Design Guidelines and a Prototype Implementation for Cyber-Resiliency in IT/OT Scenarios based on Blockchain and Edge Computing

Eugenio Balistri[1], Francesco Casellato[1], Salvatore Collura[1],
Carlo Giannelli[2] *Senior Member, IEEE*, Giulio Riberto[1], Cesare Stefanelli[1] *Member, IEEE*
[1]Department of Engineering, University of Ferrara, Ferrara, Italy
[2]Department of Mathematics and Computer Science, University of Ferrara, Ferrara, Italy

The advent of the Internet of Things (IoT) and its spread in industrial environments has changed production lines, by dramatically fostering the dynamicity of data sharing and the connectivity of machines. However, such increased flexibility (also pushed by the adoption of edge devices) must not negatively affect the security and safety of industrial environments. The proposed solution adopts the Blockchain to securely store in distributed ledgers topology information and access rules, maximizing the cyber-resiliency of industrial networks. Topology information and access rules are stored and queried in a completely distributed manner, ensuring data availability even in case a centralized controller is temporarily down or the network partitioned. Moreover, Blockchain consensus algorithms foster a participative validation of topology information, to ensure the identity of interacting machines/nodes, to securely distribute topology information and commands in a privacy-preserving manner, and to trace any past modification in a non-repudiable manner. Finally, the adoption of configurable edge gateways allows to take prompt countermeasures in case potential threats are identified, by activating access rules stored in ledgers in a secure and distributed manner. In addition to solution design guidelines and architectural considerations, the paper also presents performance results achieved with our CyberChain working prototype, with the goal of not only demonstrating the feasibility of the proposed solution but also its suitability in industrial environments.

***Index Terms*—Industrial IoT, Blockchain, Cyber-resiliency, Edge Computing, Hyperledger Fabric**

I. Introduction

[1] Information Technology (IT) has been traditionally characterized by an ever increasing push towards the connectivity openness of services and their interconnection via the Internet. In sharp contrast, Operational Technology (OT) related to the monitoring and management of industrial plants and production lines has been usually characterized by static and closed networks: not only devices and machines within shop floors were not able to connect to the Internet, but they were also able to communicate one another only in a very limited manner (or were not able to communicate at all). Recently, the adoption of the Internet of Things (IoT) approach and the integration of IT and OT have greatly increased the dynamicity of data sharing and the interconnection of machines in industrial environments [1]. In fact, nowadays industrial machines are able to dynamically provide a great amount of context information, e.g., ranging from the time required to separately produce each crafted piece to vibration data provided by on-board sensors, and to receive commands, e.g., from the type and amount of objects to be crafted to the remote update of the firmware. In addition, the industrial environment more and more is characterized by a greater and more heterogeneous set of actors. In fact, the IT/OT integration is pushing towards the (partial) connectivity openness of shop floor borders, more easily integrating other devices, e.g., laptops of external specialized technicians and small-size robots that can be relocated as needed, to improve productivity or to reduce maintenance costs.

However, the increased degree of flexibility and connectivity openness of the industrial environment must not negatively affect the security and safety of the shop floor. To this purpose, traditional solutions ensure the security of devices and transmitted data by adopting a centralized approach based on a controller acting as network manager. The controller gathers information about the topology, provides system administrators with a unified point of view of the network, allows to specify security access rules to network resources with a per-subnet granularity, and interacts with gateways and firewalls to deploy access rules. Such a solution greatly simplifies the enforcement of network security, since the controller acts as management entry-point for the whole IT/OT environment. In this manner, it allows to easily take proper configuration decisions based on a complete knowledge of the environment. However, the controller also represents a single point of failure, thus a likely target for an attacker.

As better detailed later, we state that to maximize the resiliency of industrial networks it is of paramount importance that a new controller replica should be able to easily gather previous configurations at startup, to quickly restore provided services. Moreover, machine and edge devices should be able to provide and gather information without any service disruption even during controller unavailability. Finally, there is the need to also consider that the controller cannot easily verify the truthfulness of received information, thus for an attacker able to compromise a monitoring agent would be easy to inject the controller with fake information.

Corresponding author: C. Giannelli (email: carlo.giannelli@unife.it).

[1]An earlier version of this paper was presented at the IEEE SMARTCOMP 2020 Conference and was published in its Proceedings, https://ieeexplore.ieee.org/document/9239702, doi: 10.1109/SMARTCOMP50058.2020.00021.

Based on these considerations, the paper originally contributes to the state-of-the-art literature along three primary directions:

1) first of all, the proposed solution adopts the Blockchain to maximize the network resiliency (considering both functionalities and data availability) and edge nodes as monitoring agents and reconfigurable gateways. In particular, the Blockchain distributed ledger supports the creation and query of topology information in a completely distributed manner, ensuring data availability even in case the controller is temporarily down;
2) secondly, Blockchain consensus algorithms are used to foster a participative management of topology information validation. Such a solution allows to reciprocally ensure the identity validity of interacting nodes and to securely distribute topology information and commands, also allowing to trace any past modification in a non-repudiable manner;
3) finally, by exploiting edge nodes as agents and gateways in charge of monitoring and controlling the traffic, it is possible to achieve fine-grained traffic management by selectively allowing/denying packet forwarding. In this manner it is possible to isolate compromised (or potentially legit but still unknown) devices by temporarily dropping their packets.

In the rest of the paper we present our original architecture that realizes the above idea of Blockchain-based solution for increased cyber-resiliency in OT environments. By relevantly extending our previous work [2], we detail how the target topology derives from the adoption of cyber security standards identified by international organizations. In addition, we present a detailed description of our CyberChain working prototype based on Hyperledger Fabric and a thorough analysis of an extensive set of performance results that quantitatively show the suitability of our solution for industrial environments, by also presenting the efficiency of our proposal in terms of computing load and memory consumption.

II. Security standards and best practices in IT/OT environments

IT and OT integration has been the focus of recent standardisation efforts by international agencies such as the National Institute of Standards and Technology (NIST) and the International Electrotechnical Commission (IEC), with the primary goal of providing a common framework of discussion together with clear best practices to maximize the security of industrial production environments.

In particular, NIST Special Publication 800-82 Revision 2 [3] recognizes that the integration of IT and related low-cost and widely spread communication protocols with so called Industrial Control Systems (ICS) reduces the traditional isolation of industrial equipment. However, it also makes easier for attackers to get unauthorized access to machines, e.g., since wireless protocols may allow to interfere with machine workflows even without actual physical access to machines themselves. Moreover, NIST Special Publication 800-82 Revision 2 also stresses that OT environments greatly differ from IT ones in terms of risk in case of attack, since machine misbehavior within OT environments not only could cause environment damage and production loss, but could also represent a risk for human health and safety. In fact, in case of OT environments strict safety and availability requirements can sometimes overcome data privacy ones.

Based on these considerations, NIST identified several security objectives that should be achieved. Objectives most relevant in relation to the proposed solution are: restricting logical access to OT networks (e.g., by adopting unidirectional gateways and firewalls), protecting individual machines from exploitation (e.g., by disabling unused ports/services and by tracking and monitoring machine reconfigurations), restricting unauthorized data modification (both in transit and at rest), identifying security incidents (e.g., by promptly detecting security events and potentially malicious activities), maintaining functionality during adverse conditions (e.g., by ensuring that in case a single component fails the rest of the system is able to work, even if with some reduced functionalities), and supporting to restore the system after an incident (e.g., by allowing to quickly start a new controller in case the former one has crashed).

The IEC 62443 Industrial Communication Networks - Network and System Security standard [4] is composed of a series of documents detailing technical and process-related aspects of industrial cyber security. For instance, IEC 62443-1-1 focuses on terminology, concepts, and models of so called Industrial Automation and Control Systems (IACS), IEC 62443-3-2 introduces security risk assessment for system design, IEC 62443-3-3 presents technical control system requirements and foundation requirements, and IEC 62443-4-2 makes an in-depth analysis of technical security requirements for IACS.

While a detailed presentation of the IEC 62443 standard family is out of the scope of the paper, let us present some of the primary concepts it defines, i.e., zones and conduits. Zones identify groups of logical components as well as physical devices residing within a well-defined border (either physical or logical) that are characterized by common security requirements. Conduits logically group communication channels sharing common security requirements and allow components in different zones to exchange data. The definition of zones and conduits stresses the importance of segmenting the topology in multiple subnets, with the primary purpose of limiting the capability of attackers to compromise the whole production line in case they are able to compromise a single machine.

Another notable aspect of the IEC 62443 standard is the definition of seven foundation requirements that industrial environments should always satisfy: identification and authentication control (to authenticate every user, not only humans but also processes and devices), usage control (to authorize every user action by enforcing assigned privileges), system integrity (to ensure that there is no unauthorized manipulation or modification of machines, devices, and processes), data confidentiality (to ensure the confidentiality of information both flowing through conduits or stored in repositories), restricted data flow (to segment the overall systems in zones communicating via conduits thus avoiding unnecessary traffic flows), timely response to events (to promptly identify po-

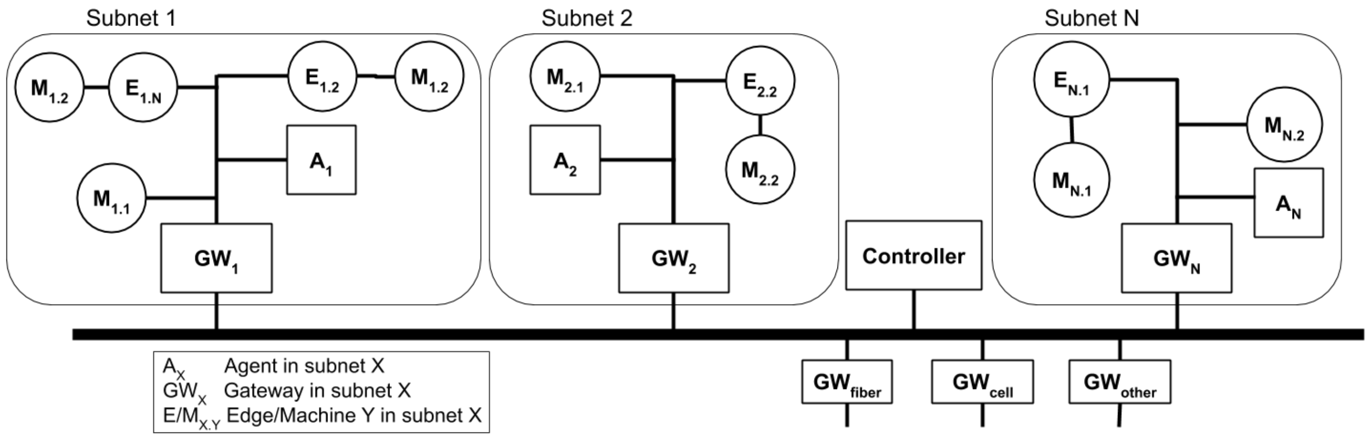


Fig. 1: Typical network topology in an industrial environment.

tential threats and notify authorities adequately), and resource availability (to allow essential services to properly work also in case of attack, even if in a degraded manner).

Taking into consideration the above standards and related requirements, we outline in Figure 1 primary components of a typical OT environment. Machines (M nodes) are either directly connected to the network or equipped with edge devices (E nodes) providing network capabilities. The whole multi-hop network is split into multiple subnets, connected one another via edge nodes acting as gateways (GW nodes). In addition, one or more gateways provide access to the Internet, e.g., one via fiber another one via cellular to provide backup connectivity, and to the network of other IT departments, e.g., administrative offices. In each subnet there are agents (A nodes) monitoring not only topology modifications but also the traffic to detect anomalies and report them to the controller. Finally, the controller resides close to the industrial environment, e.g., in a Virtual Machine (VM) in the internal datacenter, has full access to every subnet of the network, receives topology and traffic information from agents, and configures gateways. As briefly anticipated, network segmentation aims at parcelizing the overall topology in tiny subnets, to limit the visibility of each machine to a small subset of deployed machines. In this manner, in case a machine (or an edge device) is compromised, it is easier to limit horizontal attack escalation, since only few other machines are in direct visibility, thus reducing the negative impact of an attack.

Then, to ensure security and safety, OT solutions for industrial control should follow an Authentication, Authorization, and Accounting (AAA) approach, appropriately adapted to be suitable for industrial environments. In particular:

- to join the network devices have to authenticate themselves, e.g., by providing a valid X.509 certificate, and have to be in a list of known devices;
- once correctly authenticated, devices are authorized to access local and remote resources in a selective manner, based on permissions deployed and activated by the controller on gateways. To this purpose, gateways not only dispatch packets among different subnets but also act as firewalls by filtering packets flowing among different subnets as well as towards the Internet. To this purpose, gateways adopt and enforce (usually coarse-grained) access rules by considering, e.g., IP or MAC addresses of devices;
- then, monitoring agents deployed within subnets oversee the traffic generated and received by locally deployed (and already authorized) devices to identify unusual traffic patterns and rise alarms. For instance, if a machine usually sending few packets per hour suddenly starts receiving many huge packets then the agent sends an alert, since it could mean that the machine is downloading a malicious payload;
- finally, the controller traces any modification and event that could be related to an anomaly. In fact, every relevant event must be securely traced and logged, also taking note of the identity of nodes generating the event. For instance, it is required to ensure that a technician connecting to and operating on the OT environment cannot later repudiate its presence in the network. The final outcome is a clear picture about how the network evolves, also to impute eventual malfunctions to devices and technicians because of either involuntary misconfigurations or on purpose malicious attacks.

III. Blockchain and edge data management for IT/OT cyber-resiliency

To better present the proposed solution, Section III-A briefly introduces most relevant characteristics of the Blockchain technology [5]–[7]. Then, Section III-B outlines how the adoption of a Blockchain-based solution coupled with remotely configurable fine-grained gateways greatly increases the resiliency of OT environments while not limiting their flexibility.

A. Blockchain technology overview

Blockchain represents an articulated ecosystem encompassing several well-known technologies, ranging from symmetric and asymmetric cryptography to peer-to-peer distributed management based on consensus algorithms.

First of all, it is worth noting that a Blockchain can store information of any type, thus without any constraint in terms of syntax and semantic. In particular, a Distributed Ledger Technology (DLT) is a sequence of time-ordered transactions agreed among peers by adopting a distributed consensus algorithm. A Blockchain specializes the DLT by grouping transactions in immutable and linked blocks; each block is strictly correlated to the block before and the block after via secure hashes and cannot be modified once added to the Blockchain. The first block, namely the genesis block, is the only one without a previous block. New blocks can only be added after the current last block, and only if (part of) nodes hosting a copy of the ledger agree based on a given consensus algorithm. Once a block is added at the head of the ledger it cannot be modified, thus a Blockchain is an inherently incremental DLT whose past data are immutable.

Each block contains a block payload with a set of ordered transactions (each one with its own payload) and a block header specifying, among other information, a timestamp (when the block has been created), the hash of the previous block (usually 0 for the genesis block), and the hash (or Merkle tree) of the current block. Note that the hash of the current block is generated by including the timestamp, the payload (or the Merkle root of transactions' hashes [8] to increase efficiency), and even the hash of the previous block. In this manner, the hash can be used not only to verify the integrity of the current block, but also to securely chain this block with the previous one, with the ultimate goal of creating a secure chain of blocks.

In a Blockchain, smart contracts further specialize a DLT by restricting how transactions can be generated and which kind of information transactions can contain. For instance, a smart contract can limit nodes to create a new transaction only if some conditions apply, e.g., previous transactions of the same ledger have some information and external entities provide some resources. More technically, the creation of a new transaction based on a smart contract imposes that some code is executed on some data; the achieved output represents the body of the new transaction.

Despite the Blockchain is exploited either to create a generic DLT or to enforce a smart contract, to create a new transaction nodes managing copies of the same ledger must cooperate and agree one each other. To this purpose, when a node requires to add some data to the ledger, it creates a new transaction and sends it to other nodes (usually interacting in a peer-to-peer fashion). Involved nodes apply a given consensus algorithm to either accept or refuse the new transaction: in the former case, the transaction is inserted into a block and eventually added to every copy of the ledger, in the latter case the transaction is discarded.

There is a plethora of consensus algorithms that can be adopted [7] (of course, nodes collaborating to manage a given Blockchain must exploit the same algorithm). Among the others, the Proof of Work (PoW) allows to add a new block of transactions to the Blockchain only after one of the nodes involved in the consensus algorithm resolves a cryptographic puzzle, e.g., finding a nonce that added to the new block allows to get a hash with a given number of leading zeros. Such cryptographic puzzles impose relevant computational effort and time delay to create a new block and thus the longer the ledger of the Blockchain the harder it is to modify any of its blocks, ensuring the immutability of their transactions. The node resolving the puzzle is usually rewarded, e.g., in cryptocurrency, for its computational and energy effort. Practical Byzantine Fault Tolerance (pBFT) is another interesting consensus algorithm. In this case, there is no cryptographic puzzle (thus greatly reducing computational/energy overhead and also reducing the time required to create a new block) while the consensus is achieved by exchanging messages among nodes involved in the management of the Blockchain. However, since each node must send and receive multiple messages to every node hosting a ledger replica, pBFT does not scale well in huge environments with thousands of nodes involved in transaction creation.

Another important distinction among Blockchain solutions is permissionless vs. permissioned. In the former case, there is no restriction on nodes joining the Blockchain. For instance, this is the case of Bitcoin, a cryptocurrency allowing anyone to create transactions to move crytpovalue from a wallet to another. Since nodes do not know one each other (typically identified by anonymous unique identifiers), there is the need of adopting a consensus algorithm ensuring the correctness of the Blockchain, even against cyber attacks. For this reason, the PoW can be regarded as a valuable option, since a Blockchain can be compromised only if an attacker has more computational and energy power than half of the rest of the nodes. In the latter case, only authenticated nodes can participate in the network. The number of involved nodes is typically limited while the level of trust higher, and thus it is possible to adopt less computational and energy intensive solutions such as the pBFT one.

B. Traditional solution issues and Blockchain-based solution benefits

Traditional industrial control solutions are usually based on the assumption that i) it is granted the availability of a centralized node acting as controller gathering topology information and sending access rules to firewalls and ii) monitoring nodes are always trusted and behave in a regular manner by providing correct information. However, we identify some paramount issues in traditional control solutions that can greatly affect the reliability of traditional solutions. In fact, an attacker could target the controller with the purpose of limiting the capacity of network administrators to receive alarms from agents and to properly reconfigure firewalls. In addition, in case of controller unavailability, it is harder to trace events and to configure eventually added firewalls to limit damages of an ongoing attack. In fact, administrators cannot access the controller to add new topology information while the remote deployment of access rules is not possible, imposing to manually define rules and configure access rules by directly interacting with firewalls. Moreover, an attacker could target nodes with the purpose of providing untruthful information about the topology. For instance, a compromised agent could correctly authenticate itself and then send fake information about the state of local links and network resources availability.

To ensure the achievement of requirements presented in Section II (and with the final goal of maximizing OT networking resiliency), there is the need of identifying and enforcing novel and more distributed and secure approaches, overcoming aforementioned issues. In particular, we claim that in an industrial environment it is of paramount importance to ensure operational resiliency not only in case the controller is not available anymore (e.g., an attacker specifically targets it) by promptly activating a new controller replica, but also by allowing the accessibility of topology information and access rules even while the controller is not available.

In addition, to minimize the negative impact of an attack it is required to ensure the typical best practices of network segmentation in relation to network topology as well as considering data spreading and accessibility. In this manner, it is possible to limit the set of data a compromised machine can access, e.g., by providing to gateways visibility only to information related to the subnet they are connected to (rather than the whole network).

Moreover, network reconfigurability via remotely controllable gateways should be always possible, even in case of a (temporarily) partitioned network. The goal is to support fast response to identified threats by properly applying required countermeasures, not only mediated by technicians but also in a completely autonomous and decentralized manner.

Finally, due to the increased connectivity openness of the industrial environment there is the need of validating information provided by machines and edge devices. To this purpose, on the one hand, information must be univocally associated to the digital identity of the device generating it, to ensure authorship and non-repudiability. On the other hand, information provided by an authorized sender should be not granted for true (even if the sender provides a valid digital identity) but should be verified by other devices to be sure that a compromised node cannot inject the controller database with fake information.

The adoption of Blockchain provides relevant benefits, by taking advantage of its distributed and secure nature. In particular, the Blockchain dramatically increases network resiliency by:

- *improving data availability*. Topology information and network configuration are stored in distributed ledgers. In this manner it is possible to increase data availability, since in case of controller disruption nodes can still securely store new information in and retrieve configurations from the ledger. For instance, machines and edge devices can add new topology information to the ledger even if the controller is temporarily unavailable, since the ledger is managed by nodes in a distributed manner;
- *easy provisioning of fault tolerance in a seamless manner*. If the controller is down another node of the network can start behaving as new controller immediately, by gathering information from the Blockchain. In fact, once a new controller is activated, it can gather every information (even recently added ones while there was no active controller) from any node hosting a copy of the ledger, making its reactivation effortless. In this manner, on the one hand, it is easier to ensure controller fault tolerance (since related configuration data are securely distributed on multiple nodes) and, on the other hand, it is possible to also take advantage of information provided by agents during the attack to the controller itself;
- *ensuring both information authorship non-repudiation and cross-verification*. Information are added by creating new transactions, associated with the digital identity of the device (or the technician) generating the new topology data or issuing the access rule. In this manner, the Blockchain provides the notable benefit of securely tracing the authorship of information, ensuring its non-repudiability. In addition, to add a transaction (e.g., to insert a new topology information) a node has to get the approval of other nearby nodes (based on a Blockchain consensus algorithm), to ensure that a single compromised node cannot inject fake information and compromise the regular behavior of the controller. For instance, a new link advertised by a node is actually added to the topology only if other nodes of the same subnet confirm the existence of such new link;
- *allowing network segmentation in terms of data visibility*. To support network segmentation, at each subnet is associated a different ledger: nodes can access only the ledger related to the subnet they are deployed in while the controller has access to every ledger. In addition, there is a separate ledger between the controller and gateways to privately share access rules that should not be visible to machines and edge devices;
- *promptly enforcing countermeasures while ensuring production line regular workflow.* Every unknown new device should be quarantined, e.g., by greatly limiting its bandwidth or by inhibiting its ability of sending and receiving network traffic at all till not expressly allowed. On the contrary, planned newly added devices should be able to properly operate as soon as activated by OT technicians, even in case of network unavailability. To this purpose, default access rules as well as access rules for new devices going to be activated should be stored on ledgers directly accessible by gateways.

IV. Solution architecture, implementation details, and performance analysis

We have designed, implemented, and tested our CyberChain prototype by following the design guidelines presented in the previous section. In particular, CyberChain adopts Hyperledger Fabric [9], an open source project providing features to create a flexible permissioned Blockchain. Prior to present details of the implemented prototype and achieved performance results, the section introduces Hyperledger Fabric features most relevant for the comprehension of the proposed solution and also specifies why the adoption of Hyperledger Fabric is suitable to improve the cyber-resiliency of the target scenario. Interested readers can refer to [10] for an in-depth presentation of the Hyperledger Fabric project.

A. Hyperledger Fabric primary features

In Hyperledger Fabric, the overall set of nodes is called Blockchain network, supporting the coexistence of multiple ledgers, each one called channel and identifying a different set of involved nodes and a different set of smart contracts. By exploiting channels, it is possible to segment one Blockchain network in multiple private Blockchains, each one with independent consensus procedures and transaction managers, also providing to nodes per-channel access to ledgers and related data. Moreover, in Hyperledger Fabric nodes may belong to different organizations and for each organization there is a Membership Service Provider (MSP) managing credentials and identities (embedded in an X.509 certificate).

Smart contracts related to the same application are typically grouped in a unique Hyperledger Fabric chaincode [11]. Each smart contract allows to specify how a new transaction can be created, also detailing inputs and outputs. Once nodes have successfully run a smart contract it means that it is possible to create a new transaction, but it is the orderer (see below) in charge of actually adding the transaction to a new block.

Delving into finer details, primary roles of nodes are:

- *clients*, requiring the creation of a new transaction based on a specific endorsement policy, detailing how to select nodes involved in a transaction creation procedure. To this purpose, clients i) contact a subset of endorser peers as specified by the endorsement policy, e.g., at least one for each organization involved in the transaction, ii) wait for a given amount of transaction endorsements, again as specified by the endorsement policy, e.g., by adopting a vote-based consensus algorithm requiring majority/unanimous vote, and iii) send the new transaction to orderers;
- *peers*, nodes maintaining a local copy of the ledger by committing transactions and updating the ledger whenever they receive a new block. Peers can also execute smart contracts and validate transactions provided by clients;
- *committers*, nodes with the only role of maintaining the ledger and updating it whenever they receive a new block;
- *endorsers*, specific peers that can execute smart contracts whenever they receive a transaction proposal. During the endorsement of a new transaction, endorsers securely sign so-called endorsement messages (also containing transaction output, transaction id, endorser id, and endorser signature) and send it to the client requiring the new transaction;
- *orderers*, nodes collecting requests of new transactions creations, grouping multiple transactions in a block, e.g., sorting concurrent transaction requests coming from different clients, and issuing commands to peers to add new blocks on top of the ledger. Note that orderers are unaware of transaction semantics and exploit cryptographic signatures of endorsers to create new blocks.

In conclusion, let us note that Hyperledger Fabric perfectly fits the target scenario. First of all, its permissioned nature restricts the set of users/devices allowed to join the network, in compliance with basic security requirements of industrial environments. In addition, the possibility of exploiting the same Blockchain architecture to deploy multiple channels (i.e., different ledgers and smart contracts) allows to partition the access to stored data, e.g., by ensuring segmentation among different subnets. Finally, Hyperledger Fabric allows to define a consensus policy based on votes provided by endorsers, also specifying which nodes should be involved as endorsers and how many votes should be gathered. In this manner, it is possible to tune the endorsement policy, e.g., to require that a transaction related to a topology modification of a subnet must be always confirmed by at least one node of that subnet.

B. Hyperledger Fabric for industrial edge networks resiliency

Figure 2 presents primary Hyperledger Fabric components we propose to adopt to maximize the cyber-resiliency of OT environments. There are a single Gateway Ledger (GL) and multiple Subnet Ledgers (SLs). GL stores information about access rules gateways have to enforce. Each SL maintains topology information related to a given subnet and is stored (in addition to the controller) on agents, gateways, and edges of the related subnet.

Typically, agents gateways, and the controller act as Hyperledger Fabric clients and create and/or store transactions with topology information and access rules. In principle, even machines could act as Hyperledger Fabric clients as well as committers since these roles do not require high computational capabilities. However, it is advisable to avoid such a solution to reduce software complexity, to avoid the frequent need of software updates, and to save the local storage. In fact, the Hyperledger Fabric component could interfere with safety-critical industrial equipment processes and operations. Moreover, the update of industrial equipment should be done very seldomly, since an unsuccessful update could compromise the industrial equipment with safety risks and monetary loss. The adoption of additional software on industrial equipment, e.g., Hyperledger Fabric libraries, would require more frequent updates, thus increasing the risk of machine issues. Finally, it is not advisable to locally store a copy of SL on industrial equipment, since its size could become very huge, grater than the available storage. In any case, machines not able to create a new transaction can ask to their companion edge devices to create a new transaction, but with the shortcoming that the transaction will be signed with the digital certificate of the edge device.

By delving into finer details, the controller has access to every ledger, typically to query transactions from SLs and to create new GL transactions to add access rules. Network administrators access information by connecting to the controller via an Hyperledger Fabric client. Finally, orderers run on gateways and the controller. In case of new transactions with topology information, local agents and gateways are involved in the transaction consensus procedure to ensure the validity of the new information. To this purpose, for every SL there are always (at least) three nodes that contribute to the channel consensus: the agent, the subnet gateway, and the controller. For the GL, nodes participating to the consensus procedure are every gateway and the controller. In every channel the endorsement policy is based on "majority": 2 out of 3 for Subnet Channels related to topology information and 3 out of 5 for the Gateway Channel related to access rules.

Let us note that the Blockchain architecture presented in Figure 2 has been designed with the primary objective of maximizing the cyber-resiliency of the industrial infrastructure. To better outline how the adoption of the Blockchain

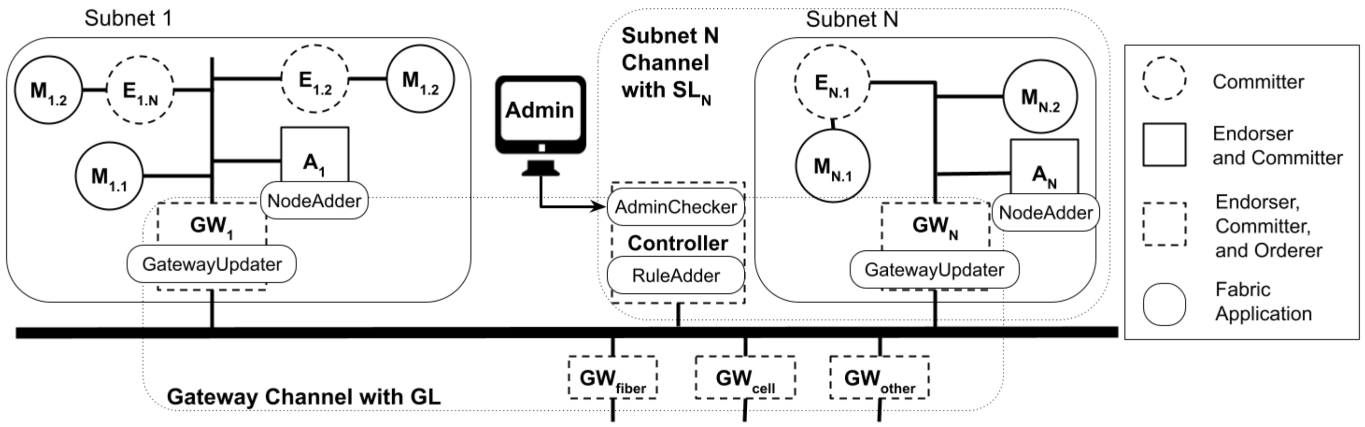


Fig. 2: Blockchain deployment in the industrial environment.

maximizes the resiliency of industrial networks, let us consider four different situations that can may happen:

1) if an edge fails, the Subnet Channel will continue to execute regularly, since the agent will be able to read and write SL since all the nodes participating in the consensus procedure (agent, gateway, and controller) continue to participate in the channel;
2) if an agent fails, the channel will continue to execute regularly, since 2 out of 3 of endorser peers (i.e., gateway and controller) are still working;
3) if a subnet gateway fails, the network will be partitioned and the subnet isolated from the rest of the network. It will not be possible to add transactions to SL (preventing from topology information modifications) both from the subnet internal agent and from the controller. However, it will be possible to query the ledger and retrieve previous topology events, since nodes of the subnet and the controller will continue to have access to a copy of the SL;
4) if the controller fails, all the subnets can continue to work regularly since there are always at least two nodes participating in the consensus procedure for every Subnet Channel. In this case, the controller does not temporarily have an overall vision of the status of the whole network (since the controller is the only one with topology information of every subnet) but can connect to one of the subnet gateways to check the status of the related subnet. Finally, when a new controller is activated it can gather the GL and every SL by interacting with active gateways, thus quickly recovering and providing to system administrators its functionalities again.

With regards to the Gateway Channel, if either a gateway or the controller node is compromised the network will continue to properly work. In particular, if the controller has been compromised a new instance of the controller will be able to connect to a gateway node to keep querying the GL.

On top of the Hyperledger Fabric components we have developed and deployed four different Hyperledger Fabric applications, exploiting Fabric Client libraries to interact with the Blockchain:

- *NodeAdder* resides on agent nodes and, when an agent identifies the presence of a new device in its subnet, it is responsible of the transaction request creation for its own SL;
- *RuleAdder* runs on the controller and it listens to new block creations on Subnet Channels. When a new block with the data of the new device on the subnet is created, it generates a GL transaction request with a network rule for the specific node recently added on SL;
- *GatewayUpdater* resides on each gateway and listens to new block creations on the Gateway Channel. When a new rule is approved on GL, GatewayUpdater executes the rule saved on the Blockchain and enables, disables, or limits the traffic accordingly to the rule;
- *AdminChecker* runs in the controller node. It is the system entry point for administrators, allowing to access topology and network data stored in the Blockchain.

Figure 3 presents primary details of the CyberChain architecture. Note that only one node per type is represented in the figure while in an actual deployment there will be n agent nodes and n gateway nodes, with n the number of subnets in the industrial environment. In particular:

- in every node there is an Hyperledger Fabric module performing as both endorser and committer;
- orderers are deployed on gateway and controller nodes;
- agent nodes host only the SL related to their subnet, gateway nodes the SL related to their subnet and the GL, and the controller node n SLs related to every subnet and the GL;
- agent nodes host the NodeAdder app, gateway nodes the GatewayUpdater app, and the controller node both AdminChecker and RuleAdder apps;
- gateway nodes are equipped with a firewall.

Agent, gateway, and controller nodes interact one another in a peer-to-peer manner to perform endorsement and commit procedures. In addition, gateway and controller nodes interact via the orderer to put endorsed transactions within new blocks. By focusing on intra-node interactions, NodeAdder identifies the presence of a new node in its subnet and makes a topology transaction request through the local endorser. After the transaction has been added to the SL, the new transaction event is notified to the RuleAdder in the controller node, thus triggering a new access rule transaction. Then,

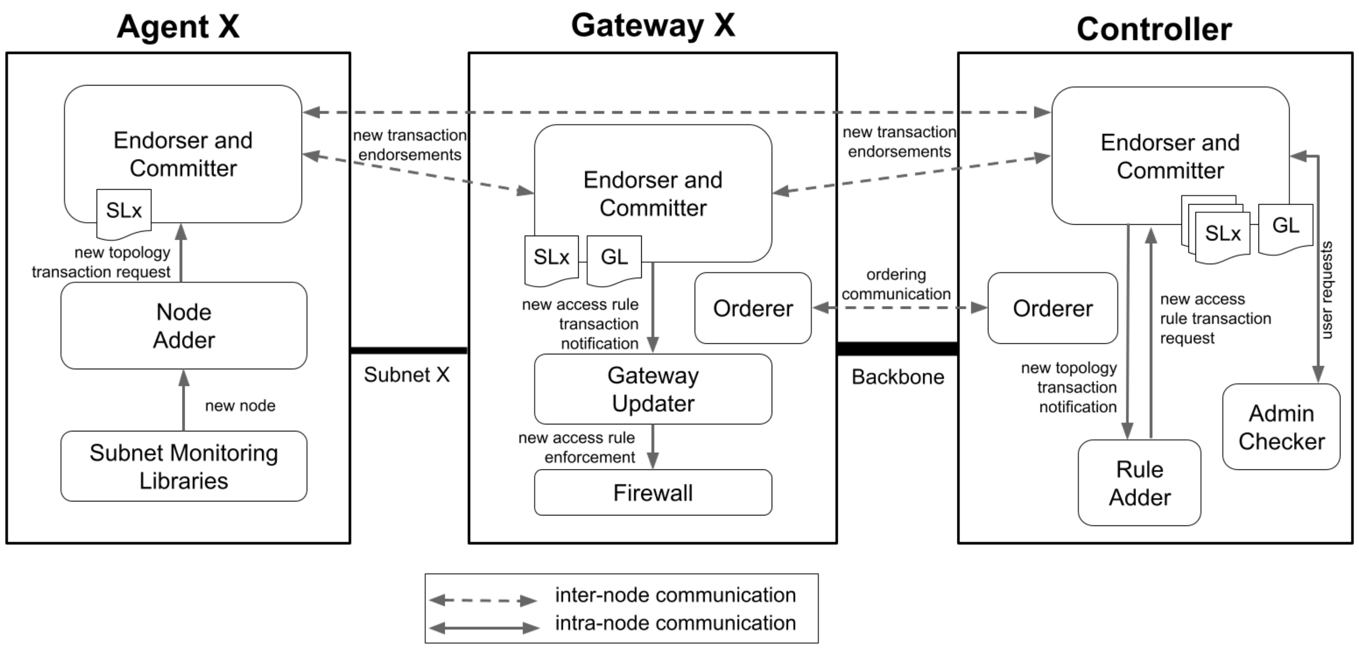

Agent X
Gateway X
Controller
Endorser and Committer
SLx
GL
new transaction endorsements
new topology transaction request
Node Adder
new node
Subnet Monitoring Libraries
Subnet X
Backbone
new access rule transaction notification
Orderer
ordering communication
Gateway Updater
new access rule enforcement
Firewall
new access rule transaction request
new topology transaction notification
user requests
Rule Adder
Admin Checker
inter-node communication
intra-node communication

Fig. 3: CyberChain architecture.

when this transaction is endorsed and added to the GL, the GatewayUpdater is notified thus triggering the enforcement of the new access rule on the firewall. Finally, the AdminChecker allows authorized users to access information available in the controller node, e.g., to retrieve the overall network topology and the set of access rules.

By delving into finer details, Figure 4 outlines the sequence diagram with the steps performed within the Blockchain network whenever a new edge node joins a subnet, considering both the new topology information in the SL and the new access rule in the GL. Note that gateways are configured following a hybrid blacklist/whitelist approach. Only previously identified and authorized devices and processes can freely communicate. On the contrary, every newly added and unknown device or process is considered as a potential threat, by default limiting its bandwidth and by dropping every packet to/from the device/process and the Internet. Then, the proposed solution interacts with the controller (via Blockchain, see details below) to verify if it is legit and, if not, it completely denies any communication of that device.

When an agent detects a new edge node within its subnet, it activates the Fabric NodeAdder application that 1) adds the topology change by inserting a new transaction in the related SL. This request 2) is notified to endorsers of the Subnet Channel, i.e., agent, gateway, and controller nodes, and must be approved by at least 2 out of 3 of them. In particular, the subnet smart contract immediately endorses the transaction request if the node performing the endorsement (and thus running the smart contract) already noticed the new edge node; otherwise, it performs a ping to verify its presence (eventually, the transaction is rejected after a temporal deadline). Once the endorsement phase is completed, 3) the signed transaction is sent by the agent to the orderer service provided by the controller and the gateways and then 4) the orderer service inserts the transaction into the following block. When the maximum number of transactions or a timeout is reached, the block is propagated to the rest of the network. Finally, 5) the block is received from every subnet Blockchain node (controller, gateway, agent, and edges) and is validated.

Once committed, the new block in the SL triggers the Fabric RuleAdder application, registered as a ledger listener on the controller node and waiting for new SL blocks. Once the application is aware of the new edge node (gathering the information from a transaction of the new block) it queries a local database to verify if the new device is legit, e.g., by retrieving the list of devices that should be active in the given subnet. If the device is not listed, the Fabric RuleAdder application notifies the system administrator about the new edge node via email, to require the manual definition of a new access rule or, more in general, to signal a potential threat. Moreover, it adds an access rule completely denying any networking capability to the unknown device. If the device is legit, the Fabric RuleAdder application verifies if system administrators have setup a specific access rule, e.g., to limit communication of that device only to a small set of other devices; otherwise, it means that the device do not have specific limitations.

In any case, 6) the Fabric RuleAdder application sends a GL transaction request (with the access rule either allowing or denying device communication) in the Gateway Channel, thus requiring the endorsement of the controller and the gateways. When the Fabric RuleAdder application 7) has received the majority of responses signed by channel endorsers, 8) it sends the signed transaction to the orderer service running on top of the controller and the gateways, which will add the new transaction into the following block. After the new block is created, 9) it is sent to every node of the Gateway Channel to allow its validation. Finally, once the controller and gateways have validated the transaction with the access rule for the new edge, 10) GL notifies to the Fabric RuleAdder application

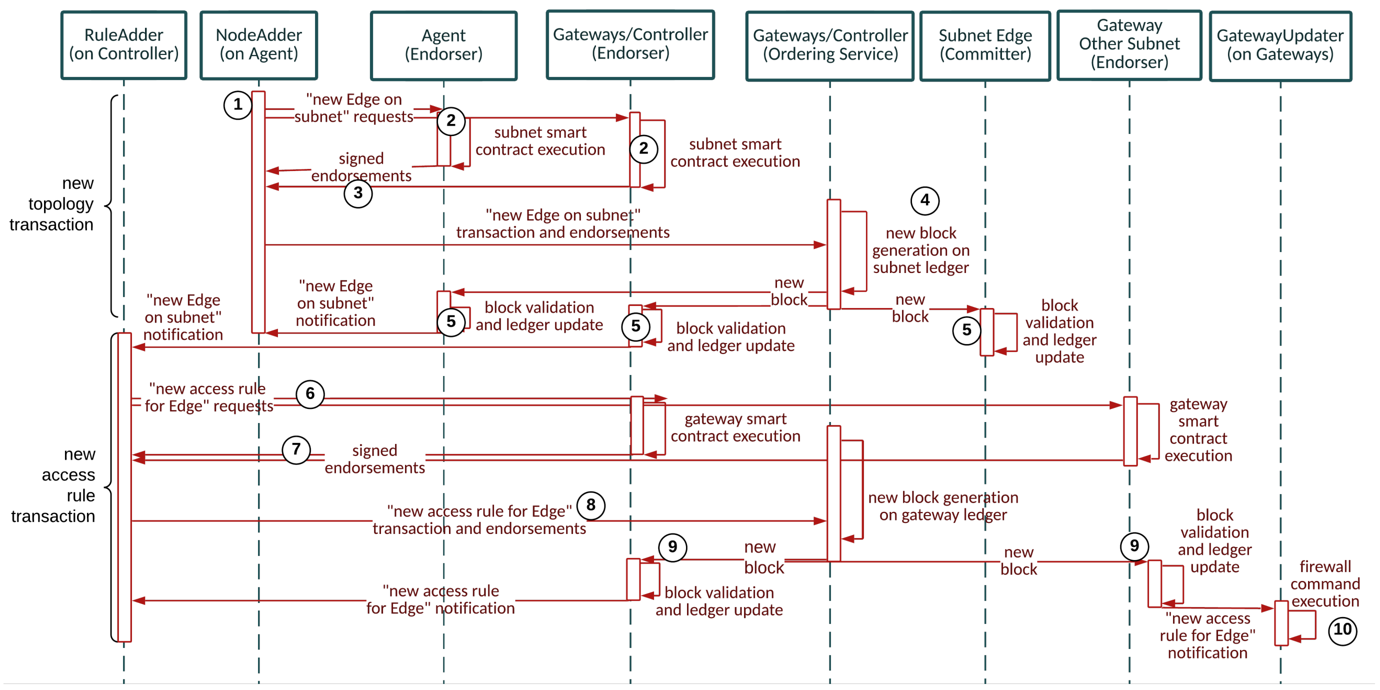

RuleAdder (on Controller)
NodeAdder (on Agent)
Agent (Endorser)
Gateways/Controller (Endorser)
Gateways/Controller (Ordering Service)
Subnet Edge (Committer)
Gateway Other Subnet (Endorser)
GatewayUpdater (on Gateways)
new topology transaction
"new Edge on subnet" requests
subnet smart contract execution
signed endorsements
"new Edge on subnet" transaction and endorsements
new block generation on subnet ledger
new block
block validation and ledger update
"new Edge on subnet" notification
new access rule transaction
"new access rule for Edge" requests
gateway smart contract execution
new block generation on gateway ledger
"new access rule for Edge" transaction and endorsements
"new access rule for Edge" notification
firewall command execution
1
2
3
4
5
6
7
8
9
10

Fig. 4: Sequence diagram of subnet and gateway ledger transaction generation.

that the new block has been correctly added. Finally, Fabric GatewayUpdater applications running on gateways act as block listeners and are notified about new blocks in an event-driven fashion. In this manner they receive new blocks (and related transactions) as soon as available, allowing gateways to gather and apply new access rules in a prompt manner.

C. CyberChain prototype implementation

To demonstrate the feasibility of the proposed solution, we have implemented the CyberChain prototype, a Blockchain solution based on the Hyperledger Fabric and on Fortigate and OPNsense programmable gateways[2]. The developed Blockchain infrastructure is composed of a variable number of Subnet Channels and a Gateway Channel. Each Subnet Channel is composed of two edge nodes, an agent node, and a gateway node, plus the shared controller one; the Gateway Channel is composed of the controller and a variable number of gateways (one for each subnet).

Every Subnet Channel adopts the same subnet chaincode to create new transactions specifying a topology modification. To test our infrastructure we have created `addNode` and `getNodes` methods to insert a new node and to display every subnet node, respectively. Both methods exploit an object written in the Go language named “node”, consisting of an identification name, a timestamp (when the node has been added to the network), and an IP address. The `addNode` method is invoked by endorsers validating transaction proposals provided by an agent that has identified a new node. The `getNodes` method is invoked by the Fabric AdminChecker application on top of the controller to get the list of available nodes and to create an overall view of the whole topology.

[2]Developed source code, configuration files, and companion documentation of our proof of concept CyberChain prototype are available at https://github.com/DSG-UniFE/CyberChain

By delving into finer details, we have developed three different versions of the `addNode` method: `addNodeNaive` simply endorsing every new node transaction request (not shown for the sake of briefness), `addNodePing` verifying that the node actually exists, and `addNodeVerify` also checking if the node is legit.

The `addNodePing` method receives as input three parameters, i.e., name, timestamp, and IP address (see Listing 1, error handling omitted). Then, it invokes the support function `checkPing` by passing the IP address (1, Point 1) and executing the ping up to three times: if no reply is received, `checkPing` returns an error and the endorsement phase is aborted, otherwise (1, Point 2a) the host is active and thus the new node can be added to the ledger, also triggering an "addNode" event (1, Point 2b). Note that we decided to create a dedicated asynchronous function (see "<-" symbol) to permit the endorsement phase to wait for the ping check, otherwise the chaincode would not wait for the ping reply.

Listing 1: addNodePing subnet chaincode.

```
func (s *NodesContract) addNodePing(
          APIstub shim.ChaincodeStubInterface,
          args []string) sc.Response {
 name := args[0]
 timestamp := args[1]
 ip := args[2]

 // 1) Check ping asynchronously
 packetsRecv := <-checkPing(ip)

 // 2) Add node
 getState, err := APIstub.GetState(name)
 if bytes.Equal(getState,[]byte("")) {
  // 2a) Create a new node
  node := Node{name, timestamp, ip}
  nodeB, marshalErr := json.Marshal(node)
  putErr := APIstub.PutState(name, nodeB)
```

```
    // 2b) Emit "add node" event
    eventPl := "Addedd Node "+name+" ip "+ip
    pl := []byte(Pl)
    err := APIstub.SetEvent("addNode",pl)

    return shim.Success([]byte(
      fmt.Sprintf("Added %s", name)))
  }
  return shim.Error("Error in addNode")
}
```

The more sophisticated `addNodeVerify` method checks not only if a node exists, but also if its type is legit (see Listing 2, error handling omitted). Note that in this manner the chaincode can perform a first-step coarse-grained filter, complementary to the more fine-grained one performed by the RuleAdder application on the controller, as depicted in Section IV-B. To support the `addNodeVerify` chaincode, OT technicians identify the list of industrial devices (i.e., their models) that can be deployed in the plant. For each device model, OT technicians gather the list of programs (and related size) it is expected the device model hosts in the default system directory and calculate the hash value. Then, the chaincode is enriched with the list of hash values related to legit industrial device models. The `addNodeVerify` method performs an ssh connection to the device (Listing 2, Point 1), remotely invokes a method to gather the list of installed programs in `/user/bin` (Listing 2, Point 2), and calculate the hash value (Listing 2, Point 3). In particular, the command `ls -al -time-style=long-iso /usr/bin` changes the format of the displayed timestamp while the utility `awk` keeps only following parameters: permissions, size, timestamp, and name.

If the hash value is in the list of permitted hash values, then it means the newly discovered device is one of the device types expected to be in the plant, otherwise the endorsement is refused (Listing 2, Point 4). Let us note that such a solution imposes additional overhead, since it is required to enrich the chaincode with the list of legit hash values and endorser nodes have to perform an additional ssh connection to the device, thus increasing the time required for the endorsement procedure. Moreover, it is expected that devices are configured with a local user with minimal permissions and known credentials allowing the remote ssh connection and the remote method invocation. However, we believe that such procedure can greatly improve the resiliency of the OT environment, since it allows endorsers not only to identify if a new device actually exists, but also to assess if it is expected that the device type is deployed in the plant.

Listing 2: addNodeVerify subnet chaincode.

```
func (s *NodesContract) addNodeVerify (
          APIstub shim.ChaincodeStubInterface ,
          args []string) sc.Response {

  name := args[0]
  timestamp := args[1]
  ip := args[2]

  // 1) Perform ssh connection
  sshConfig := &ssh.ClientConfig {
   User: "user",
   Auth: []ssh.AuthMethod{ssh.Password("pass"),},
   HostKeyCallback: ssh.InsecureIgnoreHostKey()
  }
  conn, err := ssh.Dial("tcp", ip, sshConfig)
  sess, err := conn.NewSession()
  defer sess.Close()

  // 2) Execute command on remote node
  var b bytes.Buffer
  sess.Stdout = &b
  var cmd string =
     "ls -al --time-style=long-iso /usr/bin/
     | awk 'NR>=4'
     | awk '{print $1, $5, $6, $7, $8;}'"
  sess.Run(cmd);

  // 3) Compute the hash of the output
  h := sha256.New()
  h.Write([]byte(b))

  // 4) Verify is the hash authorized,
  // if not the endorsement is refused
  str_hash := hex.EncodeToString(h.Sum(nil))
  _, found := Find(GetLegitHashList(),str_hash)
  if !found {
    return shim.Error( fmt.Sprintf (
      "ERROR: hash not found") )
  }

  // 5) Add node
  getState, err := APIstub.GetState(name)
  if bytes.Equal(getState,[]byte("")) {
   // 5a) Create a new node
   node := Node{name, timestamp, ip}
   nodeB, marshalErr := json.Marshal(node)
   putErr := APIstub.PutState(name, nodeB)

   // 5b) Emit "add node" event
   eventPl := "Addedd Node "+name+" ip "+ip
   pl := []byte(eventPl)
   err := APIstub.SetEvent("addNodeVerify",pl)

   return shim.Success([]byte(
     fmt.Sprintf("Added %s", name)))
  }
  return shim.Error("Error in addNodeVerify")
}
```

In addition, the Gateway Channel adopts the `addRule` chaincode to create new transactions containing access rules (not presented for the sake of briefness). Each transaction contains a Go object named "rule", consisting of an identification name, the rule creation timestamp, and the command field containing the actual command gateways have to execute. Since the proposed solution can adopt different programmable gateways, we have defined a common and flexible syntax for the representation of a generic access rule (see Listing 3). This representation is parsed by the `GatewayUpdater`, in charge of transforming retrieved access rules in a format suitable for the destination gateway. For example, Listing 4 represents an access rule that blocks the telnet service (port 23) from host A on subnet 192.168.50.50/24 to host B on subnet 192.168.51.51/24.

Listing 3: Syntax for an access rule.

```
data = {
  "description": <string>,
  "srcintf": [{"name": <string>}, ...],
  "dstintf": [{"name": <string>}, ...],
  "srcaddr": [{"name": <string>}, ...],
  "dstaddr": [{"name": <string>}, ...],
  "action": <string>,
  "enable": <string>,
  "schedule": <string>,
  "service": [{"protocol": <string>,
               "sport": [<integer>,
                          ...]},
              {"protocol": <string>,
               "rport": [<string>,
                          ...]},
              ...]
}
```

Listing 4: Example of access rule.

```
data = {
  "description": "Block telnet service
                  from Host A to Host B",
  "srcintf": [{"name": "port1"}],
  "dstintf": [{"name": "port2"}],
  "srcaddr": [{"name": "192.168.50.50/24"}],
  "dstaddr": [{"name": "192.168.51.51/24"}],
  "action": "deny",
  "enable": "true",
  "schedule": "always",
  "service": [{"protocol": "tcp",
               "sport": [23]}]
}
```

The rule object also contains an identifier of the entity issuing the access rule, based on the X.509 certificate of the Fabric RuleAdders application on top of the controller (together with an identifier of the technician that has added the access rule). In this manner, it is possible to trace for each access rule who and when has added it to the Blockchain. Moreover, the `addRule` chaincode (running not only on the controller but also on gateways endorsing new transactions with access rules) also verifies the validity of the related X.509 certificate, thus making harder for an attacker to inject fake access rules.

Finally, to enforce the application of access rules we have adopted two programmable gateways: FortiGate virtual appliances (FortiGate-VM) by Fortinet and the open source OPNsense solution. The former is based on the proprietary operating system FortiOS, specialized in security control by providing network functionalities for the Next-Generation Firewalls (NGFWs) offered by Fortinet. The latter implements its routing and firewalling functionalities on the HardenedBSD open source operating system. To interact with them, both gateways provide a specific REST API interface handling all the functionalities of the gateway. In particular, we used the following endpoint specialized to handle firewall policies:

- *Fortigate*, `api/v2/cmdb/firewall/policy/` to insert, delete and apply a rule;
- *OPNsense*, `/api/firewall/filter/addRule` to insert a rule, `api/firewall/filter/delRule` to delete a rule and `/api/firewall/filter/apply/` to apply a rule.

Based on the adopted programmable gateway, GatewayUpdater gathers access rules and adapts them accordingly, e.g., in case of access rule in Listing 4 it applies Listing 5 and Listing 6 for Fortigate and OPNSense gateway respectively.

Listing 5: Fortigate rule.

```
data = {
  "name": "Block telnet service from host A to
          host B",
  "srcintf": [{"name": "port1"}],
  "dstintf": [{"name": "port2"}],
  "srcaddr": [{"name": "192.168.50.50/24"}],
  "dstaddr": [{"name": "192.168.51.51/24"}],
  "action": "deny",
  "status": "enable",
  "schedule": "always",
  "service": [{"name": "Telnet"}],
  "nat": "disable",
  "logtraffic": "all"
}
```

Listing 6: OPNsense rule.

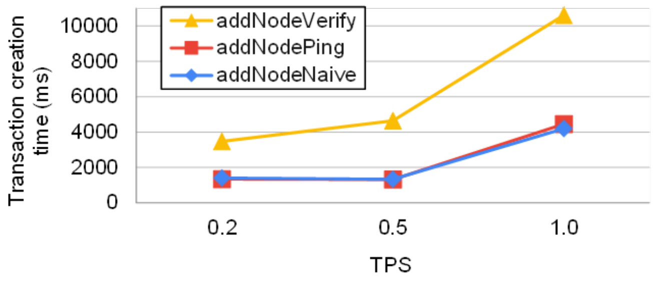
```
data = {
  "description": "Block telnet service from
                  host A to host B",
  "interface": "port1",
  "source_net": "192.168.50.50/24",
  "protocol": "TCP",
  "direction": "in",
  "destination_net": "192.168.51.51/24",
  "destination_port": "telnet",
  "action": "block"
}
```

D. Performance evaluation

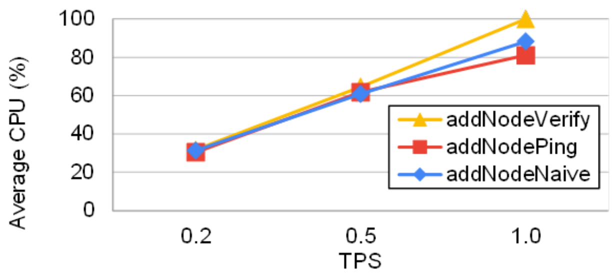
Based on the implemented CyberChain prototype, we have experimentally validated the proposed solution. The primary goal is to quantitatively verify the latency (and related resource consumption) of two primary steps of the sequence diagram in Figure 4: the addition of a topology information in the Blockchain when a new node is discovered and the application of the related new access rule.

The testbed is based on OpenStack (Train version) VMs running Ubuntu 18.04 LTS and characterized by either 1 vCPU and 1 GB RAM or 2 vCPUs and 2 GB RAM. In this manner it is possible to verify the scalability of the proposed solution with nodes with reduced/average capabilities, thus not imposing the deployment of expensive nodes within the industrial topology. We have set Hyperledger Fabric orderers to create blocks with one transaction, rather than waiting for additional transaction requests. In this manner new blocks are created without additional delay as soon as a new transaction request is endorsed, allowing prompt spread of topology information and access rules. This approach is also justified by the fact that in real conditions topology and access rule changes of an industrial subnet do not usually occur very frequently.

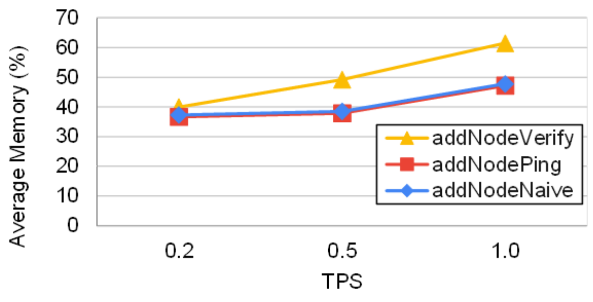
Figure 5 and Figure 6 present the average time (10 runs for test) required to generate a new topology transaction at increasing Transactions Per Second (TPS) by comparing performance achieved with `addNodeNaive`, `addNodePing`, and `addNodeVerify` chaincodes. The testbed is composed of 3 VMs, one for the agent, one for the gateway, and one



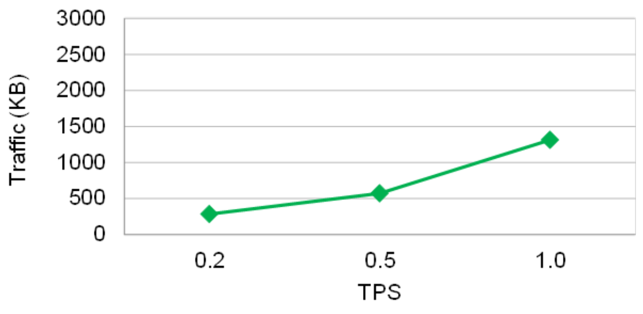
(a) Average transaction creation time (ms).



(b) Average CPU consumption (%).

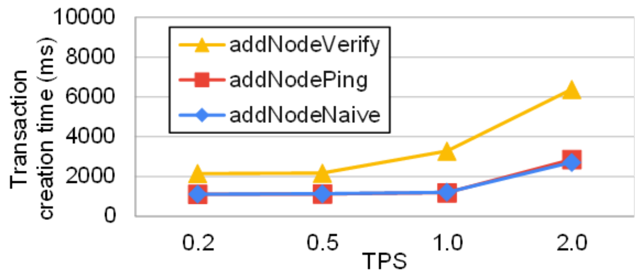


(c) Average memory consumption (%).

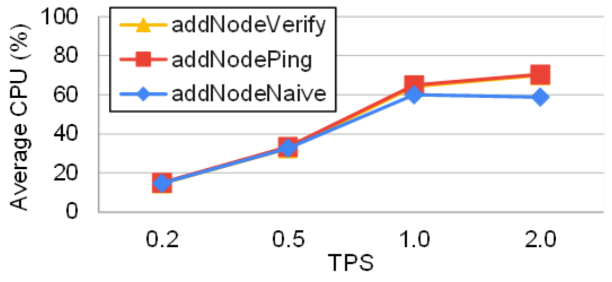


(d) Traffic (KB).

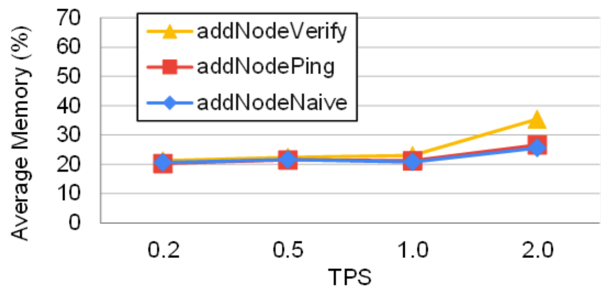
Fig. 5: Topology transaction performance at increasing TPS with 1 vCPU and 1 GB RAM.



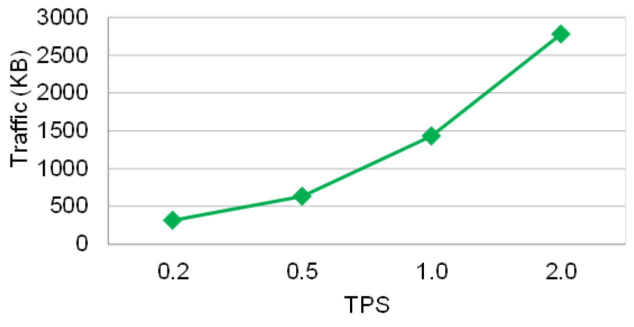
(a) Average transaction creation time (ms).



(b) Average CPU consumption (%).



(c) Average memory consumption (%).



(d) Traffic (KB).

Fig. 6: Topology transaction performance at increasing TPS with 2 vCPUs and 2 GB RAM.

for the controller. When exploiting VMs with 1 vCPU and 1 GB RAM the time to create a new topology transaction (Figure 5a) does not considerably increase till 0.5 TPS, with `addNodeVerify` imposing additional delay if compared with simpler `addNodeNaive` and `addNodePing` solutions.

By comparing memory consumption among the two different cases presented in Figure 5c and Figure 6c, it can be seen that the RAM is never saturated, with maximum occupancy at 40-60% in case of 1 GB RAM and 20-30% in case of 2 GB RAM. By focusing on CPU consumption in Figure 5b and Figure 6b, in case of 1 vCPU the utilization is about 30% for every chaincode in case of 0.2 TPS, while it reaches about 60% at 0.5 TPS, and finally it reaches the CPU saturation at 1.0 TPS with the `addNodeVerify` method, greatly increasing the time required to generate new transactions and preventing from the possibility of further increasing the TPS. Similarly, in case of 2 vCPUs the CPU usage increases with higher TPS, starting from a minimum of 15% for 0.2 TPS to values of 60-70% at 2.0 TPS. However, it is worth noting that with 2 vCPUs and 2 GB RAM (Figure 6) there is no CPU saturation. In this case the overall time required to create new transactions is considerably lower, allowing to reach 1.0 TPS

TABLE I: Access rule enforcement (ms).

	Average (ms)	Standard deviation (ms)
iptables	40	7
OPNsense	849	92
Fortigate	1086	1

without imposing relevant delays and successfully creating new transactions also in the 2.0 TPS case, even if delays start to increase.

Furthermore, we tested the traffic generated among nodes while creating new transactions (see Figure 5d and Figure 6d). To this purpose, we present the case of traffic between an agent and a gateway while creating a new topology transaction, but similar considerations can be done for other nodes and with access rule transactions. In particular, we show the generated traffic during 60 s periods while increasing the TPS. We do not provide differentiated results for `addNodeNaive`, `addNodePing`, and `addNodeVerify` since we focused on the traffic among agent and controller nodes due to the exchange of new transaction requests and new blocks, which is the same despite the adopted method.

To ensure that the generated traffic is completely and correctly measured, we have performed tests while avoiding CPU and RAM saturation by exploiting the less CPU demanding `addNodeNaive` method. In fact, in case of saturation the completion of some transactions may be greatly delayed, possibly not allowing to fully detect the generated traffic. Also note that even in case of no transaction creation there is a constant background traffic of 33 KB every 60 s due to peer-to-peer management communication among Hyperledger Fabric nodes. As Figure 5d and Figure 6d show, the overall generated traffic is limited, ranging from about 300 KB at 0.2 TPS to about 2800 KB at 2.0 TPS. Based on achieved results, we can affirm that the proposed solution does not relevantly impact on networking capabilities of the industrial network, usually with bandwidth of 100 Mbit/s or greater, thus ensuring its smooth operation.

We have also verified the time required to apply a new access rule, starting from when the Fabric RuleAdder application (deployed on the controller node) requests to add a new access rule transaction to when the access rule is actually enforced on the gateway. To this purpose, we consider the case of a single access rule (thus not increasing the TPS), since efficiency and scalability considerations for the `addRule` chaincode are similar to what presented above for topology transaction creation with `addNodeNaive` chaincode. In particular, we consider the case of a node with TCP traffic allowed only on port 80. A new access rule imposes that only TCP traffic on port 443 is allowed, therefore it is required to remove the previous rule and then to add the new one.

The time required to add a new access rule transaction and to spread it to nodes running the Fabric GatewayUpdater application is 1610 ms with 85 ms standard deviation (VMs with 1 vCPU and 1 GB RAM, average of 10 runs). Note that in this case we have considered an industrial topology composed of 3 different subnets. Thus, the gateway Blockchain involves 4 nodes, i.e., the controller plus one gateway for each subnet.

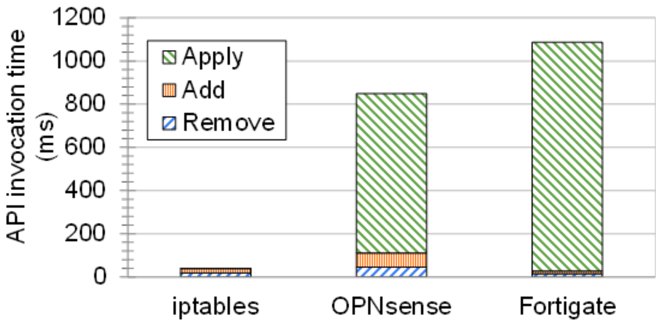


Fig. 7: Gateway access rule application.

The time required to actually enforce the access rule depends on the adopted programmable gateway. Table I presents the cumulative time required to remove a previous access rule and apply a new one in case of exploiting Fortigate and OPNsense. Note that we also show the baseline case of adopting a trivial gateway we have implemented receiving access rules in the Listing 4 format and applying them by invoking the low-level iptables command. For each test we have performed 10 runs, achieving 40 ms, 849 ms, and 1086 ms (standard deviations 7 ms, 92 ms, and 1 ms) for iptables, OPNsense, and Fortigate respectively. Achieved results are given by the sum of times required to remove a previous rule, to add a new rule, and to actually apply the rule (details in Figure 7). Iptables-based simple gateway imposes very limited delay, while OPNsense and Fortigate take much more time. Fortigate takes sightly more than OPNsense, with faster remove and add procedures but with greater time to apply the rule. Let us note that while iptables is much more efficient, Fortigate and OPNsense programmable gateways provide a wider set of capabilities, e.g., greater expressiveness to define rules. In any case, presented results demonstrate that it takes about less than 3 s to spread (1610 ms) and apply (1086 with the slowest gateway) a new access rule.

Finally, to provide a comparative analysis we have also deployed and tested a traditional solution based on a centralized controller. The main goal is to show if and how much a traditional solution is more efficient than our Blockchain-based one. To this purpose, we have deployed an IT/OT monitoring and control solution based on widely adopted threat prevention, detection, and response tools: Wazuh[3] and ElastAlert 2[4]. The former is an open source platform based on Elasticsearch supporting a plethora of features, e.g., monitoring of endpoints, threat detection, and incident response. The latter is a reporting framework interacting with Wazuh to retrieve information and providing alerts in case it identifies anomalies, traffic spikes, or other patterns that could be of interest.

In the deployed scenario Wazuh has been be used to aggregate and analyze data sent from agents, in charge of sending information about discovered nodes via the syslog protocol. To make easier the comparison with the proposed CyberChain solution, agents send to Wazuh information related to only one new discovered node in each packet (since in each topology transaction there is only one new topology information). In

[3]https://wazuh.com/

[4]https://elastalert2.readthedocs.io/en/latest/index.html

TABLE II: Traditional centralized solution performance (ms).

addNode	syslog	ElastAlert 2	Method	Total (Std. dev.)
Naive	32	578	0.4	610.4 (20.0)
Ping			6.0	616.0 (19.9)
Verify			543.0	1153.0 (20.1)

addition, we exploited ElastAlert 2 to verify if a discovered node is either actually a new one or a node already known. This verification is performed by enforcing methods discussed in Section IV-C. Then, if required ElastAlert 2 interacts with firewalls to activate access rules for newly discovered nodes.

In the traditional solution the centralized controller runs on a 2 vCPU 2 GB RAM node with CentOS 7 as operating system. Tests focused on the time required to: send via syslog a newly discovered node from an agent to Wazuh, the execution of the ElasticAlert 2 query to retrieve information from Wazuh, and the time to execute `addNodeNaive`, `addNodePing`, and `addNodeVerify` methods. Table II presents achieved performance results separately for each step, as well as the total time average and standard deviation (10 runs for each method). It takes 32 ms to send the message via syslog, 578 ms to execute the ElasticAlert 2 query, and 0.4 ms, 6.0 ms, and 543.0 ms to execute `addNodeNaive`, `addNodePing`, and `addNodeVerify` methods, respectively. Overall, it takes 610.4 ms for `addNodeNaive`, 616.0 ms for `addNodePing`, and 1153.0 ms for `addNodeVerify`. Note that, as already done when presenting CyberChain results in Figure 5 and Figure 6, we have not considered the time required to invoke firewall API, separately presented in Figure 7.

In conclusion, let us note that the centralized traditional solution takes less time if compared with our CyberChain solution in case of no saturation. For instance, with 2 vCPUs 2 GB RAM CyberChain takes 2146 ms for the `addNodeVerify` topology transaction and 1610 ms for the access rule one, while the traditional centralized solution takes 1153.0 ms to dispatch topology information and it does not need to distribute access rules on other nodes (since only stored in the centralized controller, representing a single point of failure). However, let us stress that, on the one hand, delays are at the same order of magnitude, and on the other hand, the traditional centralized solution does not support (among other features) secure information cross-validation based on Blockchain consensus algorithms, network resiliency in case of network partitioning, and data immutability and traceability as our CyberChain solution does.

E. Discussion

As presented above, the CyberChain solution successfully achieves the objective of enhancing cyber-resiliency in IT/OT environments by increasing information validation and data availability. Moreover, achieved performance results demonstrate its suitability in terms of TPS by considering the topology variability of a typical industrial environment. However, we recognize that the use of the Blockchain technology can lead to drawbacks due to storage occupation if not correctly set. To evaluate its feasibility we consider the storage required in a real-world scenario, considering that the size of topology and access rule transactions are 6.5 KB and 7.5 KB, respectively. Considering a scenario with n subnets and assuming that for each subnet agents identify r topology modifications every day, each day there would be a memory occupation due to new transactions equal to $r * 6.5$ KB for an agent node (since storing only one SL), equal to $r * (6.5 + n * 7.5)$ KB for a gateway node (since storing an SL and the GL and considering that every topology modification may require a new access rule) and equal to $r * n * (6.5 + 7.5)$ KB for the controller (since storing every SL and the GL).

Considering a case with 4 subnets ($n = 4$) and every subnet each day has 4 topology modifications ($r = 4$), after 10 years there would be an occupation of less than 95 MB for agent nodes, about 533 MB for gateway nodes, and about 818 MB for the controller node. Considering the target scenario, it is likely that the controller is hosted on a powerful machine, without stringent storage constraints. Similar considerations apply to gateway nodes, usually devices in charge of processing high traffic rates and thus equipped with medium CPU, memory, and storage capabilities. Finally, agent nodes could be hosted on edge devices with limited storage capabilities, but in any case required storage can be considered suitable even for small single board computers. Considering a much more challenging (and, in our opinion, unlikely) case, if topology modifications happen in each subnet every 10 minutes, then in 10 years the storage consumption would be 3.4 GB for agent nodes, 19.2 GB for gateway nodes, and 29.4 GB for the controller node. Even in this case, we believe that required storage should not be a concern for a typical edge node, possibly equipped with an additional memory card.

Another possible drawback that could represent a limitation in the deployment and adoption of the proposed solution is the degree of software customizability of nodes involved in the target environment. In fact, the CyberChain solution requires to deploy and run custom software modules on agent, gateway, and controller nodes, as described in Figure 3. In particular, Hyperledger Fabric libraries should run on gateway nodes to make them capable of taking part of the endorsement phase. Let us note that while agent and controller nodes typically support software customization, industrial gateway nodes also acting as firewalls could be less flexible to ensure node security. For instance, by considering Fortigate and OPNsense only the latter allows the installation of software modules via the pkg package manager, by downloading packages from a software repository managed by OPNsense. Furthermore, to install external software modules not available in the repository there is the need of exploiting specific compiling and building tools. On the one hand, we believe that in the future the degree of gateway flexibility will increase, thus allowing to customize their behavior. On the other hand, to easily address this issue even in case of closed gateways it is possible to deploy companion edge devices (coupled with the gateway node) in charge of running required CyberChain software modules.

V. Related Work

Blockchain has been recently proposed in several industrial use cases, pushed by its capability of logging events in a distributed and secure manner without requiring any trusted centralized authority. For instance, Blockchain has been adopted in real estate to guarantee data reliability in management workflows [12] or in supply chains to remove any central authority while supporting tracing, tracking, and business transactions [13]–[15]. To this regard, [16] presents a survey of recent state-of-the-art contributions exploiting Blockchains to increase efficiency, reliability, and transparency of the supply chain.

The Blockchain has also been adopted in IoT environments [17] to support the security of processes and data management. For instance, [18] outlines how the adoption of the Blockchain can support the secure sharing of data among flying unmanned autonomous vehicles, while [19] exploits the Blockchain to share partial models among smart home environments to securely adopt a federated learning solution. The HomeChain solution [20] focuses on smart-home scenarios and supports mutual authentication among devices in the same environment. In particular, it exploits Blockchain, group signature, and message authentication to provide a range of features, from auditing of users' activity in a reliable manner to authentication of home gateways. Fortified-Chain [21] adopts Blockchain in the Medical IoT (MIoT) scenario to store Electrical Health Records (EHR) in a decentralized manner while not compromising system security and privacy. In particular, the proposed architecture introduces an hybrid computing paradigm with Blockchain-based Distributed Data Storage System (DDSS) to avoid the exploitation of a cloud-centric EHR, typically characterized by high latency, high storage cost, and single point of failure. In addition, the adoption of smart contracts allows to alert medical emergency services in an automatic manner.

Recent contributions started to focus on Industry 4.0 scenarios [22], [23]. For instance, [24] analyzes benefits of adopting the Blockchain in the automotive industry to support trusted and cyber-resilient information distribution among currently non-collaborative organizations. [25] presents a solution adopting the Blockchain to support servitization of ice cream machines, by exploiting smart contracts to ensure the validity of data related to machine usage. Similarly, in [26] the Blockchain is used to allow machine owners to share idle machines capacity, by securely storing in the Blockchain relevant events related to machine usage. [27] supports actors of the manufacturing supply chain to make agreements and payments based on the Blockchain in a secure and distributed manner, without any intermediary. Since agreements are stored in transactions and thus impose the payment of a fee, the proposed solution adopts an hybrid approach by allowing actors to pay only for agreements that actually need to be secured. [28] exploits the Blockchain to ensure data privacy of IoT devices by adopting smart contracts to validate connection rights based on predefined privacy permission settings predefined and on the availability, for each IoT device, of a set of stored of known misbehaviours. The BPIIoT solution [29] exploits the Blockchain to develop an IIoT platform, with the notable benefit of addressing issues related to the lack of security, trust, and island connectivity typical of many IIoT environments. In particular, BPIIoT exploits Blockchain smart contracts as a mechanism to achieve an agreement among service consumers and manufacturing resources supporting the delivery of on-demand manufacturing services. The BASA solution [30] allows cross-domain authentication IIoT environments. In particular, it allows to authenticate devices by other devices even if in a different administrative domain, also without requiring to expose identity information. In this manner, BASA is able to ensure trust in untrusted domains without the need of adding any third party entity.

While recent research efforts have focused the attention in adopting the Blockchain to support several industrial use cases (as above reference demonstrate), in the literature only few work specifically proposes networking-related solution based on Blockchain. For instance, the Blockchain has been adopted to dynamically manage 5G slices and, e.g., avoid double-spending of the same radio frequency slice [31]. [32] outlines that the Blockchain can represent a common negotiation platform in factory environments, by allowing involved actors to save coordination and transaction costs, speed up slicing agreements, and also ensure a level of trust enough to enable automatic agreements. [33] manages the Software Defined Networking (SDN) control layer on top of a distributed ledger, by supporting the authentication of IoT devices and ensuring their secure access. Moreover, it exploits the Blockchain to securely store traces and provide them for following forensic analysis. The DistBlockNet solution [34] jointly exploits SDN and Blockchain in IoT environments to manage networking information in a peer-to-peer, secure, and verifiable manner. In particular, DistBlockNet is able to efficiently detect attacks in the IoT network by taking advantage of the peer-to-peer nature of the Blockchain while also limiting the imposed overhead. Note that this solution focuses on the efficient distribution and management of OpenFlow information, while our solution adopts a more general point of view with the paramount objective of maximizing the resiliency of industrial networks notwithstanding (part of) network nodes are compromised. [35] considers trust issues between data and control planes in SDN environments. The proposed solution adopts an efficient mechanism based on Blockchain and edge computing to securely verify that SDN flows are legit. Readers interested in an in-depth analysis of Blockchain adoption in IoT scenarios can refer to [36], [37].

Similarly to our approach, the BoSMoS solution [38] aims at defending OT environments from cyber-attacks, by focusing on unauthorized software updates. To this purpose, it securely stores in the Blockchain a snapshot of the legit machine software status, thus it can later verify either if it is the same or if has been changed by an intruder. Thus, this solution allows to identify threats, but it does not allow to increase the resiliency of the OT environment. The IoTCop solution [39] considers the possibility that part of IoT devices in the same environment can be compromised, e.g., based on malicious firmware attacks, with the need of isolating them. To this purpose, it adopts Hyperledger Fabric to ensure the possibility of identifying compromised devices and enforce security policies, as long as the majority of devices are not compromised.

In conclusion, papers above demonstrate the current interest of academic and industrial researchers in adopting the Blockchain technology in industrial environments. We believe that the proposed solution represents a relevant contribution, originally proposing Blockchain to ensure network cyber-resiliency by ensuring data availability and fast recovery of functionalities even in case the network is temporarily partitioned.

VI. Conclusions

The increasing integration of IT and OT environments is pushing towards dynamicity and connectivity openness of industrial solutions, paving the way for the fourth industrial revolution. However, as demonstrated by recent standardization efforts of international organizations such as NIST and IEC, there is the need of identifying and enforcing suitable management solutions ensuring the cyber-resiliency of industrial environments, also in case (part of) nodes are compromised or the network partitioned. To this purpose, the paper originally proposes the adoption of Blockchain and edge computing to ensure data availability (and fast recovery of functionalities) by spreading topology information and access rules in a peer-to-peer and secure fashion. In addition, the adoption of the Blockchain pushes for a participative validation of topology information and non-repudiability of issued commands, based on consensus algorithms reducing the risk of injection of fake information by compromised nodes.

We believe that presented solution guidelines and architectural considerations together with the availability of our CyberChain working prototype based on the Hyperledger Fabric Blockchain represent a solid contribution to support the resiliency of industrial environments. Furthermore, presented performance results achieved with our prototype demonstrate the feasibility of the proposed solution and encourage in further investigating this novel cyber-resiliency approach. In particular, we are considering the possibility of adopting the SDN approach to autonomously gather traffic information and manage routing rules based on the OpenFlow protocol. Moreover, we intend to widen our performance analysis by testing the proposed solution in real-world industrial plants.

Acknowledgment

Partially funded by POR-FESR Emilia Romagna 2014-2020, "I4S: Industria 4.0 Sicura" project and "SmartChain" project.

References

[1] A. Corradi et al., "Smart Appliances and RAMI 4.0: Management and Servitization of Ice Cream Machines", IEEE Transactions on Industrial Informatics, vol. 15, no. 2, pp. 1007-1016, Feb. 2019.
[2] E. Balistri, F. Casellato, C. Giannelli, C. Stefanelli, "Blockchain for Increased Cyber-Resiliency of Industrial Edge Environments", 2020 IEEE International Conference on Smart Computing (SMARTCOMP), Bologna, Italy, 2020, pp. 1-8.
[3] K. Stouffer. S. Lightman, V. Pillitteri, M. Abrams, A. Hahn, "Guide to Industrial Control Systems (ICS) Security, SP 800-82 Rev. 2", May 2015.
[4] International Electrotechnical Commission, "IEC 62443: Industrial network and system security", available online at https://www.iec.ch/cyber-security
[5] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System”, available online at https://bitcoin.org/bitcoin.pdf, 2008.
[6] D.J. Yaga, P.M. Mell, N. Roby, K. Scarfone, “Blockchain Technology Overview”, NIST Interagency/Internal Report (NISTIR) - 8202, 2018.
[7] M. Belotti, N. Božić, G. Pujolle, S. Secci, “A Vademecum on Blockchain Technologies: When, Which and How”, IEEE Comm. Surveys & Tutorials, 2019.
[8] M. Wu, K. Wang, X. Cai, S. Guo, M. Guo, C. Rong, "A Comprehensive Survey of Blockchain: From Theory to IoT Applications and Beyond", IEEE Internet of Things Journal, vol. 6, no. 5, pp. 8114-8154, Oct. 2019.
[9] "Hyperledger Fabric: a Blockchain Platform for the Enterprise", available online at https://hyperledger-fabric.readthedocs.io/en/release-2.3/
[10] E. Androulaki et al, “Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains”, Proceedings of the 30th EuroSys Conf. (ACM EuroSys ’18), April 2018.
[11] “Smart Contracts and Chaincode”, available online at https://hyperledger-fabric.readthedocs.io/en/release-2.3/smartcontract/smartcontract.html
[12] M. Lia, L. Shenc, G.Q. Huangc, “Blockchain-enabled workflow operating system for logistics resources sharing in E-commerce logistics real estate service”, Computers & Industrial Eng. (Elsevier), vol. 135, 2019.
[13] K. Salah, N. Nizamuddin, R. Jayaraman, M. Omar, “Blockchain-Based Soybean Traceability in Agricultural Supply Chain”, IEEE Access, vol. 7, 2019.
[14] M. I. S. Assaqty et al., "Private-Blockchain-Based Industrial IoT for Material and Product Tracking in Smart Manufacturing", IEEE Network, vol. 34, no. 5, pp. 91-97, September/October 2020.
[15] W. Alkhader, N. Alkaabi, K. Salah, R. Jayaraman, J. Arshad and M. Omar, "Blockchain-Based Traceability and Management for Additive Manufacturing", IEEE Access, vol. 8, pp. 188363-188377, 2020.
[16] G. Perboli, S. Musso, M. Rosano, "Blockchain in Logistics and Supply Chain: A Lean Approach for Designing Real-World Use Cases", IEEE Access, vol. 6, 2018.
[17] D. Miller, “Blockchain and the Internet of Things in the Industrial Sector”, IT Professional (IEEE), vol. 20, no. 3, 2018.
[18] P. Abichandani, D. Lobo, S. Kabrawala. W. McIntyre, "Secure Communication for Multiquadrotor Networks Using Ethereum Blockchain", IEEE Internet of Things Journal, vol. 8, no. 3, pp. 1783-1796, 1 Feb., 2021.
[19] Y. Zhao et al., "Privacy-Preserving Blockchain-Based Federated Learning for IoT Devices", IEEE Internet of Things Journal, vol. 8, no. 3, pp. 1817-1829, 1 Feb., 2021.
[20] C. Lin, D. He, N. Kumar, X. Huang, P. Vijayakumar and K. R. Choo, "HomeChain: A Blockchain-Based Secure Mutual Authentication System for Smart Homes", IEEE Internet of Things Journal, vol. 7, no. 2, pp. 818-829, Feb. 2020.
[21] B. S. Egala, A. K. Pradhan, V. R. Badarla and S. P. Mohanty, "Fortified-Chain: A Blockchain Based Framework for Security and Privacy Assured Internet of Medical Things with Effective Access Control", IEEE Internet of Things Journal (accepted for publication).
[22] T.M. Fernández-Caramés, P. Fraga-Lamas, "A Review on the Application of Blockchain to the Next Generation of Cybersecure Industry 4.0 Smart Factories", IEEE Access, vol. 7, pp. 45201-45218, 2019.
[23] J. Leng et al., "Blockchain-Secured Smart Manufacturing in Industry 4.0: A Survey", IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 51, no. 1, pp. 237-252, Jan. 2021.
[24] P. Fraga-Lamas, T. M. Fernández-Caramés, "A Review on Blockchain Technologies for an Advanced and Cyber-Resilient Automotive Industry", IEEE Access, vol. 7, pp. 17578-17598, 2019.
[25] E. Balistri et al., "Servitization in the Era of Blockchain: the Ice Cream Supply Chain Business Case", 2020 International Conference on Technology and Entrepreneurship (ICTE), Bologna, Italy, 2020, pp. 1-8.
[26] S. Geiger, D. Schall, S. Meixner, A. Egger, "Process traceability in distributed manufacturing using blockchains", Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC 19), New York, NY, USA, 2019, 417-420.
[27] B. Kaynak, S. Kaynak and O. Uygun, "Cloud Manufacturing Architecture Based on Public Blockchain Technology", IEEE Access, vol. 8, pp. 2163-2177, 2020.
[28] F. Loukil, C- Ghedira-Guegan, K. Boukadi, A.N. Benharkat, E. Benkhelifa, "Data Privacy Based on IoT Device Behavior Control Using Blockchain", ACM Trans. Internet Technol. 21, 1, Article 23, January 2021.
[29] L. Bai, M. Hu, M. Liu and J. Wang, "BPIIoT: A Light-Weighted Blockchain-Based Platform for Industrial IoT", IEEE Access, vol. 7, pp. 58381-58393.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2021.3104624, IEEE Internet of Things Journal

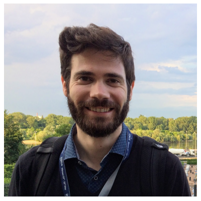
[30] M. Shen et al., "Blockchain-Assisted Secure Device Authentication for Cross-Domain Industrial IoT", IEEE Journal on Selected Areas in Communications, vol. 38, no. 5, pp. 942-954, May 2020.
[31] D.B. Rawat, "Fusion of Software Defined Networking, Edge Computing, and Blockchain Technology for Wireless Network Virtualization", IEEE Communications Magazine, vol. 57, no. 10, pp. 50-55, October 2019.
[32] J. Backman, S. Yrjölä, K. Valtanen, O. Mämmelä, "Blockchain network slice broker in 5G: Slice leasing in factory of the future use case", 2017 Internet of Things Business Models, Users, and Networks, 2017.
[33] M. Pourvahab, G. Ekbatanifard, "An Efficient Forensics Architecture in Software-Defined Networking-IoT Using Blockchain Technology", IEEE Access, vol. 7, pp. 99573-99588, 2019.
[34] P.K. Sharma, S.Singh, Y. Jeong, J. H. Park, "DistBlockNet: A Distributed Blockchains-Based Secure SDN Architecture for IoT Networks", IEEE Communications Magazine, vol. 55, no. 9, pp. 78-85, Sept. 2017.
[35] J. Hu, M. Reed, N. Thomos, M. F. AI-Naday and K. Yang, "Securing SDN-Controlled IoT Networks Through Edge Blockchain", IEEE Internet of Things Journal, vol. 8, no. 4, pp. 2102-2115, 15 Feb., 2021.
[36] K. Peng, M. Li, H. Huang, C. Wang, S. Wan and K. -K. R. Choo, "Security Challenges and Opportunities for Smart Contracts in Internet of Things: A Survey", IEEE Internet of Things Journal (accepted for publication).
[37] W. Zhao, C. Jiang, H. Gao, S. Yang and X. Luo, "Blockchain-Enabled Cyber-Physical Systems: A Review", IEEE Internet of Things Journal, vol. 8, no. 6, pp. 4023-4034, 15 March, 2021.
[38] S. He, W. Ren, T. Zhu and K. R. Choo, "BoSMoS: A Blockchain-Based Status Monitoring System for Defending Against Unauthorized Software Updating in Industrial Internet of Things", IEEE Internet of Things Journal, vol. 7, no. 2, pp. 948-959, Feb. 2020.
[39] S. S. Seshadri et al., "IoTCop: A Blockchain-Based Monitoring Framework for Detection and Isolation of Malicious Devices in Internet-of-Things Systems", IEEE Internet of Things Journal, vol. 8, no. 5, pp. 3346-3359, 1 March, 2021

Eugenio Balistri received his B.Sc degree in management and computer engineering from the University of Palermo, Italy and his M.Sc. in computer and automation engineering from the University of Ferrara, Italy. He is currently a research assistant at the Engineering Department of the University of Ferrara. His research activities currently focus on Industrial Internet of Things, Blockchain technologies, smart water metering, and 5G networks.

Francesco Casellato received his B.Sc. degree in computer engineering and his M.Sc. degree in computer science engineering from the University of Ferrara, Italy. He was a research assistant at the Engineering Department of the University of Ferrara until July 2021. His research interests focus on smart cities, smart water metering, Industrial Internet of Things, blockchain and private cloud.

Salvatore Collura received his B.Sc degree in management and computer engineering from the University of Palermo, Italy and his M.Sc. in computer and automation engineering from the University of Ferrara, Italy. He is currently a research assistant at the Engineering Department of the University of Ferrara. His research activities currently focus on Industrial Internet of Things, cyber security and cyber resiliency in Industry 4.0 and smart utility applications.

Carlo Giannelli received the Ph.D. degree in computer engineering from the University of Bologna, Italy, in 2008. He is currently an Associate Professor in computer science with the University of Ferrara, Italy. His primary research activities focus on Industrial Internet of Things, Software Defined Networking, Blockchain technologies, cyber security in Industry 4.0, location/based services, heterogeneous wireless interface integration, and hybrid infrastructure/ad hoc and spontaneous multi-hop networking environments.



Giulio Riberto received his B.Sc. degree in information engineering and his M.Sc. degree in computer science engineering from the University of Ferrara, Italy. He is currently a research assistant at the Engineering Department of the University of Ferrara. His research interests focus on smart cities, smart utility applications, Industrial Internet of Things, cyber security, and cyber resiliency of Industry 4.0 plants.

Cesare Stefanelli received his Laurea degree in electronic engineering and his Ph.D. in computer science engineering from the University of Bologna, Italy. He is currently a full professor of computer science engineering in the Engineering Department of the University of Ferrara. His research interests include distributed and mobile computing, adaptive and distributed multimedia systems, network and systems management, and network security.