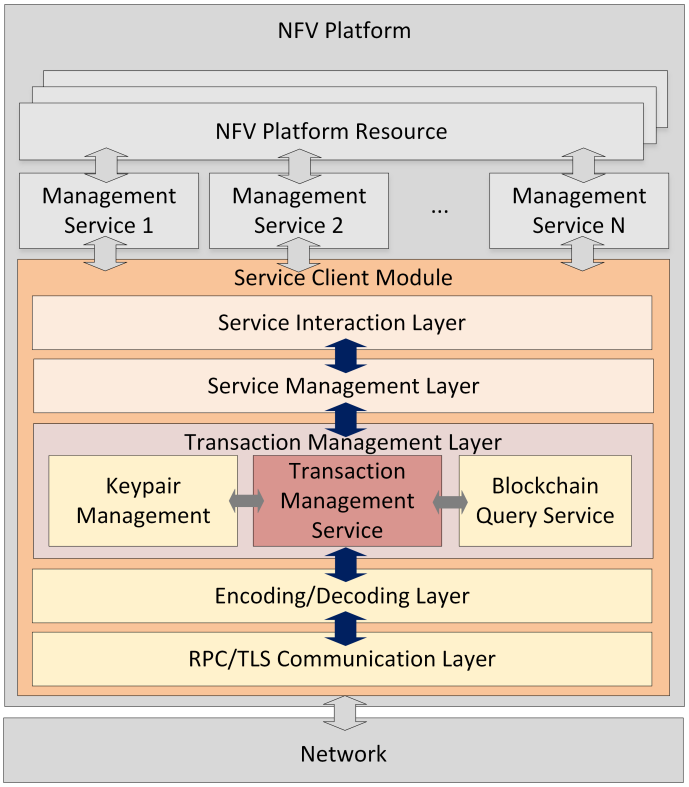


Figura 3.5: Service client module architecture. The service client module connects to a SMB module to send and retrieve transactions regarding the configuration requests to the associated VNF management service.

The service client module, shown in Figure 3.5, is responsible for carrying out periodic queries for pending service transactions for the NFV data center management service associated with the module, executing the requests contained in the retrieved service transactions and sending response transactions originating from a NFV management service to a SMB module. This module connects to one or more SMB modules, depending on their configuration. A service client module has the following specific components in addition to the common ones for its type:

- **Service management layer**: This layer decomposes valid service transactions in appropriate requests for the data center management service associated with the current service client module instance. Then retrieves the result and error information from request execution, in order to enable the construction of an appropriate response transaction. This structure is also responsible for

keeping track of already processed transactions to avoid duplicate request execution

- **Service interaction layer**: This layer enables communication, as well as performs appropriate data format conversions, between the management services of a NFV data center and the service management layer.

The service module requires the configuration of its key pair, issued by the SMB managing federation, addresses of SMB modules and the query interval. It is worth to notice that the SMB managing federation has all authorized services key pairs, necessary for auditability purposes, and must maintain a public list of authorized service module public keys, so that the services are recognizable and addressable by the tenants. During operation, the service client module will issue a request to a registered SMB module for the latest unanswered service transactions addressed to its public key. This query is performed periodically at the interval specified in the module configuration, and is answered by a transaction set, even if it is an empty one. Upon the retrieval of a set of service transactions, the service client module will perform local validation of each one. If a service transaction is retrieved and passes validation, the service client module performs the contained request. Then it creates a response transaction to be sent to a SMB module containing the result and associated errors for the performed request. Service transactions containing performed requests have their timestamp and header are recorded to avoid request duplication. Transaction validation consists of:

- checking whether the received transaction is addressed to the current service client module public key;
- verifying that transaction signature is correct and belongs to transaction sender;
- verifying that the transaction header is different from an already processed transaction.

The use of the service client module requires no modifications to the services of the NFV data center platform, but requires modifications to the service module interaction layer, so that it is able to execute requests using the associated management service, as well as retrieve request results and associated execution errors. These modifications are intended to be performed by NFV data center provider administrators, or implemented by the NFV platform vendors. A service client module is run for each management service instance in an NFV data center that adopts the proposed management service logging architecture.

The tenant client module, illustrated in Figure 3.6, is responsible for enabling the tenants to manage their VNFs and SFCs by sending transactions to an adequate blockchain module, for querying information from blockchain modules and for managing tenant key pairs. This module can be run in any station, concomitantly or not with a blockchain module. If the tenant does not have a local blockchain module, the tenant client module must be configured with one or more network addresses of blockchain modules. A tenant client module has a tenant interaction layer in addition to the common ones for its type that contain the following specific components:

- **Public key database**: This component stores known public keys and their owners identity in order to aid configuration management and transaction addressing.
- **Configuration database**: This component stores VNF configurations retrieved or created by the tenant.
- **VNF management service**: This service exposes functionality that allows the tenant to manage, create or edit VNF configurations, as well as manage their VNFs and SFCs through the NFV datacenter services. The transaction management layer is used to query, retrieve and create the necessary transactions to perform user requests in a way that is transparent to the user. The exposed service functionalities are reachable to the tenant via RPC, hence, it facilitates the client do devise an interface that suits his needs, such as a graphical or command line based one.

The tenant client module is used by VNF owners, which are NFV data center tenants instantiating SFC services based on service software provided by VNF vendors, and by VNF vendors and other parties that wish to offer predefined configuration sets.

It was an architectural option to not store a copy of a blockchain in each VNF client module and service client module, as this would result in high disk space requirements and setup delays for VNFs and management services. This decision does not imply a significant decrease in security, since messages transmitted between client modules and blockchain modules contain signed, and usually encrypted, transactions. A possible attack resulting from this choice is a denial of service performed by a malicious blockchain module. In this attack, a malicious module would stop answering queries from specific client modules, or even all of them. The configuration of multiple blockchain module addresses in the client modules is recommended as a way to mitigate this type of attack.

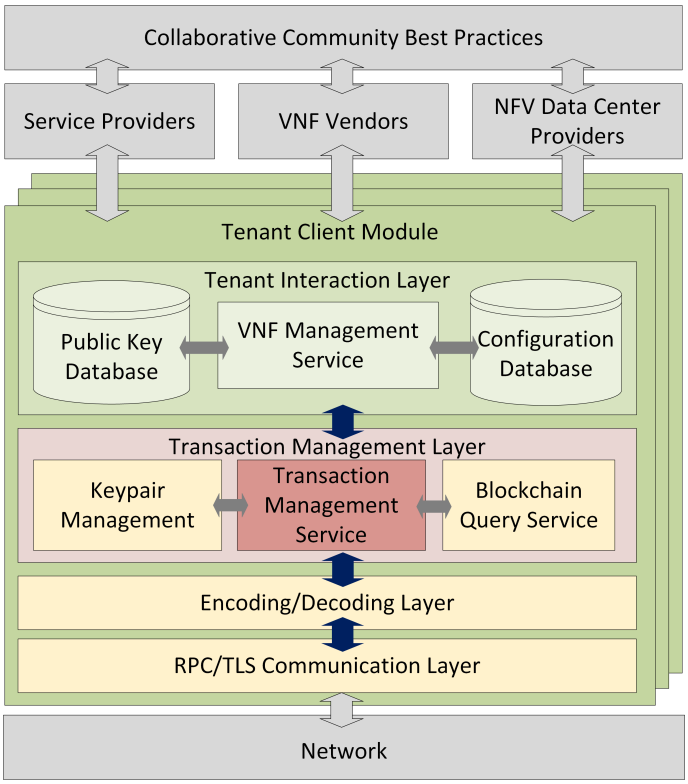


Figura 3.6: Tenant client module architecture. The tenant client module is the interface for configuration authors. It allows the tenant who owns the VNFs to assign them confidential configuration information, and interested parties to publish public configuration templates.

3.4 Key management

The proposed architecture adopts a key management scheme that decouples entity identity from their associated key pairs, anonymizing tenant identity and VNF identity [47]. This mitigates targeted attacks based on blockchain public information. However, transaction execution transparency is preserved [48], and transaction history auditing is possible for an authorized observer with shared tenant private information. In some cases, a tenant may be interested in disclosing their public key. Accounted cases stand for predefined VNF configuration, published by VNF manufacturers, or VNF best configuration practices, shared by the tenant community and service providers. These predefined configurations may aid the initial configuration of tenant VNFs. The architecture uses four logical groups of asymmetric key pairs:

- A **VNF key pair** aims at uniquely identifying a VNF as the sender or recipient of transactions and at signing transactions emitted by a VNF client module. The VNF client module generates this key pair during its initialization, and then sends an encrypted configuration transaction addressed to the tenant public key containing its key pair. A tenant may change this key pair via a configuration transaction.

- A **logical VNF group key pair** enables simultaneous addressing of VNFs by a single transaction. A VNF can belong to multiple groups, so it can store multiple VNF group key pairs. Tenants set these key pairs as part of an encrypted transaction, to avoid any chance of private key leaking as virtual machine configuration data. Although the functionality performed by VNF logical group key pairs could be implemented using a symmetric group key, we choose to use asymmetric key pairs to maintain the uniformity of the transaction recipient field as a public key. This decision conserves the anonymity of VNFs, as an uninformed observer is unable to distinguish between a transaction intended for a VNF group or a specific VNF.

- A **tenant key pair** identify the tenant as senders of transactions, and enable transaction signing. It is important to remember that a VNF client module will only accept transactions from a known sender.

- A **blockchain module key pair** enables the identification of SMB and CRB module instances, as well as the signing of a closed block and the consensus protocol messages. A federation-appointed party certifies this key pair and is ties it to the blockchain node identity. It is important to note that this type o key pair is not used in transactions. Although the proposed architecture assumes software-generated key pairs for simplicity, any module key pair may be stored in smart cards, trusted platform modules or other tamper-proof hardware devices that are compatible with the targeted data center platform.

As in most blockchain implementations, there is no default public key infrastructure (PKI) for key pairs used in transactions [22]. Client modules generate keys on the fly and can replace them promptly in VNFs through an encrypted VNF configuration transaction. In addition, there is no key pair revocation mechanism. This scheme mitigates attacks commonly related to PKIs because trust is explicit. Once used, the public keys will remain as transaction identifiers in the blockchain. In this way, it is not possible for a blockchain observer to track transactions between tenants and VNFs, unless the observer knows tenant and VNFs related key pair information in advance.

3.5 Proposed blockchain structure and transaction schemes

The proposed blockchains are a linked lists of blocks that store transactions. Each block is identified by the signed hash of its contents, which includes the preceding block signed hash. Hence, each block is uniquely identified in such a way that any

observer is promptly able to verify block integrity and sequence integrity. Furthermore, the inclusion of a block in the chain turns the previous block immutable and irreplaceable. A block structure is as follows:

- **Header field**: The header contains block identifying information and the proof of consensus validation.
  - **Identifier**: This field contains the block signed hash of the content field data. This signature is to be verified against the public key in the signer field of current block content.
  - **Proof**: This field contains a list of tuples containing each consensus participant public key, current block content signed by said consensus participant, and his decision regarding current block acceptance.
- **Content**: This field contains all data pertaining to the current block.
  - **Timestamp**: This field contains the date and time of block signing.
  - **Signer**: This field contain the public key of the blockchain module who proposed the block.
  - **Previous**: This field contain the previous signed block content hash.
  - **Transaction list**: This field contains a list of transactions to be stored.

A transaction represents an atomic piece of information to be stored in a blockchain. The proposed architecture defines two transaction classes for each blockchain. The configuration-repository blockchain (CRB) has the configuration transaction class and the configuration request transaction class. Tenant or VNF client modules issue configuration transactions to assign configurations to VNFs, or to define a configuration reference. Only tenant client modules can issue configuration request transactions to request the configuration state of a particular VNF. The service-management blockchain SMB has the service transactions class and response transactions class. Tenant client modules issue a service transaction to request an action from a NFV data center management service. Then the management service responds each service transaction with exactly one response transaction. Every transaction consists of a header and a set of content fields. The header of a transaction is its signed transaction hash. Signing is performed with the transaction sender's private key to ensure the authenticity of the content field set, and the hash function ensures transaction data integrity. All transactions carry common data fields:

- **Type**: This field identifies the type of transaction issued.
- **Timestamp**: This field contains the date and time of transaction signing.

- **Sender**: This field identifies the transaction sender by its public key.
- **Recipient**: This field identifies the transaction recipient by its public key.

Configuration-repository blockchain (CRB) transactions are devised to store and retrieve configuration information from the blockchain. As such, they contain the following additional fields:

- **CID (Configuration IDentifier)**: This field defines a unique identifier for a configuration.
- **VID (Version IDentifier)**: This field defines a unique configuration for a specific CID.
- **Description**: This field may contain a descriptive text for the associated configuration.
- **Configuration**: This field figures solely in configuration transactions, and contains configuration data.
- **Next Recipient**: This field figures solely in configuration request transactions, and identifies the recipient of the corresponding configuration transaction by its public key.

A VNF client module must answer each configuration request transaction addressed to it with a configuration transaction addressed to the public key in its next recipient field. In a configuration request transaction, the sender fills the CID, VID and description fields with the values to be used in the correspondent configuration transaction. If a configuration transaction has a recipient, the sender must encrypt all CRB specific fields and the sender field with the recipient public key. In this way, confidential information that could aid attackers is protected, without loss of auditability for the tenant, who holds the necessary key pairs to trace his own transaction. In order to define a public default configuration, the sender must create it with an empty recipient field and unencrypted transaction data.

The service-management blockchain (SMB) acts as an intermediary between a tenant and a NFV data center management service, as such, both transactions are straightforward. The service transaction contains two additional fields:

- **Request**: This field contains the desired command, parameters and authorized credential of a VNF or SFC management action. The request field format is dependent on the NFV data center platform used, and must match what is expected by the service client modules.

- **Response Key**: This field contains a symmetric key to be used in the response transaction.

The response transaction has three additional fields:

- **Service Transaction**: This field identifies the correspondent service transaction by its header;
- **Result**: This field contains the result from a correspondent request performed by the appropriate NFV data center management service;
- **Error**: This field contains any error channel output originated by the NFV data center management service process.

A service transaction has all its specific fields and the sender field encrypted with the recipient public key. A response transaction has all its specific fields and the sender field encrypted with the symmetric key contained in the associated request transaction. In this way, the content of the service management operation is protected, avoiding tenant exposure, but with no loss of the intended auditability functionality for the SMB managing federation, as it will be able to read all transaction information exchanged between a tenant and a service. It is worth to notice that an attacker cannot change the service client key pair without turning the service inactive, as the tenant clients have no other way to access the services. It is the duty of the SMB managing federation to maintain an accurate data center management services public key list and to ensure the provenience of its service list, which could be easily accomplished with a signature.

The validation of a transaction by a CRB or SMB module must observe if all the rules and field semantics described above are obeyed. An invalid transaction is discarded immediately. The validation of the transactions is performed locally at each module and is essential for the correct operation of the consensus algorithm. A consensus participant votes for the acceptance of a transaction if and only if the transaction is valid locally and does not conflict with any other transaction already accepted by that consensus participant. Such a requirement is important for the employed PBFT consensus algorithm [15, 43].

3.6 Secure migration of virtualized network functions

Migration of a virtualized network function (VNF) consists in copying its configuration and states to another machine on the network. The use of conventional

migration mechanisms, commonly used to migrate virtual machines, implies exposing the system to its security vulnerabilities [16]. Even though the communication from the origin to the destination may be encrypted, data is not encrypted from VNF origin to VNF destination, because they need to be understood by the hypervisor [12]. However, a VNF migration procedure can be treated differently from the migration of a traditional virtual machine due to its simpler scope of use. VNFs are special purpose non-persistent virtual machine instances, created on-demand and based on a predefined read-only service image. As such, VNFs do not retrieve past state information from a stored volume and are permanently destroyed when shut down. VNFs do not receive connections either and, therefore, there is no client session information associated with a VNF or links to be reestablished. What differentiates one VNF from another VNF based on the same image is solely its configuration state. Therefore, a tenant can migrate a VNF, without loss of functionality, by creating a new VNF instance on the intended location, transferring its latest configuration state. Finally, requesting the service function chaining (SFC) platform configuration to switch from the current VNF instance to the new VNF instance on its service path.

This work proposes carrying out the migration of the service performed by a VNF through transactions on the proposed blockchain architecture. In fact, the owner of the NFV, a tenant, migrates a VNF from one location to another by executing a service transaction. Hence, a VNF migration is sequence transactions in which the configuration of the origin VNF is written in the CRB, and then accomplished by reading this configuration at the destination VNF. VNF migration is an operation performed on both configuration-repository blockchain (CRB) and service-management blockchain (SMB). The steps to perform a VNF migration are:

1. The tenant client module sends a service transaction to the service-management blockchain (SMB) requesting a new VNF at the desired destination.

2. The tenant watches the SMB waiting for a response transaction that signals the new VNF is ready, and then watches the CRB for a configuration transaction that contains the VNF key pair.

3. The tenant client module sends a configuration request transaction to the CRB addressed to the source VNF public key, with the destination VNF public key in the next recipient field.

4. The destination VNF checks the CRB periodically and detects the new configuration, then it retrieves it and load the new configuration state.

5. The tenant client module sends a service transaction to the SMB switching his SFC path to utilize the destination VNF.

6. The tenant client module sends a service transaction to the SMB terminating the source VNF.

As it does not depend on the data center NFV platform used, this migration mechanism is compatible with VNFs running on separate cloud platforms, even if the virtualization architecture is different. In addition, the process could be fully automated.

Chapter 4

Performance Evaluation of the Blockchain Module Prototype

A prototype of the proposed architecture was implemented as a proof of concept, and a suitable blockchain were developed and implemented in the Python 2.7 language, using pyCryptodome library for cryptography related operations. The main objective of the prototype is to evaluate the trade-offs of the proposed architecture in a controlled environment with the essential functionality set. Production implementations of this architecture can be constructed, without loss of functionality, in Ethereum [28] or Hyperledger[1] based blockchains. However the specific consensus constraints, blockchain structure and transaction schemes must be ported to the desired environment. We implement a specific prototype to obtain a complete control of all variables and parameters and, consequently, facilitating performance measurements.

4.1 Prototype environment and setup

The main evaluation targets are the transaction writing and retrieving rates, and the request time overhead introduced by the utilization of a blockchain module (CRB module or SMB module). We will evaluate the transaction times for the CRB, because the SMB transaction size is small and it can be compared to a CRB transaction with a data size between 10 B and 1 kB. We evaluate the request time overhead for the SMB because we estimate that the impact of a delay would be more critical, as management operations tend to require more than one service transaction. In most experiments, the number of blockchain modules was varied from one up to ten, which would correspond to ten NFV data centers for the CRB. We decided to stop our evaluation at ten blockchain modules because it represents

[1] https://www.hyperledger.org/

enough consensus participants for highly utilized systems and the results point to predictable behavior should more consensus participants be added.

Our prototype uses Rivest–Shamir–Adleman (RSA) public key cryptography system with a 2048-bit key length parameter. For the signatures, our prototype uses the Public Key Cryptography Standard #1 Probabilistic Signature Scheme (PKCS#1-PSS), which produces 256-bit signatures based on a RSA key pair. The prototype of the proposed architecture runs and is evaluated in the Open Platform for NFV (OPNFV)[2]. Although OPNFV is used for the prototype evaluation, it is important to note that the proposed architecture, as well as the prototype, is platform agnostic. The blockchain modules were evaluated in Intel Core i7-4770 3.4 GHz based computers with 32 GB RAM, the VNF client modules were evaluated in OPNFV VNFs, executing a dummy forwarding service, and the tenant client modules are executed in Core i5-2410M 2.30GHz notebooks for the writing experiments, and at the same machine of the blockchain modules for the reading experiments, to account for high speed data center networks.

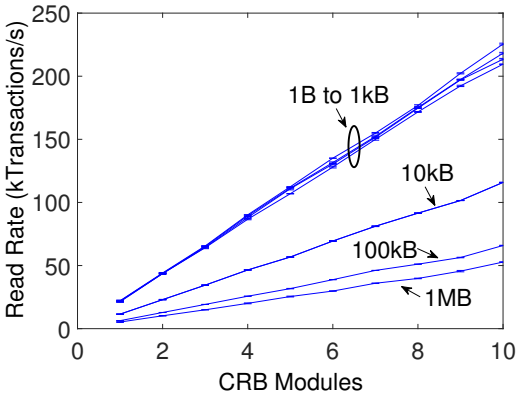
4.2 Evaluation of conducted experiments

Two experiments are conducted in order to evaluate the variables that may impact: i) the transaction processing behavior, ii) the consensus delay, and iii) the total message data exchanged between blockchain modules.

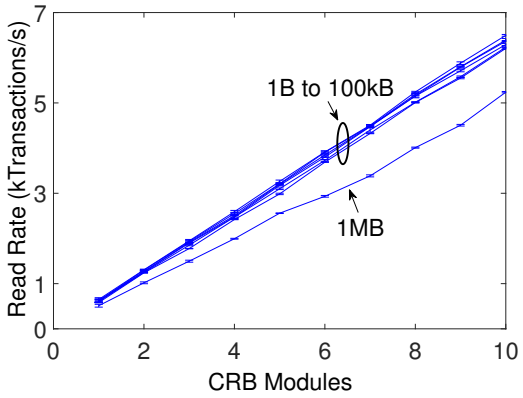
The first experiment aims at measuring the maximum sustained transaction request rate from one to ten configuration-repository blockchain (CRB) modules. The transaction request rate is a close estimate of the number of client module (VNF client module and tenant client module) requests that can be answered simultaneously. As mentioned before, the number of blockchain modules is varied from one up to ten, which we deem sufficient because it corresponds to up to ten NFV data centers for the CRB. During the measure of request rate, configuration transactions were requested as fast as possible by ten VNF client modules, which are sufficient to reach the maximum transaction read rate for ten CRB modules. The write rate measures were performed by constantly issuing pre-computed configuration transaction from the tenant module, so that transaction generation time does not affect the results.

In Figure 4.1a we fixed configuration size in 1 B, and varied block data size between 1 B and 1 MB to evaluate the impact the transaction seek time in function of the number of transactions inside a block. It is important to note the bigger is block data size, the bigger the number of transactions in a block. Transactions were requested randomly from block contents using a uniform distribution because the first transactions in a block are accessed significantly faster than the last transaction

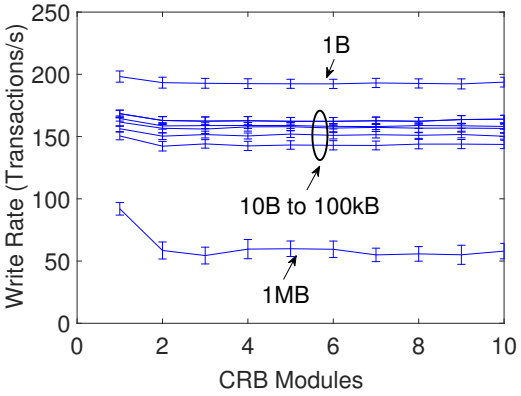
[2]https://www.opnfv.org/



(a) Read requests/s for a fixed configuration size of 1 B, and block data size between 1 B and 1 MB.



(b) Read requests/s for a fixed block data size of 1 MB, and a configuration size between 1 B and 1 MB.



(c) Write requests/s for a configuration size between 1 B and 1 MB.

Figura 4.1: Prototype maximum transaction processing rates

on an uncached block. The results demonstrate that block sizes up to 1kB do not pose significant decreases in performance. That is explained by the utilization of a disk sector size of 4kB, which allows blocks up to this size to be retrieved in a single read operation. Greater block sizes affect request rates more severely as continuous memory placement is necessary. We consider these results satisfactory as the read rates scale linearly with the number of consensus participants, and the prototype is able to handle 50 thousand requests per second, even with large configuration data sizes.

In Figure 4.1b we fixed block data size in in 1 MB, and varied configuration size between 1 B and 1 MB to evaluate the impact of transaction size in its retrieval time from storage o delivery. In this case, the retrieved transaction was always the first one inside a block, so that transaction seek times are constant. Results shows that block size dominates read request time up to 100 kB transactions. On the other hand, for large-size transactions, network bandwidth limits the rate because

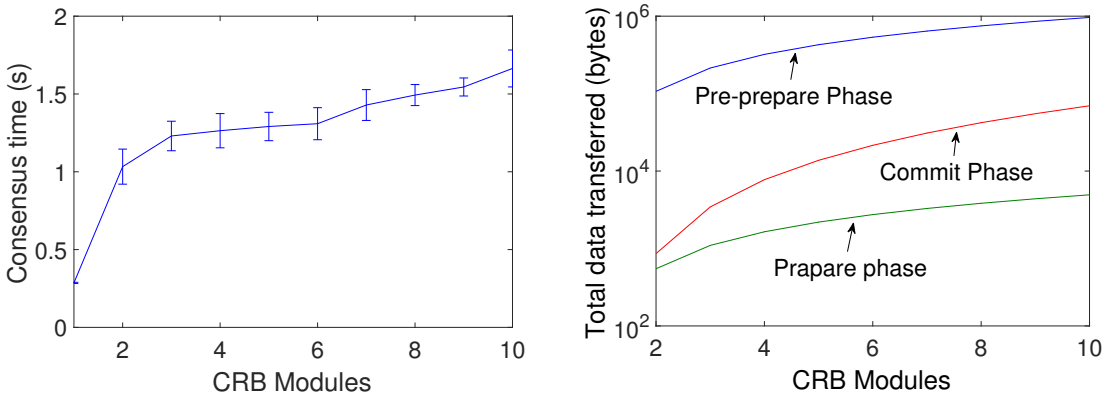
we have to send a lot of packets that cause link (or switch) congestion. Furthermore, our prototype scales linearly with the number of blockchain modules, as illustrated in Figures 4.1a and 4.1b. We consider these results satisfactory as the read rates scale linearly with the number of consensus participants, and the prototype is able to handle 5 thousand requests per second, even with large configuration data sizes.

In Figure 4.1c we varied configuration sizes between 1 B and 1 MB to evaluate the impact of the transaction size in the sustained CRB block writing rate. Results show that the maximum request write rate is stable and does not increase with blockchain module addition. This is due to the adopted consensus algorithm, which centralizes block creation and thus imposes the consensus leader's writing capacity as a bottleneck for the system. A second bottleneck lies in consensus participants transaction validation rate, that is processor intensive operation. Nevertheless, unprocessed transactions remain in the transaction log to be included in the next consensus round, implying the architecture is tolerant to processing rate peaks up to the buffering capacity of blockchain consensus participant. Transaction rates for 1B transactions provide an upper bound for write request rates and the disparity between the write and read request rates is a consequence of cryptographic transaction verification. Transaction sizes from 10B to 100kB offer a small linear performance degradation for geometric size increases. For larger transactions of 1MB, write request rate is affected by process memory constraints. We argue that configuration updates to VNF configuration are not very frequent, and that a stable transaction write rate of more than 150 transactions per second written to the blockchain is enough to sustain optimal CRB operation. As a reference, Bitcoin sustains its network with a rate of at most[3] seven transaction written per second, and Ethereum network has a historical peak[4] of 15 written transaction per second.

The second experiment consists in verifying consensus delay and data exchanged in relation to the number of consensus participants, Figure 4.2. To this end, a batch of 100 configuration transactions, 400 B in size each, was sent to the consensus leader for each experiment setup. Results show that consensus delay slowly increases with the number of nodes, but remains under 2 seconds for ten participants, Figure 4.2a. Hence, we succeed to obtain a short consensus time as desired and we consider this an excellent result. As a reference, even though the comparison is unfair due to the difference in consensus algorithm type, Bitcoin consensus takes around ten minutes, in which only a single 1 MB block with up to 4200 transactions is validated. In addition, our analysis of total data exchanged during consensus, Figure 4.2b, shows that the pre-prepare and prepare phases of the adopted consensus algorithm scale linearly, while commit phase data exchange requirements scales

[3]Bitcoin transaction rate chart. https://blochchain.info/

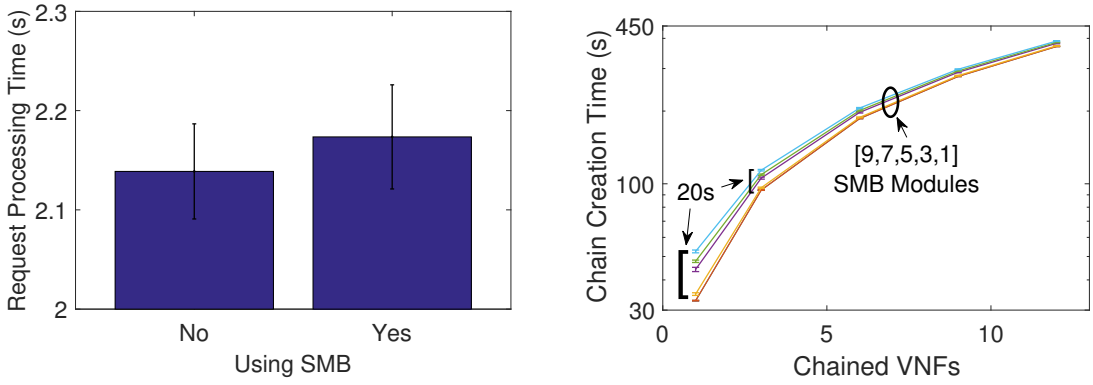
[4]Ethereum transaction chart. https://etherscan.io/



(a) Consensus time (b) Total message data exchanged

Figura 4.2: Prototype consensus evaluation for a 400 B firewall configuration at 100 write transaction requests/s.

quadratically with the number of consensus participants. This results demonstrate that the blockchain-adapted PBFT consensus algorithm behaves as exactly as expected in a PBFT-based consensus in relation to the message exchange rate, and close enough in relation to consensus behavior time [33]. Hence the modifications introduced to PBFT in this proposal do not affect the expected algorithm behavior in the evaluated scenario. However, when the communication overhead nears the configuration data volume, a noticeable data transmission bottleneck would be verified for the consensus leader, who is responsible for configuration data propagation for all other consensus participants.



(a) Data center management service request processing time for one SMB module

(b) SFC creation time as a function of chain length, varying from 1 to 9 SMB modules

Figura 4.3: Prototype evaluation of SMB-induced time overhead.

In the next experiment, we want to evaluate the delay introduced by our proposal in guaranteeing a secure environment. We evaluate the impact of the indirect service management approach introduced by the service-management blockchain (SMB). Figure 4.3a shows the request processing time for the NFV data center

management services when using our proposed architecture, compared with direct native service interface usage. Results show that, considering the 95% confidence interval, the mean delay added to request processing time in NFV data center management requests is of less than 0.02 seconds, and up to 0.13 second at the worst case. Hence, this result demonstrates that the delay introduced by the proposed architecture approach is insignificant. In Figure 4.3b we evaluate the total service function chain creation time to evaluate the impact of the overhead introduced by the proposed architecture in normal VNF data center operation. In the evaluated scenario, we varied the number of SMB modules from one (no consensus) up to nine to ascertain the impact of consensus-introduced delay in the overall chain creation time. Results for the case of a chain with one VNF show a small, but noticeable, delay introduced to chain creation time. This delay varies from one to 20 seconds added to the original 30 seconds. As the number of VNFs in the chain increase, while the consensus-introduced delay remains constant, chain creation time overtakes the delay introduced by consensus by more than 20 times. This experiment demonstrates that the impact of consensus overhead, estimated as the ratio between the chain creation time of the evaluated case and the case with one SMB node, varies from medium to insignificant as the number of VNFs in a chain grows.

Chapter 5

Conclusion

Network function virtualization (NFV) and service function chaining (SFC) are alternative software-based technologies to leverage commercial off-the-shelf hardware to perform and replace hardware-specialized middleboxes roles. While reducing CAPEX and OPEX, the software-based approach enables multiple infrastructure providers to provide highly customized end-to-end communication services. Network function vendors also benefit of reduced time to market (TTM) in order to offer specific virtual network functions, when compared to the long development cycle associated to hardware-based middleboxes. Nevertheless, the resulting programmability of the network core, exposes all traffic that is forwarded by it to an increased number of threats. Therefore, new solutions are necessary to fault, configuration, account, performance, and security (FCAPS) management that reduce the possible virtual network function (VNF) attack vectors to mitigate the new threats.

In this context, this work proposed a two-fold blockchain-based architecture for the secure configuration management of VNFs, as well as to provide transparent and reliable interservice auditability to datacenter VNF management services. Our two blockchains are called configuration repository blockchain (CRB) and service management blockchain (SMB). Furthermore, we demonstrated through experimental results that our proposed blockchain-based repositories and specialized transaction schemes, coupled with our proposed PBFT-based consensus mechanism, are a good solution to implement configuration, administration, and security management in conformance with FCAPS model at multiple federated NFV data centers. The proposed architecture is resilient to collusion attacks or faults from up to a third of the blockchain modules, and no information is compromised even during a successful collusion attack, multiple blockchain module faults or network partitions.

This work also proposed a platform-agnostic VNF migration scheme that leverages the capabilities of both proposed blockchains, and preserves the confidentiality of VNF information. This migration scheme is able to recreate VNFs even when different NFV platforms or processor architectures are used. This was accomplished

analyzing the main differences of usage scope between a general purpose virtual machine and a virtual network function (VNF), then eliminating redundant steps of conventional migration mechanisms to derive only the necessary core functionality. Thus, we decomposed a VNF migration in a sequence of operations, and translated that sequence to transactions in which the configuration of the origin VNF is transposed to the destination VNF using configuration repository blockchain (CRB) transactions. Meanwhile, the orchestration of the involved NFV data center platforms is performed through transactions in one or more service management blockchains (SMBs).

A prototype of the proposed architecture was developed to with the main objective of evaluating the trade-offs of the proposed architecture in a controlled environment with the essential functionality set. We carefully implemented a specific prototype to obtain complete control of all variables and parameters involved and, consequently, facilitating the performance measurements and analysis. In most experiments, the number of blockchain modules was varied from one up to ten, which would correspond to ten NFV data centers for the CRB, or a high utilized datacenter federation for the SMB, with as much management requests per seconds expected as VNF configuration changes per second in ten data centers. The experimental behavior with more than ten consensus participants seems highly predictable, and should maintain the same behavior up to network link capacity exhaustion. We decided to limit our evaluation at ten blockchain modules because it represents enough consensus participants for highly utilized systems and the results point to predictable behavior should more consensus participants be added.

Transaction writing and retrieving rates were evaluated for the CRB module, while the request time overhead introduced by the utilization of indirect blockchain-assisted operations was evaluated for the SMB module. The results were very satisfactory, and further analysis has shown that experimental results from one blockchain are indeed representative of the approximate behavior of the other. As the SMB transaction size is small, SMB module sustained transaction rates can be estimated from CRB experimental results. Using total transaction size as an estimate, SMB transactions rates should be close to CRB transactions rates for transactions between 10 B and 1 kB. Meanwhile, all CRB operations require a single transaction and, therefore, even if the connection method used as a baseline comparison for a native direct configuration time may vary, the expected configuration time overhead introduced by the indirect CRB approach should still be within ten seconds.

Experimental results demonstrate that the disk sector size directly impacts in the performance blockchain information retrieval, and careful planning and architectural evaluation is necessary to find an optimal disk sector size that reduces the number of operations necessary to access the contained information. We conjecture

that larger disk sectors would be beneficial to performance, but they must not be so large that their memory placement negatively affects blockchain block caching in memory. Furthermore, the experimental results uncovered important aspects of the relation between block size, transaction size and network bandwidth pertaining stored transaction retrieval rates. This culminates in implementation trade-offs that should be observed when planning an implementation of the proposed architecture. The definition of a maximum block size also sets the maximum transaction size, limiting the maximum configuration potential of a single transaction and, therefore, a block should not be so small that would break configuration transactions in many smaller parts. For a block of fixed size, transaction retrieval performance is defined by two main factors: transaction seek time, proportional to the number of transactions in a block. Hence inversely proportional to transaction size; and the available network bandwidth, that degrades after the transaction size surpasses a certain threshold, because if seek time is null, the blockchain module transaction retrieval rate is approximately constant despite transaction size, as results have shown. On the other hand, transaction-writing performance decreases sharply at a certain block size threshold, because it is bounded by network bandwidth during consensus in a way analogous to transaction-retrieval rates is affected by the transaction size. Hence, we advise the experimental measurement of transaction write rates to determine the ideal maximum block size for a target federation, suggested as the inflection point of a write rate for block size curve. Then adjusting disk sector size to the largest divider of block size that does not require increased instruction cycles to be loaded to memory. Then a minimal transaction size should be set based on the transaction seek time performance and network bandwidth trade-off. It is worth to notice that the network bandwidth upper bound for consensus is based on the bandwidth between blockchain modules, while the network bandwidth for transaction retrieval is based on the bandwidth available to the client of a blockchain. SMB and CRB parameter trade-offs can be estimated by the same process, however requirements on maximum available transaction size are less stringent, because management commands are usually smaller than configuration files, hence transaction size requirements on the overall parameter trade-offs can be relaxed.

Consensus related results were deemed especially good. Experimental results demonstrate that the PBFT-based proposed protocol for the blockchain case behaves exactly as its distributed processing original counterpart in regards to expected performance curves behavior and message exchange data volume. Furthermore, when compared to eventual consistency proof-based protocols such as the ones from Bitcoin or Ethereum, the transaction write rate and consensus times achieved are excellent, with mean sustained write rate more than twenty times and ten times the rate of the former at their best possible rate, respectively. Although the usage pat-

terns are different, as an estimate, the proposed architecture could tent to a tenant community at least ten times greater than the one from Bitcoin. In addition, consensus time remains approximately constant even with the addition of more consensus participants. This should remain true until the consensus-message exchange-data volume surpasses the underlying network bandwidth capacity. Hence, consensus message-validation speed of consensus participants and the block creation speed of the consensus leader are the main performance bottlenecks for consensus. Nevertheless, this bottlenecks do not impact the proposed scenarios, because the achieved performance during stress conditions is still greater than the estimated write rate requirements. Write rate bottlenecks are due to the adopted consensus algorithm, which centralizes block creation and is heavily reliant on cryptography operations for message authentication, in order to resist collusion attacks and byzantine faults. However, the most computationally expensive consensus operations are sequential hashing and signature validation. Both operations are fully parallelizable, hence these operation should scale linearly with the addition of multiprocessing capabilities and processor cores, to enable much shorter consensus times. It is worth to note that even if the leader centralizes block creation, this does not constitute a single point of failure, as the leader can be promptly replaced by any consensus participant as soon as it detects the failure, what happens in a few seconds consensus of inactivity. Moreover, if the consensus leader fails after the pre-prepare PBFT consensus phase, the consensus round finishes with no issues. Nevertheless, this result points to new research directions, such as investigating the impact of specialized hardware to perform cryptography operations on PBFT-based consensus algorithms.

Further experimentation was conducted regarding blockchain indirect approach to management operations and consensus time impact on request time overhead, when compared to NFV data center native interfaces. Results show that the blockchain indirect approach has insignificant sub-second impact on request times, and that the major impact on request times results from consensus time. For single transaction operations, this consensus time overhead is of a few seconds and do not pose significant concerns. For multi-transaction operations,however, such as the establishment of a service function chain and accompanying VNFs, the introduced operation delay may become noticeable. Nevertheless, this introduced delay is overshadowed by the total necessary time for the creation of a service chain, which is proportional to chain size, while the introduce delay is constant for the same transaction number. Our experimental data show that the consensus delay becomes minimal for a service function chain of three VNFs and insignificant for a service function chain of six VNFs or more.

In conclusion, besides eliminating single points of failure and providing high availability to NFV management information, the proposed architecture ensures

the confidentiality and anonymity of all sensitive VNF management information. In addition, even with the enforced confidentiality and anonymity security features, the proposed architecture maintains and further enhances the auditing capabilities of authorized entities. Furthermore, the aggregate benefits of the proposed architecture are obtained without significant performance compromises for the evaluated NFV data center platform.

Finally, we envision a few future research directions that would aid the field of blockchain-enabled NFV security, such as the investigation of the portability of configuration update functionality to other service function chaining (SFC) platform elements, and the impact of different machine hardware and system architectures in blockchain performance. Furthermore, we would like to measure the trade-off of different consensus algorithms on the proposed architecture, the performance of different cryptography algorithms on prototype implementations, the benefits of hardware-assisted cryptography on consensus time and blockchain transaction write rate, and the impact of hardware-based endpoint security technologies on further enhancing the architecture.

Referências Bibliográficas

[1] BHAMARE, D., JAIN, R., SAMAKA, M., et al. "A Survey on Service Function Chaining", *J. Netw. Comput. Appl.*, v. 75, n. C, pp. 138–155, nov. 2016. ISSN: 1084-8045. doi: 10.1016/j.jnca.2016.09.001.

[2] MASSONET, P., DUPONT, S., MICHOT, A., et al. "Enforcement of global security policies in federated cloud networks with virtual network functions". In: *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, pp. 81–84, Oct 2016. doi: 10.1109/NCA.2016.7778597.

[3] JOHN, W., PENTIKOUSIS, K., AGAPIOU, G., et al. "Research Directions in Network Service Chaining". In: *2013 IEEE SDN for Future Networks and Services*, pp. 1–7, Nov 2013. doi: 10.1109/SDN4FNS.2013.6702549.

[4] BOURAS, C., KOLLIA, A., PAPAZOIS, A. "SDN & NFV in 5G: Advancements and challenges". In: *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pp. 107–111, March 2017. doi: 10.1109/ICIN.2017.7899398.

[5] MIJUMBI, R., SERRAT, J., L. GORRICHO, J., et al. "Management and orchestration challenges in network functions virtualization", *IEEE Communications Magazine*, v. 54, n. 1, pp. 98–105, January 2016. ISSN: 0163-6804. doi: 10.1109/MCOM.2016.7378433.

[6] XU, X., PAUTASSO, C., ZHU, L., et al. "The Blockchain as a Software Connector". In: *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pp. 182–191, April 2016. doi: 10.1109/WICSA.2016.21.

[7] RAMAN, L. "OSI systems and network management", *IEEE Communications Magazine*, v. 36, n. 3, pp. 46–53, Mar 1998. ISSN: 0163-6804. doi: 10.1109/35.663327.

[8] ITU-T STUDY GROUP 4. "ITU-T Recommendation M.3400". In: *ITU-T Series M: TMN and Network Maintenance: International Trans-*

mission Systems, Telephone Circuits, Telegraphy, Facsimile and Leased Circuits, International Telecommunication Union, February 2000. https://www.itu.int/rec/T-REC-M.3400-200002-I/en.

[9] NANNRA, A. "Blockchain Enabled Industrial Strength Trust for Modern Business Environments". In: *Cisco Innovation Blog*, April 2017. https://blogs.cisco.com/innovation/blockchain-enabled-industrial-strength-trust-for-modern-business-environments. Last accessed February 3rd 2018.

[10] MATTHEWS, S. "Is Blockchain the Silver Bullet of IoT?" In: *Internet of Things World Forum - IoTWF'2017*, London, United Kingdom, May 2017.

[11] PATTARANANTAKUL, M., HE, R., MEDDAHI, A., et al. "SecMANO: Towards Network Functions Virtualization (NFV) Based Security MANagement and Orchestration". In: *2016 IEEE Trustcom/BigDataSE/ISPA*, pp. 598–605, Aug 2016. doi: 10.1109/TrustCom.2016.0115.

[12] FIROOZJAEI, M. D., JEONG, J. P., KO, H., et al. "Security challenges with network functions virtualization", *Future Generation Computer Systems*, v. 67, pp. 315 – 324, 2017. ISSN: 0167-739X. doi: https://doi.org/10.1016/j.future.2016.07.002.

[13] ONGARO, D., OUSTERHOUT, J. "In Search of an Understandable Consensus Algorithm". In: *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pp. 305–319, Philadelphia, PA, 2014. USENIX Association. ISBN: 978-1-931971-10-2.

[14] MATTOS, D. M. F., DUARTE, O. C. M. B., PUJOLLE, G. "Um Protocolo Simples e Eficiente para Atualização Consistente de Políticas em Redes Definidas por Software com Controle Distribuído". In: *XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2017*, Belém, PA, Brazil, May 2017.

[15] CASTRO, M., LISKOV, B. "Practical Byzantine Fault Tolerance". In: *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pp. 173–186, Berkeley, CA, USA, 1999. USENIX Association. ISBN: 1-880446-39-1.

[16] LAL, S., TALEB, T., DUTTA, A. "NFV: Security Threats and Best Practices", *IEEE Communications Magazine*, v. PP, n. 99, pp. 2–8, 2017. ISSN: 0163-6804. doi: 10.1109/MCOM.2017.1600899.

[17] COUGHLIN, M., KELLER, E., WUSTROW, E. "Trusted Click: Overcoming Security Issues of NFV in the Cloud". In: *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, SDN-NFVSec '17, pp. 31–36, New York, NY, USA, 2017. ACM. ISBN: 978-1-4503-4908-6. doi: 10.1145/3040992.3040994.

[18] MASSONET, P., DUPONT, S., MICHOT, A., et al. "An architecture for securing federated cloud networks with Service Function Chaining". In: *2016 IEEE Symposium on Computers and Communication (ISCC)*, pp. 38–43, June 2016. doi: 10.1109/ISCC.2016.7543711.

[19] PATTARANANTAKUL, M., TSENG, Y., HE, R., et al. "A First Step Towards Security Extension for NFV Orchestrator". In: *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, SDN-NFVSec '17, pp. 25–30, New York, NY, USA, 2017. ACM. ISBN: 978-1-4503-4908-6. doi: 10.1145/3040992.3040995.

[20] REYNAUD, F., AGUESSY, F. X., BETTAN, O., et al. "Attacks against Network Functions Virtualization and Software-Defined Networking: State-of-the-art". In: *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pp. 471–476, June 2016. doi: 10.1109/NETSOFT.2016.7502487.

[21] NAKAMOTO, S. "Bitcoin: A peer-to-peer electronic cash system". 2008. http://bitcoin.org/bitcoin.pdf.

[22] MUKHOPADHYAY, U., SKJELLUM, A., HAMBOLU, O., et al. "A brief survey of Cryptocurrency systems". In: *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pp. 745–752, Dec 2016.

[23] BOZIC, N., PUJOLLE, G., SECCI, S. "A tutorial on blockchain and applications to secure network control-planes". In: *3rd Smart Cloud Networks Systems*, pp. 1–8, Dec 2016. doi: 10.1109/SCNS.2016.7870552.

[24] CHRISTIDIS, K., DEVETSIKIOTIS, M. "Blockchains and Smart Contracts for the Internet of Things", *IEEE Access*, v. 4, pp. 2292–2303, 2016. ISSN: 2169-3536. doi: 10.1109/ACCESS.2016.2566339.

[25] BOUDGUIGA, A., BOUZERNA, N., GRANBOULAN, L., et al. "Towards Better Availability and Accountability for IoT Updates by Means of a Blockchain". In: *2017 IEEE European Symposium on Security and Privacy*

Workshops (EuroS PW), pp. 50–58, April 2017. doi: 10.1109/EuroSPW.2017.50.

[26] AZARIA, A., EKBLAW, A., VIEIRA, T., et al. "MedRec: Using Blockchain for Medical Data Access and Permission Management". In: *2016 2nd International Conference on Open and Big Data (OBD)*, pp. 25–30, Aug 2016. doi: 10.1109/OBD.2016.11.

[27] FRANTZ, C. K., NOWOSTAWSKI, M. "From Institutions to Code: Towards Automated Generation of Smart Contracts". In: *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, pp. 210–215, Sept 2016. doi: 10.1109/FAS-W.2016.53.

[28] WOOD, G. "Ethereum: A secure decentralised generalised transaction ledger". 2014. http://bitcoinaffiliatelist.com/wp-content/uploads/ethereum.pdf.

[29] FUJIMURA, S., WATANABE, H., NAKADAIRA, A., et al. "BRIGHT: A concept for a decentralized rights management system based on blockchain". In: *2015 IEEE 5th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, pp. 345–346, Sept 2015. doi: 10.1109/ICCE-Berlin.2015.7391275.

[30] ZYSKIND, G., NATHAN, O., PENTLAND, A. S. "Decentralizing Privacy: Using Blockchain to Protect Personal Data". In: *Proceedings of the 2015 IEEE Security and Privacy Workshops*, SPW '15, pp. 180–184, Washington, DC, USA, 2015. IEEE Computer Society. ISBN: 978-1-4799-9933-0. doi: 10.1109/SPW.2015.27.

[31] XU, X., WEBER, I., STAPLES, M., et al. "A Taxonomy of Blockchain-Based Systems for Architecture Design". In: *2017 IEEE International Conference on Software Architecture (ICSA)*, pp. 243–252, April 2017. doi: 10.1109/ICSA.2017.33.

[32] SAITO, K., YAMADA, H. "What's So Different about Blockchain? Blockchain is a Probabilistic State Machine". In: *2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 168–175, June 2016. doi: 10.1109/ICDCSW.2016.28.

[33] VUKOLIĆ, M. "The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication". In: Camenisch, J., Kesdoğan, D. (Eds.), *Open Problems in Network Security: IFIP WG 11.4 International Workshop,*

iNetSec 2015, Zurich, Switzerland, October 29, 2015, Revised Selected Papers, pp. 112–125, Cham, Springer International Publishing, 2016. ISBN: 978-3-319-39028-4. doi: 10.1007/978-3-319-39028-4_9.

[34] BISTARELLI, S., MANTILACCI, M., SANTANCINI, P., et al. "An End-to-end Voting-system Based on Bitcoin". In: *Proceedings of the Symposium on Applied Computing*, SAC '17, pp. 1836–1841, New York, NY, USA, 2017. ACM. ISBN: 978-1-4503-4486-9. doi: 10.1145/3019612.3019841. Disponível em: <http://doi.acm.org/10.1145/3019612.3019841>.

[35] TSENG, L. "Bitcoin's Consistency Property". In: *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, pp. 219–220, Jan 2017. doi: 10.1109/PRDC.2017.39.

[36] SCHWARTZ, D., YOUNGS, N., BRITTO, A. "The Ripple protocol consensus algorithm", *Ripple Labs Inc White Paper*, 2014. https://ripple.com/files/ripple_consensus_whitepaper.pdf.

[37] CACHIN, C., VUKOLIC, M. "Blockchain Consensus Protocols in the Wild", *CoRR*, v. abs/1707.01873, 2017.

[38] GILBERT, S., LYNCH, N. "Perspectives on the CAP Theorem", *Computer*, v. 45, n. 2, pp. 30–36, Feb 2012. ISSN: 0018-9162. doi: 10.1109/MC.2011.389.

[39] CACHIN, C., KURSAWE, K., PETZOLD, F., et al. "Secure and Efficient Asynchronous Broadcast Protocols". In: Kilian, J. (Ed.), *Advances in Cryptology — CRYPTO 2001: 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19–23, 2001 Proceedings*, pp. 524–541, Berlin, Heidelberg, Springer Berlin Heidelberg, 2001. ISBN: 978-3-540-44647-7. doi: 10.1007/3-540-44647-8_31.

[40] ABADI, D. "Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story", *Computer*, v. 45, n. 2, pp. 37–42, Feb 2012. ISSN: 0018-9162. doi: 10.1109/MC.2012.33.

[41] BANO, S., SONNINO, A., AL-BASSAM, M., et al. "Consensus in the Age of Blockchains", *CoRR*, v. abs/1711.03936, 2017. Disponível em: <http://arxiv.org/abs/1711.03936>.

[42] LAMPORT, L. "The Part-time Parliament", *ACM Trans. Comput. Syst.*, v. 16, n. 2, pp. 133–169, maio 1998. ISSN: 0734-2071. doi: 10.1145/279227.279229. Disponível em: <http://doi.acm.org/10.1145/279227.279229>.

[43] LISKOV, B. "From Viewstamped Replication to Byzantine Fault Tolerance". In: Charron-Bost, B., Pedone, F., Schiper, A. (Eds.), *Replication: Theory and Practice*, pp. 121–149, Berlin, Heidelberg, Springer Berlin Heidelberg, 2010. ISBN: 978-3-642-11294-2. doi: 10.1007/978-3-642-11294-2_7.

[44] MILLER, A., XIA, Y., CROMAN, K., et al. "The Honey Badger of BFT Protocols." In: Weippl, E. R., Katzenbeisser, S., Kruegel, C., et al. (Eds.), *ACM Conference on Computer and Communications Security*, pp. 31–42. ACM, 2016. ISBN: 978-1-4503-4139-4. doi: 10.1145/2976749.2978399.

[45] DOLEV, D., YAO, A. "On the security of public key protocols", *IEEE Transactions on Information Theory*, v. 29, n. 2, pp. 198–208, Mar 1983. ISSN: 0018-9448. doi: 10.1109/TIT.1983.1056650.

[46] BALLANI, H., CHAWATHE, Y., RATNASAMY, S., et al. "Off by default!" 2016.

[47] KOSBA, A., MILLER, A., SHI, E., et al. "Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts". In: *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 839–858, May 2016. doi: 10.1109/SP.2016.55.

[48] ZHANG, Y., WEN, J. "An IoT electric business model based on the protocol of bitcoin". In: *2015 18th International Conference on Intelligence in Next Generation Networks*, pp. 184–191, Feb 2015. doi: 10.1109/ICIN.2015.7073830.