Journal of Theoretical and Applied Information Technology
15th July 2022. Vol.100. No 13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

OSMOTIC COMPUTING ENHANCEMENT BASED ON BLOCKCHAIN APPROACH

ISLAM GAMAL [1], HALA ABDEL-GALIL [2], ATEF GHALWASH[3]

[1]Teacher Assistant, Helwan University – Faculty of Computers and Artificial Intelligence, Egypt
[2]Associate Professor, Helwan University – Faculty of Computers and Artificial Intelligence, Egypt
[3]Professor, Helwan University – Faculty of Computers and Artificial Intelligence, Egypt
E-mail: islam.gamal@live.com

ABSTRACT

Osmotic computing is an emerging concept for managing aggressively distributed IoT applications facilitating IoT ecosystem to compensate the limitation of computational, power consumption resources' capabilities offer the extension for IoT system to use the power of IoT, edge, and cloud resources within a federated environment to grantee utilization and efficiency of these re-sources. In essence, osmotic computing solves end-to-end IoT solution issues related to the orchestration, deployment, and reconfiguration of osmotic abstraction microelements (MELs) that are constructed and federated across cloud, edge and IoT infrastructures. In consequence, while enabling the dynamic orchestration of the osmotic resources, there is opportunity for attacks and security threats resulting from the continuous reconfiguration of the osmotic topology where providing security and privacy is a key issue in which malicious software/hardware injection at-tacks are compromising data security, integrity, and confidentiality. Blockchain is a fundamental technology, as a distributed and decentralized ledger, is a nominated solution to face the security limitations of osmotic computing solution. The objective of this research is to propose a new blockchain based osmotic computing architecture to leverage private permissioned Hyperledger Fabric (HLF) blockchain aiming to present a security guard by tracking MELs operations into the blockchain ledger to offer authorized access control and establish the root of trust for applying the principal features of traceability, immutability, transparency, and auditability. The implemented architecture prototype is tested on both simulated and real-life environments to validate the architecture hypothesis of running a transparent and traceable osmotic IoT ecosystem maintaining the required security levels.

Keywords: *Osmotic Computing, Blockchain, Hyperledger Fabric.*

1. INTRODUCTION

In 2017, the global Internet of Things (IoT) market surpassed $100 billion in revenue for the first time, and forecasts indicate that this figure will rise to around $1.6 trillion by 2025 [1]. The Internet of Things (IoT) is a network of interconnected computing devices with unique identifiers that can transfer data over a network without the need for external communication. Small IoT devices, from sensors to beacons to wearables, have their own processing power, and create a massive amount of data for processing and analytics. It is often inefficient to send all this data to the cloud for processing. Also, data transfer relies on network availability and can pose security, data protection, and privacy challenges. IoT analytics is moving from the cloud to the edge because of security, latency, autonomy, and cost [2]. However, distributing and managing loads to several increasing numbers of nodes at the edge of the network can be a tedious and exhausting process.
Osmotic Computing is a dynamically managing heterogeneous resources throughout Cloud, Edge, and IoT resources using an adaptive computational model. [3], where the computation is shared between all the three layers' resources according to the demand. Microelements (MELs) are the core abstraction component of osmotic computing which encapsulate objects (microservices or microdata) deployed in lightweight virtualized containers for both loose coupling away of the hardware and operating system dependencies, to elastically manage resources over federated infrastructures. Through osmotic dynamic orchestration, MELs can migrate across IoT, edge, and cloud dedicated infrastructures in accordance with different operational conditions correlated to infrastructure

and application performance, quality of service (QoS), and load balancing. Microelements can migrate within their Membrane (Software Defined Membrane), which is another osmotic abstraction that represents a virtual environment based on the core infrastructure (e.g., cloud, edge, and IoT infrastructure resources), without having any contact and interaction with the external world. However, the growth of an osmotic solution means that the number of traversing MELs is increasing exponentially, in which security and privacy is presented as a key issue in the osmotic computational model.

The issue from osmotic perspective is described as malicious software/hardware injection where software MELs or even osmotic nodes are injected to osmotic network that cause destruction on the efficacy of existing osmotic ecosystem. The injected entities have the ability to sharing, receiving, storing, processing, redirecting, and transmitting data from/to system which implies risks of data privacy, integrity and confidentiality.

Blockchain is an immutable/unchanged shared ledger that enables the development of tracking transactions and tracing assets in a business network [4]. An asset can be physical (e.g., a cash money, vehicle, cash, or any tangible property) or virtual (e.g., stocks, intellectual property, computing, copyrights, cryptocurrency). Virtually valuable property can be traded and traced on a blockchain network, preventing threats, and delivers cost savings with efficiencies. Blockchain maintains always increasing list of ordered records called blocks, which are added in a consecutive order. Hence, blockchain has the potential to solve the security and privacy issues associated with osmotic computing.

In this paper, osmotic computing security enhancement based on blockchain approach is presented where blockchain capabilities are emerged with osmotic computing aiming to challenge the security and privacy issues presented by the osmotic architecture. Blockchain's consensus algorithms validate every transaction, ensure that the MELs including both microservices and microdata are transmitted by designated infrastructure nodes is verified and validated, and have not been tampered with during osmotic transactions or even through injected MELs or nodes. This could be accomplished through the integration of distributed ledger blockchain to the MELs orchestration process ensuring the ownership and integrity of MELs.

The remainder of the paper is structured as the following: the background and related work are discussed in section 2. The proposed system design and implemented are presented in section 3. In section 4 experiments and their collected result are discussed for the validation. Finally, section 5 summarizes the paper and present the future work.

2. BACKGROUND AND RELATED WORK

This section will address prior research that has been used as the primary source of knowledge for this report and is applicable to the topics of our investigation (cloud, edge, osmotic computing, and blockchain).

2.1 Cloud and Edge Computing

Now, the number of the IoT devices connected to internet exceeds one billion devices, which increases the demand on both required storage and computing. Edge computing refers to computing executed at the location nearby the generated data or its end user which the data source is located. Edge architecture offers computing to take place more quickly by reducing latency and response delay. Software programs running at the edge can perform faster and efficiently, resulting in a better performance and improved general user experience. Cloud computing is a computational model that incorporates shared computing resources instead of local servers or personal devices to run programs. The terminology is described as the large data centers available to multiple users' consumption over the Internet (public cloud) or on a private network (private cloud). Extending edge computing to with cloud computing resources offers overcoming the limitations of both IoT devices and edge computing resources' capabilities with the cost of increasing the latency and delay.

According to [5,] implementing a function as a service platform implementation to deploy functionality on fog nodes with restricted capabilities meets the needs of various application scenarios and diverse fog node deployments, but the critical overhead is an unaddressed concern. While adopting application latency requirements in edge planning through an artificial intelligent approach is discussed in [6] where a new design is presented to improve the effectiveness of smart cities services through minimizing the response and optimizing the energy consumption, however it focused on one scenario which cloud not be guaranteed to maintain with different applications. [7] Edge computing edge management for shared smart city services while retaining privacy, in which only a simulation is presented with the absence of real implemented use case. An evaluation of open source serverless computing frameworks support at the edge [8],

evaluating open source serverless platforms, Kubeless, namely, Apache OpenWhisk, Knative, and OpenFaaS, but no technical contribution is presented.

2.2 Osmotic Computing

Osmotic Computing [3] is a promising new computation architecture for integrating cloud computing resources with edge computing and IoT infrastructure, creating a new ecosystem where resources is shared between cloud, edge, and IoT. Osmotic computing main concert is to apply the balance between resource utilization and load balancing through management, orchestration, configuration, and deployment of the shared resources (see Figure 1).

The fundamental component is microelement (MEL) which is an abstraction for both functionality (microservice) and stored data (microdata). MELs are managed through an osmotic manager to facilitate the moving across layers. Basically, osmotic computing focuses is to allow to add/remove/migrate MELs between all three layers to adopt the performance of the network and utilize its resources. Moreover, MELs are virtualized resources which are running as virtual containers to eliminate the dependency on the hosted environment, allowing the deployment of different infrastructure resources.

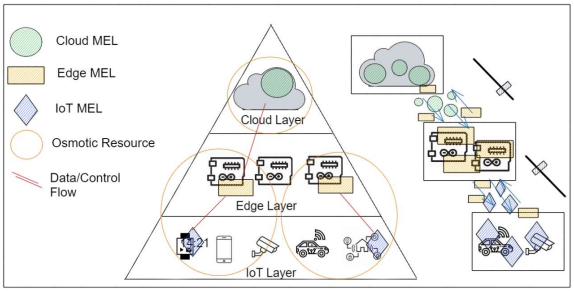

Figure 1: Osmotic Computing

Software defined membrane (SDMem) is another vital concept of osmotic computing, which is an abstraction to represent the virtualized environment based on the assigned infrastructure (e.g., cloud and edge infrastructures). MEL can be migrated through its assigned membrane without the need to contact any external entity.

Osmotic computing provides dynamic orchestration of deployed MELs whether through IoT devices, edge, or cloud infrastructures, ensuring the satisfactory quality of service (QoS) and detecting potential faults or capacity limitations.

In [9], a proposed a five layers adaptable architecture that adds security measurement across different levels of osmotic computing paradigm addressing the need for separately securing edge computation, edge storage and emerging edge configurations as the computing resources increases, but the authors didn't present any implementation details for the proposed design. The authors of [10] introduced an adaptable model based on the standards of lock door and intermediate state management that allows for trust virtualization by effectively differentiates between trusted and untrusted parties, but the proposed model is limited to the pervasive online social networks. The authors describe the concept of Software Defined Membrane in [11] where SDMem is designed depending on a security policy where system administrator defines types of the potential attached and its expected system damage in which the policy driven concept is discussed without any design specifications. CATMOSIS is demonstrated in [12] which suggests the integration of security features based on catalytic and osmotic computing over the 5G networks where the suggestion is discussed from the evaluation survey assessment.

2.3 Blockchain and Smart Contracts

Blockchain is a security technology that allows for the creation of a distributed ledger that logs any changes or transactions that take place inside a network. Instead of a solitary server, this ledger repository is decentralized and housed on every component that is a part of the blockchain.

A blockchain network's smart contract and ledger are two of the most important software modules for regulating access to the distributed ledger, automating operations, controlling elements of transactions, and executing particular actions if certain circumstances are satisfied. As executable code, these contracts take occur within a blockchain. The condensed algorithm accepts or rejects changes or transactions that occur within the network based on agreement among all network members. A transaction is then logged in the ledger once it has been validated. Data or transactions that happen on the network and are uploaded to the ledger are encrypted, kept in a block, and subsequently become immutable and un-editable. The two main categories of blockchains are:

1. Permissionless blockchain: a public blockchains where any method can add a block to the blockchain, but only if a cryptographic equation is solved which considered a proof of work (POW). A wide range of applications implementing permissionless blockchain including Bitcoin, Zerocash, and Ethereum.
2. Permissioned blockchain: a blockchain where the decision of adding a block is the responsibility of

a central organization called certificate authority or CA after running the consensus algorithm and collecting the decisions. Permissioned blockchains could be implemented as public or private blockchains. Examples for permissioned blockchain including Hyperledger Fabric (HLF), Quorum, and Corda.

Integrating blockchain with IoT has been studied in [13] introducing the prospects and challenges where an overview of blockchain technology with emphasis on the IoT perspective. [14] has developed a blockchain supported framework for IoT applications, where a blockchain based methodology is presented to achieve manageability and security for the IoT applications. Applying blockchain and smart contracts are used to clinical data maintaining security and providing an implementation for decision support for patients depending on blockchain while achieving privacy, traceability, and security are achieved, was discussed in [15]. In [16], a utilization for SLA (service level agreements) optimizing the available resources and automatic deployments, aiming to achieve a full dynamic and decentralized architecture. The triple Diffie-Hellman protocol is extended depending on blockchain for eliminating single point of failure based on smart contracts for edge and cloud platforms is discussed proposing a protocol, was proposed in [17].

3. ARCHITECTURE DESIGN

In this paper, we design an osmotic computing model based on the permissible Hyperledger blockchain for providing the security and privacy of end-to-end IoT solutions, as shown in Figure 2. The designed model's main requirements are (1) *managed nodes security*: the identity of all managed nodes is maintained by the registry, and all nodes are monitored to prevent unauthorized attacks; (2) *network quality*: the identity of all MELs is maintained by registry, and each node and MEL update is easily tracked, and its source maintained; (3) *transparency*: Network members have complete visibility into the status of MELs; and (4) data access: Any user data stored on- or off-chain is governed by user consent.

The basic procedure is to create a peer certificate on an edge or IoT managed node and register it with the cloud managed node to which it is linked. The authority for the managed node's transaction is then checked in the cloud using the blockchain registered public key, and the initial ledger is distributed. It also asks the ordering service to send the updated ledger to all the peers. When records are

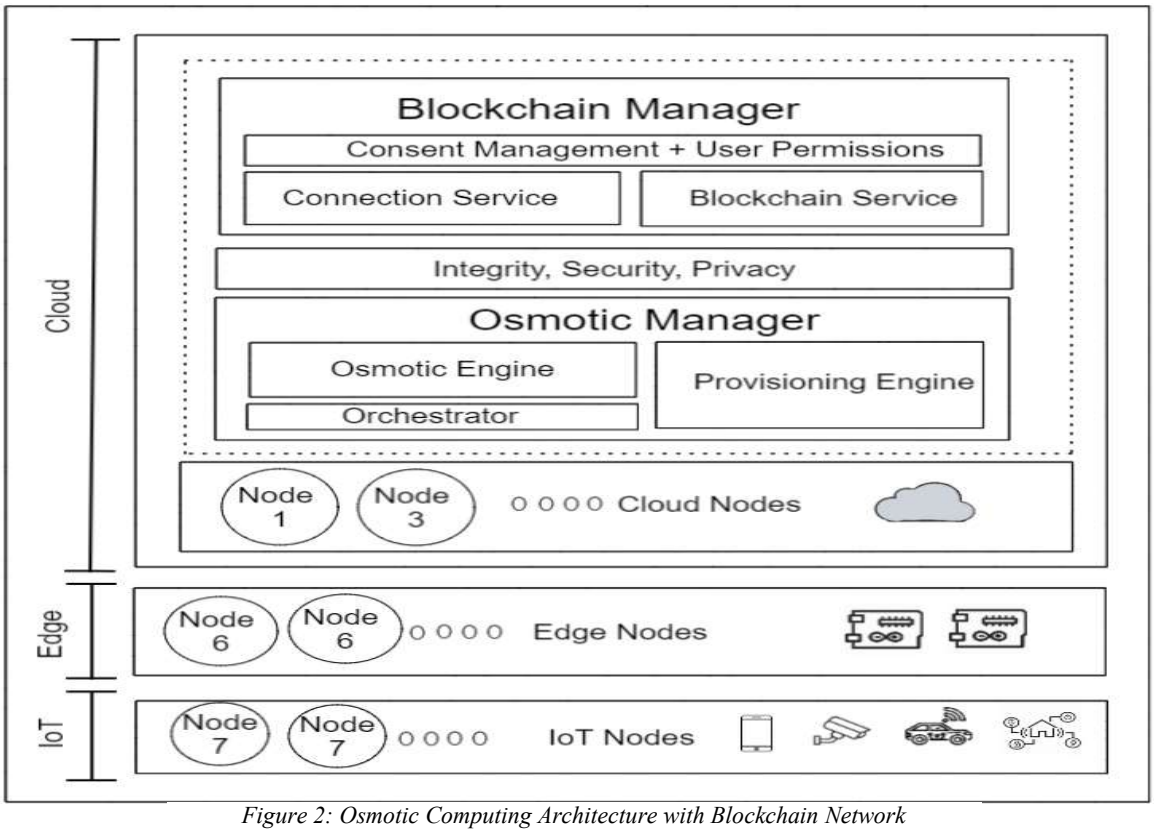

Figure 2: Osmotic Computing Architecture with Blockchain Network

processed, they are added to the ledger, and peers in the same channel can request transactions like querying and exchanging data from the newly registered edge device.

All managed nodes are registered to blockchain network as peer nodes as shown in Figure 3, while the ordering service consists of the cloud nodes where network connection is sustained. The ledger is distributed among all peer nodes. Whenever the ordering service updates the ledger, a fresh copy is redistributed again. The orderer peers are equipped with Raft protocol to be able to publish to records, store records and process records. Moreover, osmotic SDmem is represented as blockchain channel which is a private blockchain subnet for communication between two or more specific networks managed nodes, for the purpose of conducting private and confidential transactions.

The osmotic manager represents a blockchain certificate authority (CA), which is a trusted entity that issues secure sockets layer (SSL) certificates and is in charge of maintaining the membership service provider's integrity. It manages member registration, certification, and node regulation. As a result, certificate authorities play a crucial role in establishing a permissioned blockchain, registering members, tracking identities within the network, and removing deprecated accounts. The main workflow for MELs transaction with the blockchain will be as following:

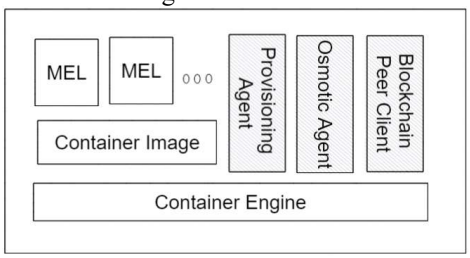

Figure 3: Blockchain Enabled Managed Node Design

1. Creation of transaction request: MELs transactions including creating, stopping, or migrating new/existing MELs. The peer node (cloud, edge, or IoT resource node) creates a request within a peer client installed on the node creating a request through its embedded SDK of a selected programming language.
2. Transaction Endorsement: After submitting a transaction, it must be validated by the specified channel endorsement policy. Endorsement policy specifies a set of peers on the same channel must execute chaincode for a transaction to be considered valid.
3. Submission to orderer peers: Ordering service is a set of orderer nodes (osmotic cloud nodes) that is responsible for executing the consensus algorithm, generating the endorsed transaction block. the ledger block is read only for both block content and blocks sequence to provide immutability.
4. Commitment of transaction: After orderer service generates the transaction block, the block is being broadcasted to all the peers on the channel for ratifying as a final validation action.
5. Submission to ledger: Once the peers receive and ratify the transaction block, the block is written into the ledger.

In essence, the massive amount of data generated from the MELs osmotic operations due to dynamic orchestration could be easily tempered through the exposed network configurations stored within each node, blockchain protects the integrity of these data through keep the reconfiguration audit trail stored at the blockchain ledger to maintain the osmotic network participants protected against malicious injected MELs to offer authorized access control and establish the root of trust for applying the main features of traceability, immutability, transparency, and auditability for the osmotic network. Additionally, the access control through the osmotic manager which ensures the security and creditability of the connected MELs by signing all the participant MELs with the generated TLS SSL to keep sensitive data secure, verify ownership of the MELs, prevent attackers from injecting fake MELs, and convey trust to participant entities ensuing privacy, security, and confidentiality.

4. IMPLEMENTATION

Hyperledger Fabric is a permissioned framework implementation of blockchain technology and is a part of the Hyperledger blockchain platform. The framework provides the base for developing blockchain-based software, communication systems, and other projects. Fabric was created with the purpose of allowing users to create private blockchains that can be used within a single organization or a group of related organizations and connected to other blockchain implementations. It provides a modular architecture, allowing pluggable implementations of the various function. Fabric implements many key features as part of its architecture, including privacy, confidentiality, and integrity. The Hyperledger Fabric blockchain can be installed by downloading the latest release of the open-source framework from its public repository or by using one of the Fabric Operator packages, which encloses all complexity in simple commands. The osmotic orchestrator is implemented via Kubernetes, which is an open-source system for automating deployment, scaling, and maintaining containerized applications. Kubernetes provisions

Journal of Theoretical and Applied Information Technology
15th July 2022. Vol.100. No 13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

more advanced configurations and features to set up a cluster compared to Docker, which make it a very strong candidate for any production-scale environment.

The osmotic manager is responsible for orchestrating MEls over all the managed nodes across the 3 layers with the Hyperledger Fabric network, to enable the blockchain functionality within the osmotic architecture. All the MELs operations within the osmotic flow (e.g., add, stop, and migrate MEL/s) are considered blockchain transactions which need to be validated, verified, and approved by the blockchain network.

The orchestrator manager node is defined as a part of the fabric environment setup as a certificate authority (CA) to enable the permissioned formation of a blockchain. Snippet [1] shows the container specifications.

```
ca-tls:
  container_name: ca-tls
  image: hyperledger/fabric-ca
  command: sh -c 'fabric-ca-server start -d -b tls-ca-admin:tls-ca-adminpw --port 7052'
  environment:
    - FABRIC_CA_SERVER_HOME=/tmp/hyperledger/fabric-ca/crypto
    - FABRIC_CA_SERVER_TLS_ENABLED=true
    - FABRIC_CA_SERVER_CSR_CN=ca-tls
    - FABRIC_CA_SERVER_CSR_HOSTS=0.0.0.0
    - FABRIC_CA_SERVER_DEBUG=true
  volumes:
    - /tmp/hyperledger/tls/ca:/tmp/hyperledger/fabric-ca
  networks:
    - fabric-ca
  ports:
    - 7052:7052
```

Snippet 1: Fabric TLS CA container YAML

The managed node will have one of the following blockchain container apps running to participate actively withing the fabric network:

1. Peer client: which is responsible for: (a) manage the enrollment process, (b) execute transaction requests, (c) update local ledger, and (d) join/renew channel. A peer should be enrolled in the CA creating its certificate (see Snippet [2]). The enrollment process generates enroll id and a secret key to be used when lunching the peer. This process preserves both the integrity of the CA and private keys as the information is being shared only with the osmotic manager which acts as CA and runs the commands.

```
export FABRIC_CA_CLIENT_MSPDIR=tls-msp
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org1/peer1/assets/tls-ca/tls-ca-cert.pem
fabric-ca-client enroll -d -u https://peer1-org1:peer1PW@0.0.0.0:7052 --enrollment.profile tls --csr.hosts peer1-org1
```

Snippet 2: Peer Enrollment

After the peer is enrolled, the peer is ready to be lunched. This will be run through serving the peer docker using the generated enroll id and secret generated from the previous step (see Snippet [3]).

```
peer1-org1:
  container_name: peer1-org1
  image: hyperledger/fabric-peer
  environment:
    - CORE_PEER_ID=peer1-org1
    - CORE_PEER_ADDRESS=peer1-org1:7051
    - CORE_PEER_LOCALMSPID=org1MSP
    - CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org1/peer1/msp
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=guide_fabric-ca
    - FABRIC_LOGGING_SPEC=debug
    - CORE_PEER_TLS_ENABLED=true
    - CORE_PEER_TLS_CERT_FILE=/tmp/hyperledger/org1/peer1/tls-msp/signcerts/cert.pem
    - CORE_PEER_TLS_KEY_FILE=/tmp/hyperledger/org1/peer1/tls-msp/keystore/key.pem
    - CORE_PEER_TLS_ROOTCERT_FILE=/tmp/hyperledger/org1/peer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
    - CORE_PEER_GOSSIP_USELEADERELECTION=true
    - CORE_PEER_GOSSIP_ORGLEADER=false
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer1-org1:7051
    - CORE_PEER_GOSSIP_SKIPHANDSHAKE=true
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric/org1/peer1
  volumes:
    - /var/run:/host/var/run
    - /tmp/hyperledger/org1/peer1:/tmp/hyperledger/org1/peer1
  networks:
    - fabric-ca
```

Snippet 3: Peer Node container YAML

2. Orderer client: which is responsible for (a) manage the enrollment process, (b) execute endorsement policies, (c) packaging transactions into blocks, and (d) distribute them to anchor peers across the network. The process of setup a new orderer is similar to creating new peer, with a major difference of the used container image, but first a genesis block needed to be created as is mandatory for orderer to bootstrap itself which is accomplished by executing the running configtxgen command (see Snippet [4]).

   Now, lunching the orderer is ready through serving its container (see Snippet 5), the container is configured with enrollment id and secret in addition to the genesis block reference.

```
configtxgen -profile OrgsOrdererGenesis -outputBlock
/tmp/hyperledger/org0/orderer/genesis.block -channelID syschannel
configtxgen -profile OrgsChannel -outputCreateChannelTx
/tmp/hyperledger/org0/orderer/channel.tx -channelID mychannel
```

Snippet 4: Create Genesis Block

The mapping of osmotic SDMem defining the virtual network in which a MEL could be migrated on the Fabric is accomplished through creating a blockchain channel. A node should be used to create a channel for the CLI which executes the command. Peer1 host machine will be used (see Snippet 6).

Journal of Theoretical and Applied Information Technology
15th July 2022. Vol.100. No 13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
orderer1-org1:
  container_name: orderer1-org1
  image: hyperledger/fabric-orderer
  environment:
    - ORDERER_HOME=/tmp/hyperledger/orderer
    - ORDERER_HOST=orderer1-org1
    - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
    - ORDERER_GENERAL_GENESISMETHOD=file
    - ORDERER_GENERAL_GENESISFILE=/tmp/hyperledger/org1/orderer/genesis.block
    - ORDERER_GENERAL_LOCALMSPID=org1MSP
    - ORDERER_GENERAL_LOCALMSPDIR=/tmp/hyperledger/org1/orderer/msp
    - ORDERER_GENERAL_TLS_ENABLED=true
    - ORDERER_GENERAL_TLS_CERTIFICATE=/tmp/hyperledger/org1/orderer/tls-msp/sign-
certs/cert.pem
    - ORDERER_GENERAL_TLS_PRIVATEKEY=/tmp/hyperledger/org1/orderer/tls-msp/key-
store/key.pem
    - ORDERER_GENERAL_TLS_ROOTCAS=[/tmp/hyperledger/org1/orderer/tls-msp/tlscacerts/tls-0-
0-0-0-7052.pem]
    - ORDERER_GENERAL_LOGLEVEL=debug
    - ORDERER_DEBUG_BROADCASTTRACEDIR=data/logs
  volumes:
    - /tmp/hyperledger/org1/orderer:/tmp/hyperledger/org1/orderer/
  networks:
    - fabric-ca
```

Snippet 5: Orderer Node container YAML

Chaincode provides the structure of transactions on a channel, a chaincode needs to be installed on all the peers joined to the channel that want use the chaincode to update or query the channel ledger. Then, one member of the channel can then instantiate the chaincode on a channel and set the chaincode's endorsement policy. The ledger contains MELs' metadata such as MEL Id, MEL Application Function, Peer Name, and Organization Name. Multiple chaincodes are implemented for the descripting the MEL status and data protection (see Snippet 7). Chaincode could be implemented in Java, Go, Tor Node.js. Peer node clients have a peer state database are equipped with CouchDB to support core chaincode operations such as getting and setting a key (asset), and querying based on keys.

```
export CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org1/admin/msp
peer channel create -c mychannel -f /tmp/hyperledger/org1/peer1/assets/channel.tx -o
orderer1-org0:7050 --outputBlock /tmp/hyperledger/org1/peer1/assets/mychannel.block --tls
--cafile /tmp/hyperledger/org1/peer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
```

Snippet 6: Create New Channel

All commands, including docker deployment and Hyperledger Fabric operations are executed through osmotic manager for the full automation of the osmotic orchestration. Kubernetes using the kubectl CLI (command-line interface) or any command-line tools which in turn use the API, (e.g., kubeadm) to achieve dynamic osmotic computing platform.

```
public async createMel(ctx: Context, melId: string, melAppFunction: string, peerName: string,
orgName: string, instanceId: string)
{
    console.info('============ START : Create MEL ==========');
    const mel: MEL = {
        melId,
            docType: 'mel',
            melAppFunction,
            peerName,
            orgName,
            instanceId
    };
    await ctx.stub.putState(melId, Buffer.from(JSON.stringify(mel)));
    console.info('============ END : Create MEL ==========');
}
```

Snippet 7: Chaincode Sample

5. EXPERIMENTS AND RESULTS

Experiments are conducted to measure the blockchain performance such as transaction average response, resource utilization, and throughput. For this purpose, Hyperledger Caliper is an open-source blockchain performance analysis tool. It allows users to assess a blockchain implementation's performance using a set of defined scenarios. Moreover, the overhead introduced by the blockchain is measured by considering the following implementations: (1) Osmotic Manager (OM): where no blockchain is installed, only osmotic end-to-end dynamic orchestration; (2) Osmotic Manager Blockchain Enabled (OMBC): where the proposed design is implemented using Hyperledger Fabric v2.3. The implementation of the osmotic manager has a configuration setting to enable or disable the blockchain integration.

Table 1 shows the specifications of infrastructure used to setup test environment. Cloud nodes are EC2 instances host on AWS. Moreover, osmotic manager is t2.2xlarge EC2 instance hosted on AWS. Edge nodes are Raspberry Pi 4 model B with Broadcom model BCM2711, 4 GB LPDDR4 RAM, Quad core Cortex-A72 (ARM v8) 64-bit 1.5GHz processor, and 5.0 GHz IEEE 802.11ac wireless. Edge nodes and IoT devices are setup at the same laboratory connected to Wi-Fi router with 20 Mbps. For the IoT devices a simple mobile application is implemented and installed on iPhone 12 Pro Max device connected to the same Edge Wi-Fi router.

Journal of Theoretical and Applied Information Technology
15th July 2022. Vol.100. No 13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1: Test Environment Infrastructure Specifications

Feature	Osmotic Manager	Edge Nodes	Cloud Nodes
Number	1	4	2
OS	Ubuntu 14.04.5 LTS	Raspberry Pi OS	Ubuntu 14.04.5 LTS
CPU/s	8	4	2
CPU MHz	up to 4.0 GHz	1.5 GHz	up to 4.0 GHz
Memory Size	32 GB	4 GB	16 GB
Disk	160 GB	16 GB	80 GB

A simple client application was deployed to be installed on IoT device to create the load. The application is divided into 3 MELs: (1) monitor: constantly sends its GPS location and speed, (2) update: checks for car software updates, and (3) break: sends update whenever car speed is deaccelerating. All 3 MELs are updating a car status with speed and location, this application is developed on as a prototype to test the osmotic computing enhancements presented at this work and doesn't represent a real-life use case.

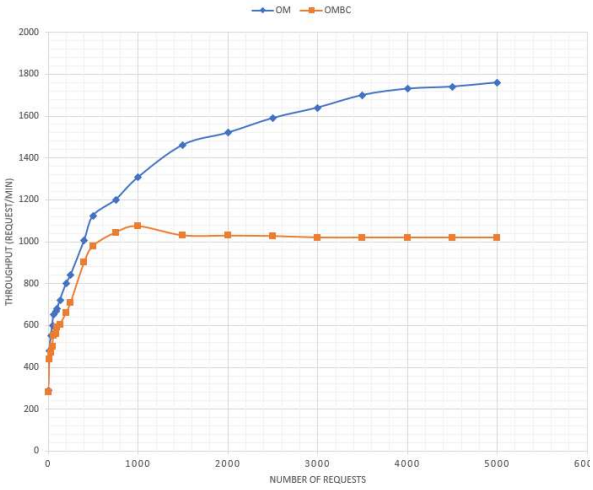

Figure 4: Throughput Experiment on OM and OMBC

The throughput experiment was conduct by performing batch requests to osmotic manager while varying the load size (different MEL requests) with 2 different scenarios: (1) OM: blockchain security configuration is disabled (2) OMBC: blockchain is enabled. As shown in Figure 4, the throughput for OM is exponentially growing with increase of the number of transactions reaching its peak at 1800 transaction per minute.

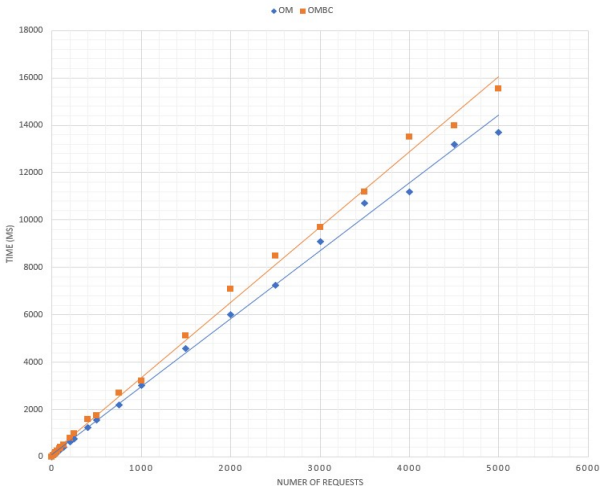

Figure 5: Response Time Experiment on OM and OMBC

Additionally, it is noticed that the throughput still increasing as a response to the number of requests indicating because of dynamic orchestration made by the osmotic manager trying to adopt to the rising load. On the other hand, the OMBC peaked at approximately 1100 transaction per minute and started to decrease due to the number of MELs operations occurred the blockchain ledger size increased resulting more time for a request to be processed as it needs to be endorsed by the orderer node.

The average response time is measured through repeating the same scenario of the throughput experiment. Figure 5 illustrates that basically there are no gaps between running the requests on OM or OMBC as the IoT device is completely isolated from the blockchain operations and both are affected with MELs operations even though it consumes more time on the OMBC. Although OM still has lower average response time which is expected as the performance of the managed nodes is affected by the overhead of the running blockchain containers.

6. CONCLUSIONS AND FUTURE WORK

This paper presented osmotic orchestrator, a new design to challenge the security, privacy, and integrity issues of osmotic computing through emerging blockchain technology with osmotic computing model to overcome the distributed architecture security vulnerabilities. The architecture integrated Hyperledger Fabric Blockchain platform defining the orchestration engine as a certificate authority for all managed nodes offering a secured private network of

communication among managed nodes. Additionally, all the MELs transactions between the managed nodes, performed as osmotic operations, are stored at the blockchain ledger for osmotic activities to provide traceability and data validation preventing any fake MEL injection into the osmotic network.

An implementation of the Hyperledger Fabric Blockchain platform as a permissioned blockchain technology integrated with osmotic ecosystem to test and analyze the performance of the proposed osmotic computing blockchain enabled model. The experiments show that the throughput is affected with the increasing load, which is resulted from large blockchain ledger size. However, MELS average response time is slightly impacted as a trivial result for computation power consumed by the blockchain agent.

Future work will concentrate on the new introduced concept on multi-cloud application and how to be utilized with the osmotic environments from both orchestration and security domains. In scenario of a shared application between multiple organizations using different cloud providers, the osmotic orchestrator architecture shall be adopted to offer dynamic secured deployments.

Furthermore, the design of artificial intelligence-based consensus algorithm to leverage the blockchain technology, where performance is the key challenge in IoT applications.

REFERENCES

[1] Industrial AI and AIoT Market Report 2021–2026. IoT Analytics. Published November 24, 2021. Accessed November 27, 2021. https://iot-analytics.com/product/industrial-ai-aiot-market-report-2021-2026/

[2] Rekha S, Thirupathi L, Renikunta S, Gangula R. Study of security issues and solutions in Internet of Things (IoT). Materials Today: Proceedings. Published online July 2021. doi: 10.1016/j.matpr.2021.07.295

[3] DAN-CRISTIAN C. Block-Cloud: The new paradigm of Cloud Computing. ECONOMY INFORMATICS JOURNAL. 2019;19(1/2019):14-22. doi: 10.12948/ei2019.01.02

[4] Zeadally S, Abdo JB. Blockchain: Trends and Future Opportunities. Internet Technology Letters. Published online September 30, 2019:e130. doi: 10.1002/itl2.130

[5] Baresi L, Quattrocchi G. PAPS: A Serverless Platform for Edge Computing Infrastructures. Frontiers in Sustainable Cities. 2021;3. doi: 10.3389/frsc.2021.690660

[6] Aral A, Brandic I, Uriarte RB, De Nicola R, Scoca V. Addressing Application Latency Requirements through Edge Scheduling. Journal of Grid Computing. 2019;17(4):677-698. doi: 10.1007/s10723-019-09493-z

[7] Xu X, Huang Q, Yin X, Abbasi M, Khosravi MR, Qi L. Intelligent Offloading for Collaborative Smart City Services in Edge Computing. IEEE Internet of Things Journal. 2020;7(9):7919-7927. doi: 10.1109/jiot.2020.3000871

[8] Kjorveziroski V, Bernad Canto C, Juan Roig P, et al. IoT Serverless Computing at the Edge: Open Issues and Research Direction. Transactions on Networks and Communications. 2021;9(4):1-33. doi: 10.14738/tnc.94.11231

[9] Mlotshwa LL, Makura SM, Karie NM, Kebande VR. Opportunistic security architecture for osmotic computing paradigm in dynamic IoT-Edge's resource diffusion. Proceedings of the 2nd International Conference on Intelligent and Innovative Computing Applications. Published online September 18, 2020. doi: 10.1145/3415088.3415097

[10] Sharma V, You I, Kumar R, Kim P. Computational Offloading for Efficient Trust Management in Pervasive Online Social Networks Using Osmotic Computing. IEEE Access.2017;5:5084-5103. doi:10.1109/access.2017.2683159

[11] Villari M, Fazio M, Dustdar S, Rana O, Chen L, Ranjan R. Software Defined Membrane: Policy-Driven Edge and Internet of Things Security. IEEE Cloud Computing. 2017;4(4):92-99. doi:10.1109/mcc.2017.3791014

[12] Choudhary G, Sharma V. A Survey on the Security and the Evolution of Osmotic and Catalytic Computing for 5G Networks. 5G Enabled Secure Wireless Networks. Published online 2019:69-102. doi: 10.1007/978-3-030-03508-2_3

[13] Uddin MdAshraf, Stranieri A, Gondal I, Balasubramanian V. A Survey on the Adoption of Blockchain in IoT: Challenges and Solutions. Blockchain: Research and Applications. Published online February 2021:100006. doi: 10.1016/j.bcra.2021.100006

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[14] Sodhro AH, Pirbhulal S, Muzammal M, Zongwei L. Towards Blockchain-Enabled Security Technique for Industrial Internet of Things Based Decentralized Applications. Journal of Grid Computing. Published online August 7, 2020. doi: 10.1007/s10723-020-09527-x

[15] Zhang P, White J, Schmidt DC, Lenz G, Rosenbloom ST. FHIRChain: Applying Blockchain to Securely and Scalably Share Clinical Data. Computational and Structural Biotechnology Journal. 2018; 16:267-278. doi: 10.1016/j.csbj.2018.07.004

[16] Kochovski P, Stankovski V, Gec S, et al. Smart Contracts for Service-Level Agreements in Edge-to-Cloud Computing. Journal of Grid Computing. Published online October 13, 2020. doi: 10.1007/s10723-020-09534-y

[17] Ruggeri A, Celesti A, Fazio M, Galletta A, Villari M. BCB-X3DH: a Blockchain Based Improved Version of the Extended Triple Diffie-Hellman Protocol. 2020 Second IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA). Published online October 2020. doi: 10.1109/tps-isa50397.2020.00020