IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 13 June 2022, accepted 9 September 2022, date of publication 20 September 2022, date of current version 6 October 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3208167

RESEARCH ARTICLE

A Decentralized Solution for Epidemiological Surveillance in Campus Scenarios

ANDREA FORNAIA, GIOVANNI MAROTTA, GIUSEPPE PAPPALARDO, AND EMILIANO TRAMONTANA
Dipartimento di Matematica e Informatica, University of Catania, 95125 Catania, Italy

Corresponding author: Giovanni Marotta (giovanni.marotta@phd.unict.it)

This work was supported by the Project TEAMS—TEchniques to Support the Analysis of Big Data in Medicine, Energy and Structures—Piano di incentivi per la ricerca di Ateneo 2020/2022.

ABSTRACT Throughout the various containment phases of a pandemic, such as Covid-19, digital tools and services have proven to be essential measures to counteract the ensuing disrupting effects in social and working interactions. In such scenarios, Nausica@DApp, the comprehensive solution proposed in this paper, eases compatibility of the in-presence activities of a campus-based corporation with the organizational constraints posed by the virus during the pandemic, or at a later endemic stage. This is accomplished throughout several intervention areas, such as personnel contact tracing, crowd gathering surveillance, and epidemiological monitoring. These operational requirements, in particular indirect contact tracing and overcrowd monitoring, call for the adoption of an absolute device localization paradigm, which, in the proposed solution, has been devised on top of the campus WiFi infrastructure, proving to be encouragingly accurate in most cases. Absolute localization, on the other hand, entails a certain amount of server-based centralized operations, which might affect the preservation of user data privacy. The novelty of the proposed solution consists in maximizing confidentiality and integrity in the handling of sensitive personal information, in spite of the centralized aspects of the localization system. This is accomplished by decentralizing contact tracing matching operations, which are entirely carried out locally, by apps running on the users' mobile devices. Contact data are pseudonymized and their authenticity is guaranteed by a blockchain. Furthermore, the proposed novel solution improves privacy preservation by eschewing recourse to the Bluetooth app-to-app channel for user data exchange, in fact a typical choice of most current contract tracing solutions. Thanks to a sensible use of the blockchain features, integrated into Nausica@DApp's microservice-based back-end, a higher degree of operation transparency can be relied upon, thus boosting the user's level of trust and enhancing the availability and reliability of data about people gathering within the campus premises. Moreover, contact tracing only requires the mobile device WiFi interface to be on, so that users are neither forced to adopt new habits, nor to grant additional device access permissions to contact tracing apps (potentially undermining their own privacy). The overall system has been analysed in terms of performance and costs, and the experiments have shown that its adoption is viable and effective.

INDEX TERMS Blockchain, contact tracing, data analysis, decentralized apps, distributed systems, epidemiological surveillance, localization, mobile applications, smart contracts.

I. INTRODUCTION

From the very beginning of the Covid-19 pandemic manifestations, the healthcare, industrial and academic communities have converged toward a shared understanding that pinpoints the development of ad-hoc mobile apps in personal user devices as a means to provide an efficient and reliable response to contact tracing and epidemiological surveillance.

Nowadays, even though several vaccines have entered the scene, we are also aware that the virus will circulate at

The associate editor coordinating the review of this manuscript and approving it for publication was Shajulin Benedict.

alternate endemic levels until the herd immunity will be achieved, if ever. In the meantime, it is essential to maintain well-established practices, such as compliance to social distancing rules, avoidance of overcrowding, preventive swab campaign, and forward and bidirectional contact tracing.

Among these best practices, digital tools and services for risk control should still play a central role in the overall game, in spite of mixed results in the level of acceptance among users so far. In fact, from the early days of the pandemic, national contact-tracing apps have instilled some privacy preserving concerns, especially if de-anonymized and linkable information are available to central authorities for contagion risk follow-ups [1], [2], [3].

It is well-known that a centralized solution can provide more effective responses to epidemiological surveillance issues, especially when coupled with absolute user localization. Unfortunately, such solutions, culturally accepted only in specific contexts, are more prone to well-known privacy concerns [4], [5].

For instance, South Korea's *Co100* relies on GPS or cellular network localization data with the aim of providing a publicly available website (https://coronamap.site), which tracks infected people trajectories. Though initially encrypted and anonymized, this information can be unveiled and provided to health authorities for contact tracing. Similarly, Singapore's *TraceTogether*, which implements relative user localization and a very sophisticated encryption protocol for data exchange [6], can enable the central server, owned by the Ministry of Health, to de-anonymize user identities.

In addition to individual privacy concerns, centralized solutions do not encourage users to trust the veracity of the publicly available aggregated data, particularly about the effects of the pandemic spread, and nowadays of the vaccine campaign, on the number of infected people and deaths. Consequently, such solutions need to implement transparency measures to guarantee and demonstrate that sensitive personal information and aggregated data are collected and used without counterfeiting them [7], [8].

Similar privacy and trust concerns can also be encountered in private environments where digital applications for pandemic monitoring are deployed. Known solutions in the literature do not tackle such issues. This is the case with *WiFiTrace* [9], which monitors personnel flow in a campus environment, based on a server-centric architecture for data exchange, storage and processing. Similarly, *vContact* (https://repository.hkust.edu.hk/ir/Record/1783.1-113518) uses WiFi to recognise smartphones and a mobile app to store location data and check whether a location later marked as infected had been visited.

A significant shift in user perception can be brought about only by promoting a shared understanding of collected data treatment. The blockchain technology [10], with its transparency, based on mutually shared trustworthiness, can open new frontiers in this territory [11]. The World Health Organization, for instance, has been involved with major technology companies and governments in the design and deployment of *MiPasa* [12], a worldwide control and communication platform fuelled by the blockchain technology, which has been employed to enable individuals, state authorities and health institutions to gather, share and correlate data to determine early detection of Covid-19 carriers and infection hot-spots.

Furthermore, a significant number of blockchain-based contact tracing applications have been proposed as an alternative to centralised solutions, as described in more detail in Section VII. This alternative and viable approach can help solution designers to heighten the trust level that users would acknowledge to the application they are supposed to download and activate in their personal devices [13]. In such a way, a higher degree of adoption rate of contact tracing apps should be consequently achieved, thus allowing the solution to be fully effective.

The solution proposed in this paper, named Nausica@-DApp, provides significant and viable answers to the challenges posed by an epidemiological surveillance scenario on campus premises. Its original design [14] is meant to trace personal contacts and handle overcrowding in university activities according to the following key choices.

- A hybrid decentralized approach in the implemented contact tracing algorithm, so as to prevent the typical privacy preserving issues related to centralized solutions in this application arena.
- An ad-hoc, tailored approach to user localization through WiFi sensing of the campus infrastructure. Such a feature enables the presence of Covid-19 positive users to be detected and localizes them in time and space, for the purpose of both direct and indirect contact tracing.
- The integration of Nausica@DApp's back-end with specifically designed blockchain components. This allows extra decentralization, trustworthiness and transparency features to be added to the handling of the user information collected by the system and distributed among involved actors; such additional features include: (i) absolute localization; (ii) contact tracing data; (iii) aggregated data for epidemiological follow-ups.
  The blockchain features are cleverly chosen in order not to compromise the overall system performance, scalability and affordability.

The remainder of this paper is organized as follows. Section II introduces the background on blockchain technology. Section III explains the system design choices and solution features. Section IV presents the back-end design. Section V describes in some detail the implemented software, focusing on blockchain-related aspects. Section VI shows the experimental results obtained putting to work the solution prototype. Relevant related work is outlined and compared in Section VII, before drawing the final conclusions in Section VIII.

II. BACKGROUND ON BLOCKCHAINS

This section presents the core features of blockchain and smart contracts technology and pinpoints its main limitations.

A. BLOCKCHAIN FEATURES

A blockchain is technically a peer-to-peer distributed ledger that registers cryptographically signed transactions in a sequence of linked blocks, namely the chain, in an append-only fashion. Each of the chained blocks stores data—such as a timestamp, a nonce or a set of transactions—and a cryptographic reference to the previous block's hash. Since every hash is unique, every block is linked to a unique parent all the way up to the ''genesis block'', which is the first one in the chain. Blockchain technology combines different well-known concepts such as digital signatures, cryptographic hashing functions, and decentralized consensus algorithms, which validate all the registered transactions without requiring a central authority. The addition of new data to the chain is in fact a decentralized activity carried out by all peer nodes running the agreed-upon consensus algorithm, whose global convergence ensures the reliability of the blockchain and the correctness of the data saved in the blocks, once each transaction has been validated and included in a new block.

The two most popular blockchain platforms, Bitcoin [15], [16] and Ethereum [17], were designed on the basis of the Proof of Work (PoW) consensus algorithm [18], which is a computationally intensive mechanism. The resulting high energy consumption, while discouraging dishonest nodes to transmit malicious blocks to the network, represents a major drawback that has led to the adoption of alternative consensus algorithms [18], such as Proof of Stake (PoS) and Delegated Proof of Stake (DPoS), Proof of Authority (PoA), and so on.

The properties provided by the hash function render the blockchain practically immutable, thus making this technology suitable to ensure authenticity of the stored data.

Blockchains can be public (or permissionless) and private (or permissioned). In a public blockchain, anyone can view the transactions, write data, or run a validator node. Private blockchains, usually managed by few permissioned nodes, may become more scalable and secure, but at the expense of decentralization, according to the blockchain trilemma, which is better explained in Subsection II-C.

B. SMART CONTRACTS

Ethereum is the first distributed ledger technology that has introduced the possibility to run smart contacts, namely programs of arbitrary length and complexity, whose executable code is saved in the blockchain storage [19].

Each smart contract is identified by a unique storage address, which references an account to which digital coins and storage space are linked. At low level, any smart contract is compiled to bytecode and sequentially executed by a virtual machine, which runs in each node of the network, once it has been invoked by a transaction sent to its address. The language complexity of smart contracts and the virtual machine architecture make the latter a Turing-complete system that could potentially run in infinite loops and never terminate, potentially freezing the entire network. To prevent this, Ethereum has introduced a unit, called *gas*, which measures the amount of computational effort required to execute specific operations on its virtual machine. Each bytecode instruction is associated with a cost, called *gas cost*, measured in *gas units*, which users must reward the validating node with when they ask to execute a smart contract's code. Gas costs of instructions are predetermined by the network and only alterable through a protocol upgrade. When users send a transaction (e.g. to invoke a smart contract), they must set the *gas price*, i.e., the price in *gwei* they are willing to pay for the gas unit cost of their transaction. A gwei (gigawei) is worth 10^{-9} Eth (Eth stands for Ether, Ethereum's native coin). Of course, the transaction takes place when some miner accepts the user's offered price. As a result, the average amount (in gwei) paid for a gas unit fluctuates over time, as shown by trackers such as https://etherscan.io/gastracker. Besides gas price, users set a maximum amount of gas units planned for execution, called *gas limit*. If an execution requires more gas than the gas limit specified by the user, a special out-of-gas exception is thrown and the virtual machine's state is *reverted* to one prior to the execution. In this case, the user will still have to pay the gas to the validating node as a countermeasure against potential Denial of Service attacks [20]. Note that a block gas limit is also enforced by Ethereum's blockchain protocol, which provides an upper bound to the possible amount of data and processing that any transaction can incur during a smart contract execution.

C. BLOCKCHAIN LIMITATIONS

Blockchain technology, in its constantly evolving stages, tries to address several issues. First and foremost, the one known as the ''blockchain trilemma'' initially formulated by Vitalik Buterin, states that, regarding scalability, security and decentralization, any improvement in one of these aspects will negatively impact on at least one of the other two [21]. This means that a fine tuning of these three factors has to be performed when designing a blockchain system solution.

Although the introduction of gas limits the amount of data and computational effort that can be spent to process a single block, a major issue is represented by transaction costs, which can also drastically rise during network congestion. Transaction fees can range from a few fractions to hundreds of dollars and have exhibited an uptrend over time (https://blockchair.com/ethereum/charts/average-transaction-fee-usd).

In the blockchain domain, program bugs trigger some peculiar software development challenges. Due to the immutability feature of blockchains, once a smart contract has been deployed on the blockchain, it cannot be modified (although it can be deleted). Consequently, patching a deployed contract is impossible, and for this reason, a significant number of smart contracts are considered to be vulnerable. In 2016, a symbolic execution analysis tool showed that

45% of 19,366 smart contracts were vulnerable with at least one security issues [22].

III. THE NAUSICA@DAPP SOLUTION

From a system architecture perspective, Nausica@DApp is a mixed decentralized solution, in which contact tracing algorithms are run at the mobile DApp side, while the back-end server supports user absolute localization, real-time monitoring of premises occupation and collected data aggregation.

Nausica@DApp does not leverage the Bluetooth channel for proximity detection, as in most similar solutions, thus eschewing the typical privacy concerns related to such a technology [23], [24]. Rather, Nausica@DApp uses the combined app-to-server and server-to-app channels supported by the campus WiFi infrastructure or the cellular network.

The adopted absolute localization of active devices allows the proposed solution to fulfil satisfactorily the major requirements of our project, which are listed below:

- direct (i.e., synchronous) contact tracing;
- indirect contact tracing (i.e., asynchronous, within the contagiousness time-frame);
- premises occupation counting with overcrowding conditions detection.

A. SYSTEM OVERVIEW

A simplified view of the system architecture is depicted in Figure 1, which shows the dominant information flows between system components.

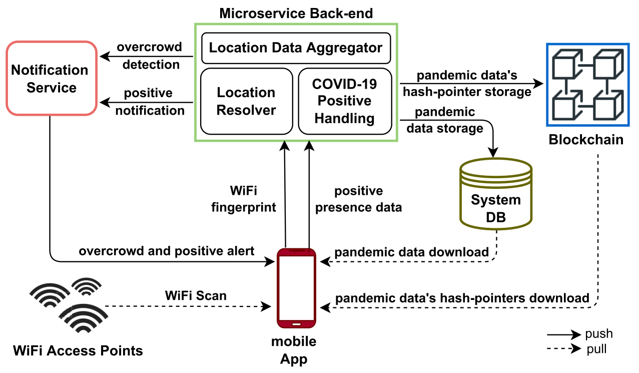


FIGURE 1. Mutual interactions of Nausica@DApp system components.

The administration back-end has been implemented in accordance with a microservice architecture, which is more extensively detailed in Section IV. It interacts with an external *Notification Service*, which provides a scalable solution to send push notifications to the mobile Nausica@DApp-enabled devices. It also interfaces with *System DB* in Figure 1, the adopted database (whether centralized or decentralized) in which pandemic-related data are stored at the disposal of authorized parties. Presence anonymized data are hashed and hashes are used as keys to refer to them in blockchain-related operations. For this reason they will be termed *hash-pointers* hereafter.

The design incorporates ad-hoc blockchain wallet and smart contract capabilities into the system architecture. In particular, these added features enable the back-end server to expose, in a decentralized and unforgeable storage, the hash-pointers to the anonymized proximity data needed by the mobile DApps to perform the internal matching operations for contact tracing. Furthermore, the blockchain components will allow other involved stakeholders, such as the academic community and the health authorities, to retrieve aggregate data about pandemic trends in a transparent and shared consensus-driven environment.

B. DEVICE LOCALIZATION

The proposed solution displays, among other features, indirect contacts tracing and real-time monitoring of premises occupation, which inherently require absolute localization of devices, i.e., positioning them in space within determined time intervals. WiFi sensing has proven to be an appropriate choice to fulfil such a requirement, both when it leverages the WiFi enterprise campus infrastructure already in place and when it relies on external service providers.

Even though scientific literature warns that WiFi indoor localization may be a complex task to accomplish [25], our basic idea is that an Access Point (AP for short) deployed in a closed environment (e.g., a laboratory) will emerge—most of the times—as the one radiating the most powerful signal strength, to such an extent that the device absolute position can be comfortably determined by the system software. Subsection VI-A of this paper will make it experimentally evident that such a choice is compatible with most indoor areas within the campus premises. In order to deal with the unlucky case of garbled WiFi locations or outdoor spots, our solution's localization algorithm resorts to an external WiFi positioning provider (https://en.wikipedia.org/wiki/Wi-Fi_positioning_system) via standard RESTful APIs.

The mobile DApp side features a GPS-driven mechanism which triggers the WiFi fingerprinting collection as soon as the device enters the monitored premises (this could be required for compliance with privacy and/or campus admission rules).

Each active instance of the DApp, in the initialization phase, first registers with the central administration server; in response, it gets back a registration user-ID that uniquely and anonymously identifies it.

After that, as shown in Figure 2, the mobile DApp initiates device localization operations by scanning the surrounding WiFi signals from APs and measuring signal intensity or RSSI (Received Signal Strenght Indicator) (step 1). Then, the DApp sends the *Location Resolver* service the collected *WiFi fingerprint* data, made up of RSSIs measured in four consecutive scans (step 2).

The *Location Resolver* then performs the following logical steps:

- it determines the Access Point's MAC/BSSID with the strongest RSSI among the four scans;

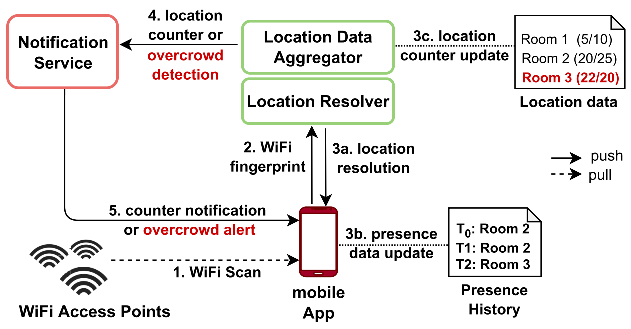


FIGURE 2. Absolute device localization process in normal or overcrowd conditions (red text): information flow among system components.

- it checks whether the anonymous user-ID has already been associated with the same AP, a different one, or none;
- it looks up in a pre-loaded table which indoor location (e.g., hall, corridor, etc.) corresponds to the selected Access Point;
- it sends the selected dominant AP's MAC/BSSID, along with the associated location name, to the originating device (step 3a), which, in turn, updates its local presence history records (step 3b).

Following the absolute localization of a device, two more actions occur.

At the server side, a dedicated service, the *Location Data Aggregator*, updates the counters related to the involved Access Points for overcrowd monitoring (step 3c). For each monitored location, if the counter exceeds a configurable threshold, (i.e. Room 3's counter), all the active devices will be notified with an overcrowd alert or, conversely, the devices will only be notified with the updated counter (steps 4 and 5).

At the mobile DApp side, the selected dominant AP's MAC/BSSID will be used to build and temporarily store an internal object, named `Presence Data`, which can be represented, in a loose JSON notation, as:

```
{AP-BSSID, Timestamp, Time-To-Live}
```

to be exploited in the contact tracing matching phase, as explained in Subsection III-C. The collection of these internal objects makes up the `Presence History`, defined as the temporally ordered sequence of presences stored on a device and not yet expired.

C. CONTACT TRACING WORKFLOW

Figure 3 shows the contact tracing process, made up of the steps discussed below:

1) if a positive-tested user wishes to adhere to the containment program, they should send their `Presence History` to the central server provided that the app has been authorized beforehand by means of a code released by an administrator;

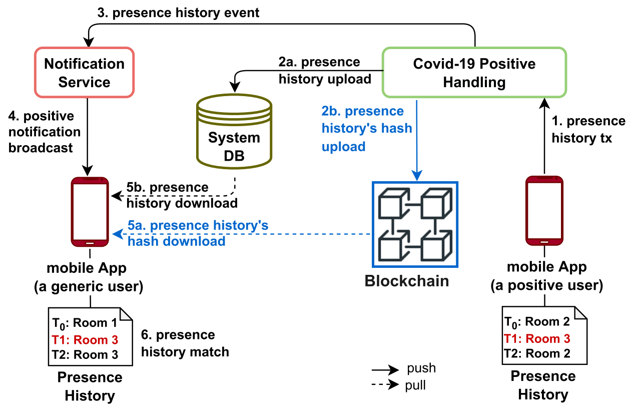

3. presence history event
Notification Service
2a. presence history upload
Covid-19 Positive Handling
System DB
2b. presence history's hash upload
1. presence history tx
4. positive notification broadcast
5b. presence history download
5a. presence history's hash download
mobile App (a generic user)
Blockchain
mobile App (a positive user)
T0: Room 1
T1: Room 3
T2: Room 3
6. presence history match
Presence History
push
pull
T0: Room 2
T1: Room 3
T2: Room 2
Presence History

FIGURE 3. Contact tracing process, involving smart contracts (blue text and arrows), with positive matching (red entries in presence histories).

2) the server writes the `Presence History` of the positive user to a newly created DB item (2a) and stores a hash-pointer to the latter in a blockchain dedicated data structure (2b);
3) the indexed references to the stored blockchain information are then transmitted by the server to the *Notification Service*;
4) the *Notification Service* pushes a contagion risk notification to the user apps along with the indexed references to the blockchain data structure which, in turn, contains the hash-pointer to the `Presence History` of the positive user;
5) each notified app pulls the `Presence History` of the new positive user from the relevant DB location (5b), through the hash-pointer previously retrieved from the blockchain (5a);
6) each app correlates, according to a given heuristics, places and times in the `Presence Data` contained in the pulled `Presence History` of the positive user with its locally stored own, in order to detect direct and indirect contacts.

IV. SYSTEM BACK-END

The overall back-end architecture is made up of three major components: a microservice-based central server, which acts as an administration entity for the whole system; a notification service, and a blockchain platform, which extends the back-end original design by adding the immutability and tamper-proof properties to the management of contact tracing and overcrowd conditions.

Figure 4 shows the central server microservice architecture.

Microservices have been implemented using the well known *Spring* (https://spring.io) Java Framework to obtain small and process-independent services.

The central server interacts with an external *Notification Service*, hosted by the *Firebase Cloud Messaging* (https://firebase.google.com/products/cloud-messaging) platform, which provides a scalable solution to send push notification to the mobile DApp devices.

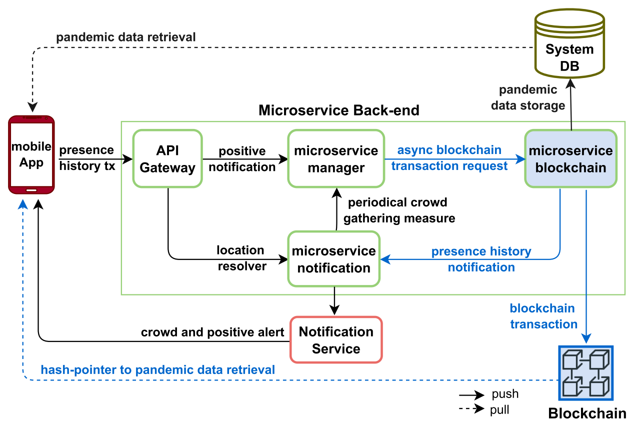


FIGURE 4. Back-end architecture with blockchain integration (blue items).

Three microservices have been implemented:

1) a *microservice-manager* responsible for managing notifications from infected users;
2) a *microservice-notification* interacting with the *Notification Service* to both prepare the notifications for the mobile DApp users and periodically measure gatherings at university locations to be later forwarded to the blockchain;
3) a *microservice-blockchain*, which adds a back-end component to the decentralized app architecture and is responsible for:
   - the sending of pandemic data to the system database and hash-pointers the blockchain storage;
   - the creation of the blockchain transactions upon asynchronous requests issued by the *microservice-manager*;
   - the interaction with the *microservice-notification* to start the alert process towards the Dapp clients.

A. BLOCKCHAIN COMPONENTS

As introduced in Subsection III-A Nausica@DApp is a complete contact tracing solution whose decentralized approach is enhanced by the introduction of blockchain elements, which add new features of open trustworthiness among distinct involved entities.

In our prototype, Ethereum (https://ethereum.org/en) has been identified as the development blockchain platform due to its mature programming model and its account-based paradigm, which characterizes its virtual machine and storage design [19]. Moreover, the availability of advanced environments for smart contract deployment and testing, such as the Kovan (https://kovan.etherscan.io) and the Ropsten (https://ropsten.etherscan.io) testnets, has further prompted the choice of the Ethereum ecosystem for our experiments.

In our decentralized approach, enabled by the deployed smart contract logic, user presence information can now be collected in either a centralized or distributed storage and made reachable to the involved parties in a transparent, authenticated and unforgeable manner. Moreover, crowd gathering data can be aggregated and stored on similar data repositories and exposed for further processing to any concerned individual or organization by means of a blockchain-driven access mechanism.

Hereafter, the features of the smart contracts that have been integrated in our solution, and their mutual interactions, are described. A graphical representation is given in Figure 5.

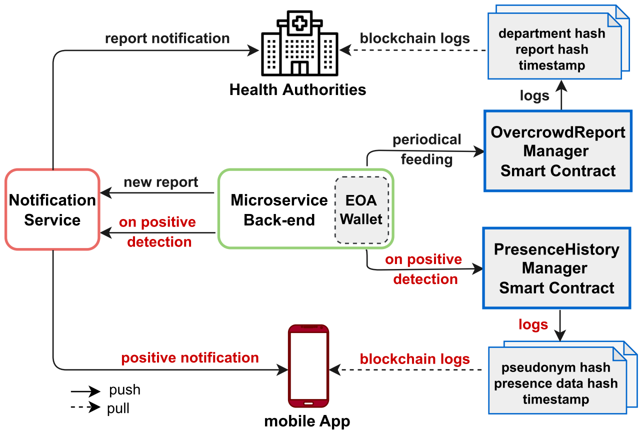


FIGURE 5. Interactions of smart contracts with system components in two scenarios: positive detection (red) and overcrowd reporting (black).

The smart contract design envisages two smart contracts, one for each working scenario, namely "contact tracing" and "overcrowd monitoring".

In the contact tracing scenario a *PresenceHistoryManager* contract is used. It is a single-instance smart contract deployed and interacted with through the relevant blockchain back-end microservice which, in turn, is associated with an Ethereum Externally Owned Account (EOA) [19]. The scenario flows according to the following operations:

1) each time the server receives a notification from a positive-tested user's device it securely stores the relevant `Presence History` (see Subsection III-C) in a DB item, whose related hash-pointer is now computed;
2) the server's *microservice-blockchain* sends the hash-pointer and the SHA3 hash of the anonymized user ID to the *PresenceHistoryManager*, by triggering a proper transaction containing the associated function call;
3) as soon as the block containing the above transaction is validated, the *PresenceHistoryManager* creates an event containing the two received hashes and the block timestamp that are included in the blockchain transaction logs, according to the Ethereum Virtual Machine programming paradigm, for external applications to retrieve them;
4) upon confirmation of the execution of the triggered transaction, the *microservice-blockchain* sends the *Notification Service* the hash-pointer to the stored `Presence History`; this will be notified to all Nausica@DApp-enabled mobile devices, so that they

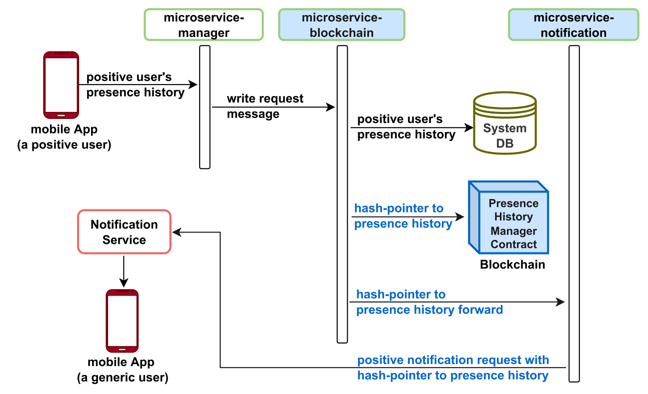

microservice-manager
microservice-blockchain
microservice-notification
positive user's presence history
mobile App (a positive user)
write request message
positive user's presence history
System DB
hash-pointer to presence history
Presence History Manager Contract
Blockchain
Notification Service
hash-pointer to presence history forward
mobile App (a generic user)
positive notification request with hash-pointer to presence history

FIGURE 6. Positive user reporting workflow through system components.

can download the `Presence History` from the DB through the received hash-pointer;

5) finally, the mobile devices can perform contact tracing matching with their own locally stored data.

B. POSITIVE DETECTION WORKFLOW

A simplified workflow of the scenario presented in the previous subsection is depicted in Figure 6, with regard to how the distributed system services and components interact.

The process starts when a Nausica@DApp-enabled device, belonging to a positive user, notifies the *microservice-manager* and transmits its locally collected `Presence History` linked to the anonymized user-ID.

These user data are first passed onto the *microservice-blockchain*, by means of an internal write request message, hashed and then stored in the hash-referable System DB. The distinctive behavior driven by the introduction of the blockchain features starts now: the *microservice-blockchain*, through its Externally Owned Account (EOA) component, creates the transaction carrying the hash-pointer to the DB location storing the `Presence History`, and sends the created transaction to the *PresenceHistoryManager* smart contract. This, in turn, emits the event logging the hashed information needed by the Nausica@DApp-enabled devices to retrieve the positive user's anonymized `Presence History` from the system DB.

Upon reception of a transaction completion message, the *microservice-blockchain* sends the hashed information, stored in the blockchain transaction logs, to the *microservice-notification*. This calls the *Notification Service* to alert all the registered users that fresh data is coming in from a new infected user. The notification payload envelops the hash-pointer to the `Presence History` along with a timestamp returned by the blockchain.

Integration of both centralized and decentralized system DBs has been tried and successfully accomplished in the prototype system. A cloud-based MongoDB's instance (https://www.mongodb.com) has been tested as a centralized key-value solution to host pandemic bulk data off-chain. As an experimental decentralized counterpart, the InterPlanetary File System (IPFS) (https://ipfs.io) has been selected. The centralized option is advisable when additional access rules to the stored data are required by the application context; on the other hand, adopting a decentralized DB extends the decentralization paradigm advocated in our approach up to the main storage component.

Dissemination of the overcrowd monitoring data follows a similar business logic as the previously described ones. A specialized *OvercrowdReportManager* contract, owned by the *microservice-blockchain* EOA, periodically (e.g. twice a day) creates transactions that deal with the SHA3 hash of contagion-related aggregate data, such as the occupation rates of each monitored location. An occupation rate data object is typically made up of a `site-ID`, a `timestamp`, and aggregation values such as `maximum number` and `average number` of attendees. The smart contract execution and the notification service are very similar to the contact tracing scenario's. The stored data will be made publicly available to a variety of stakeholders (health authorities, students, auditors, etc.) who can access them through a blockchain-enabled reference mechanism and additional system DB access control rules.

V. SYSTEM SOFTWARE

This section gives more detailed specifications about the system software, especially with regard to the blockchain-related code.

A. MICROSERVICE-BLOCKCHAIN SOFTWARE

Figure 7 shows a *microservice-blockchain*'s code fragment with the `InfectionManagerBlockchainService()` constructor creating the *Presence Manager Contract*. The developed Java classes import the Web3j library (https://github.com/web3j/web3j) that wraps, at a higher programming level, the JSON-RPC APIs exposed by the Ethereum nodes to access the blockchain.

```
public class InfectionManagerBlockchainService {
  private InfectionManager infectionManager;
  public InfectionManagerBlockchainService() {
    HttpService httpService = new HttpService(AppConfig.BLOCKCHAIN_ENDPOINT_URL)
          ;
    Web3j web3j = Web3j.build(httpService);
    Credentials cred = Credentials.create(AppConfig.
          BLOCKCHAIN_MAIN_ACCOUNT_SECRET_KEY);
    infectionManager = InfectionManager.load(AppConfig.
          INFECTION_CONTRACT_ADDRESS, web3j, cred, Constants.GAS_LIMIT,
          Constants.GAS_PRICE);}}
```

FIGURE 7. Creation of the presence manager contract.

As mentioned in Subsection IV-A, an Externally Owned Account (EOA), managed by a digital wallet facility, needs to be linked to the microservice in order to enable the creation, the deployment and the interaction with the *Presence Manager Contract*. The EOA is created, in our prototype testbed, by means of the *MetaMask* browser extension (https://metamask.io), which handles the EOA balance, and digitally signs the transaction generated by the *microservice-blockchain* of the administration server.

Our code uses the `Web3j.build(httpService)` method call to establish a connection between the *microservice-blockchain* software and a http end-point, which, in turn, acts as an access point to the blockchain network. The endpoint is provided by the cloud-based Infura infrastructure (https://infura.io), which is semantically equivalent to a proxy Ethereum node.

B. SMART CONTRACTS

The smart contracts employed by Nausica@DApp have been developed, deployed and tested by means of specialized tooling such as Remix (https://remix.ethereum.org). Deployment and testing on the Kovan and Ropsten testnets have been carried out with the intermediation of Metamask. Here we discuss *PresenceHistoryManager* as an example, *OvercrowdReportManager* having a basically similar structure.

The code listing in Figure 8 refers to the contact tracing scenario in which, as an outcome of the execution of function `newInfection()`, the *PresenceHistoryManager* emits the event `newHistoryInserted()`, thus storing in the transaction logs, inside the *Receipt Trie* of the Ethereum blockchain, the following parameters:

1) `pseudonymHash`, the infected user's hashed pseudonym, for indexed research of associated events;
2) `presenceHistoryHash`, representing the hash-pointer to the JSON `Presence History` DB object;
3) `timestamp`, corresponding to the block's time registration in the chain.

Once the transaction has been successfully completed, the *microservice-blockchain* can initiate the presence history notification process, previously discussed in Subsection IV-A and depicted in Figure 4. This will eventually supply the mobile apps with all the references to the blockchain transaction logs, which contain hash-pointers to DB items storing the anonymized infected user's `Presence History`.

```
contract PresenceHistoryManager {
  address payable public owner;
  //event reporting the infected user's presence history hash and its timestamp
  event newHistoryInserted(bytes32 indexed pseudonymHash, bytes32
        presenceHistoryHash, uint timestamp);
  constructor() {
    owner = msg.sender; }
  //function to create a new infection's log entry
  function newInfection(bytes32 pseudoHash, bytes32 presenceHash) external {
      emit newHistoryInserted(pseudoHash, presenceHash, block.timestamp); }
  function destroy() external {
    require(msg.sender == owner);
    selfdestruct(owner); }}
```

FIGURE 8. Presence history smart contract.

```
const contractAddress = '0x820813999ad2b1c28415a4daead0d998268b1a48';
//imports contract metadata
const metadata = JSON.parse(await remix.call('fileManager','getFile','
      PresenceHistoryManager.json'));
//creates a contract instance
let contract = new web3.eth.Contract(metadata.abi, contractAddress);
//sends the transaction to the contract - pseudo and presence are test data
const receipt = await contract.methods.newInfection(pseudo, presence).send({
  from: contract.defaultAccount});
```

FIGURE 9. Test code sending data to a smart contract.

Figure 9 shows a fragment of JavaScript test code mimicking the interaction of *microservice-blockchain* with the above *PresenceHistoryManager* contract. In the script, the transaction call (to the contract's `newInfection()` function) is fed with test data (`uid`, `prs`). Parameters (cf. 1-3 above) of the just triggered infection event will be emitted by the smart contract, and consequently stored in the transaction logs, to be thereafter made available to client apps for contact matching. The code makes use of a few web3.js (https://web3js.readthedocs.io) library methods, which hide the JSON-RCP APIs, as previously discussed in the *microservice-blockchain* case.

The design of smart contract `PresenceHistoryManager` endeavors to keep things simple and effective, thus optimizing the overall cost of the Ethereum Virtual Machine operations, as better detailed in Subsection VI-B. From the security point of view, storing data hashes in the public blockchain guarantees that the information stored in the system database is not tampered with. Moreover, the choice to store the bulk data in a separate database fosters the adoption of extra security features, such as data confidentiality and access user authentication, which are not inherently provided by public blockchains instead.

C. CLIENT SOFTWARE

The client software workflow follows a pattern that starts from the server-side *Notification Service*, as discussed in Subsection III-C, and depicted in Figures 3 through 5. It is this service that prompts the client apps to pull, from the blockchain transaction logs, the hash-pointer to either the contact tracing data or the overcrowd report data, as applicable. Client apps will thus be able to de-reference the data structures stored in the system DB, possibly after honoring additional DB access rules, and perform data authentication of any DB item by comparison with its unforgeable hash-pointer stored in the blockchain.

The front-end code of the mobile DApp client has been developed in the Java language under the Android Studio IDE (https://developer.android.com/studio). As said for *microservice-blockchain*, the DApp's front-end exploits the Web3j library to smoothly interface with the Ethereum's JSON-RPC APIs exposed by the blockchain nodes.

```
const contractAddress = '0x820813999ad2b1c28415a4daead0d998268b1a48';
//imports contract metadata
const metadata = JSON.parse(await remix.call('fileManager','getFile','
      PresenceHistoryManager.json'));
//creates a contract instance
let contract = new web3.eth.Contract(metadata.abi, contractAddress);
//retrieves from the blockchain logs the hashed references related to a user's
      hashed pseudo-ID
const events = await contract.getPastEvents('newHistoryInserted', {
  filter: { from: contractAddress,
    pseudonymHash:
    '0xbb27bef7fede14d52825920b5c55650dc753047c9a9f900f9b83a12686b870f8'},
  fromBlock: 0 })
//prints the retrieved information
events.forEach(element => {
  console.log(element.returnValues['presenceHistoryHash'])
  console.log(element.returnValues['timestamp']) })
```

FIGURE 10. Test code to pull transaction logs generated by a smart contract.

The Javascript test code fragment in Figure 10 simulates the essential behavior of a web app pulling previously logged data from the blockchain, by calling the web3.js `getPastEvents()` method. The example code extracts from the blockchain a hash-pointer to `Presence History` items in the database.

VI. EXPERIMENTAL RESULTS

A. ABSOLUTE LOCALIZATION TESTS

The strategic decision to adopt an absolute localization paradigm, relying on the WiFi campus infrastructure, has been validated by the tests carried out on the Nausica@DApp system prototype. Some experimental results are described in this subsection.

The adopted localization methodology is based on the idea that, inside any campus indoor location (e.g., classroom or laboratory), the internal WiFi AP will be the one radiating the dominant signal strength or RSSI. Thus, WiFi fingerprints that mobile devices periodically send to the *Location Resolver* service (see Figure 2) easily enable it to precisely locate devices using a simple maximum-signal logic. In principle, however, there is scope to implement more sophisticated localization strategies (including AI-based ones) within the *Location Resolver*.

Location information is used by back-end services *Location Data Aggregator* for occupation rate monitoring, and *Covid-19 Positive Handling* to support direct and indirect contact tracing operations.

The testbed setting involved six people, equipped with Nausica@DApp-enabled Android devices, wandering within and among lecture halls. Each device creates a WiFi fingerprint by performing four consecutive WiFi scan requests and collecting their outcomes in a JSON scan quadruple

```
Anonymous-User-ID: {
    Scan-Time-1: "[AP1-RSSI,..,APn-RSSI]",
    Scan-Time-2: "[AP1-RSSI,..,APn-RSSI]",
    Scan-Time-3: "[AP1-RSSI,..,APn-RSSI]",
    Scan-Time-4: "[AP1-RSSI,..,APn-RSSI]"
}
```

which is sent to the *Location Resolver* back-end service in charge of absolute localization. This process is repeated about every two minutes.

The experimental setup on a testing client machine consists of a variety of (python, awk, sed, bash) scripts. At the client, WiFi scan data are first collected, as the server stores them, by a per-user Firebase listener, then suitably processed by a set of filters, and finally fed to gnuplot for real-time visualization.

Figure 11 plots the results of a trial pattern in which a single device moves from lecture hall LH3 to LH4 then LH2, while being monitored by the listening service.

The plotted RSSI strengths (in dBm) of signals from the Access Points, collected over a suitable time interval, show how the user device, as long as it is within a lecture hall, always receives the internal AP's signal as the strongest one, whereas the crossing points between different RSSI plots reveal that the user is moving to a new hall.

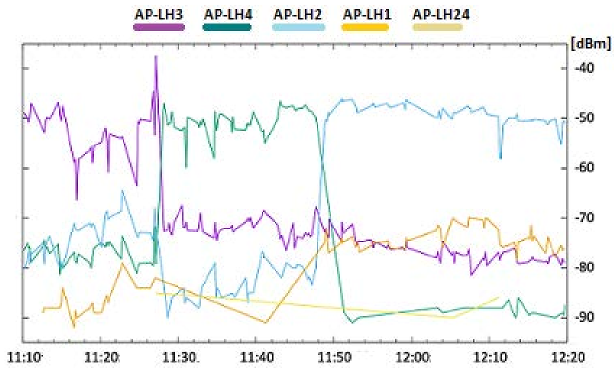


FIGURE 11. Access point signals (in dBm) detected by a user device wandering across three different lecture halls in a suitable time interval.

The plot reports just one of the replicated measurement tests carried out independently by all six devices in various locations within the campus premises. These tests all showed the same reassuring results. Other tests involved positioning the six devices in different spots of the same lecture hall, to verify that at all spots the dominant signal would be that of the hall's AP.

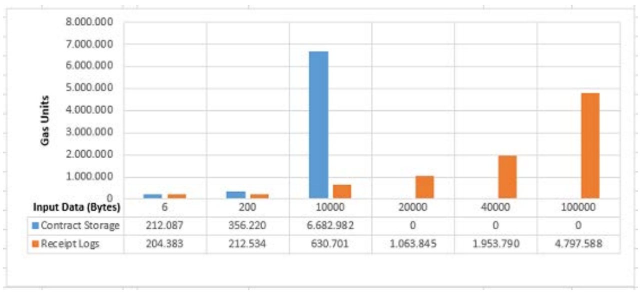
These converging experimental results confirmed that it is sensible to implement a maximum-signal logic, within Nausica@DApp, to identify the "local" AP's MAC to be included in the `Presence Data`.

B. GAS CONSUMPTION TESTS

The storage capabilities of the two Nausica@DApp smart contracts have been limited to the writing on transaction logs of references (the hash-pointers) to bulk data stored elsewhere. The rationale behind this design choice, besides the security concerns discussed in Subsection IV-A, lies in the well-known gas-related issues with the Ethereum Virtual Machine, when it comes to permanently storing bulk data in its data structures. According to Ethereum's yellow paper [17], writing data to the permanent storage of a smart contract is by far more expensive than using, for the same purpose, the transaction logs, stored in the blockchain's Receipt Trie. This was confirmed by the experimental results presented below on the dichotomy between the (smart) *contract storage* vs. *transaction logs* solutions.

An early, experimental version of our system [26] exhibits a more articulated smart contract design, which stores contact tracing and overcrowding data right to *contract storage*. Although a quite refined storage management was implemented, in that an unneeded user presence history could be removed from the blockchain by destroying its associated contract, the gas used for a typical transaction execution would end up exceeding the block gas limit, due to the size of stored data.

However, even in a later refinement, in which bulk data were instead stored in *transaction logs*, gas usage would still easily ramp up beyond any sensible limit.



Input Data (Bytes)	6	200	10000	20000	40000	100000
🟦 Contract Storage	212.087	356.220	6.682.982	0	0	0
🟧 Receipt Logs	204.383	212.534	630.701	1.063.845	1.953.790	4.797.588

FIGURE 12. Comparison of gas consumption for storage of transaction data: smart contracts vs. transaction logs.

In Figure 12 gas consumptions, for both the said storage solutions, are plotted against the amount of data transferred from the server-side by *microservice-blockchain*.

Note that the "contract storage" solution runs out of gas as soon as data size exceeds 10,000 Bytes (hence the missing blue bars in the graph), causing the transaction not to be completed by the Ethereum Virtual Machine. The graph shows that storing to "transaction logs" performs better, but data larger than 100,000 Bytes (the graph's upper bound) still prevent transactions from completing.

In real terms, considering the gas price and the Ether market value at the time of writing, a transaction storing 10,000 Bytes on-chain would cost (in US $), for each of the two above-mentioned solutions:

- Contract Storage:

$$6,682,982[gas\ unit] * 8[gwei/gas\ unit] \\ * 1.43 \cdot 10^{-6}[\$/gwei] = 76.45[\$]$$

- Transaction Logs:

$$630,701[gas\ unit] * 8[gwei/gas\ unit] \\ *1.43 \cdot 10^{-6}[\$/gwei] = 7.22[\$]$$

Even the second solution is by far too expensive to legitimate storing bulk data in the blockchain transaction logs.

Notice that both the previous, unviable, solutions are purely on-chain, storage-wise. The solution presented in this paper, instead, is a mixed on-chain/off-chain one, for it stores essential hashed data in the transaction logs and bulk data in the system DB. This turns out to yield, for our sample transaction, a fixed gas usage of 25,257 gas units, whose current price in US dollars is:

- Mixed on-/off-chain (Nausica@DApp):

$$25,257[gas\ unit] * 8[gwei/gas\ unit] \\ *1.43 \cdot 10^{-6}[\$/gwei] = 0.28[\$]$$

therefore an acceptable cost per transaction for the Ethereum ecosystem, dramatically lower than the computed costs of the previous solutions. This reduced cost results from the combined effect of employing the transaction logs as a storage repository (as in the "transaction logs" solution), and strictly limiting the amount of data stored in the blockchain.

VII. RELATED WORK

In the present section an overview of comparable solutions is proposed, loosely classified according to their application scope (i.e. indoor/outdoor) or technical approach (i.e. centralization/decentralized).

A. INDOOR CENTRALIZED SOLUTIONS

EPIC is a solution [27] that focuses on providing a fine-grained response to user localization in direct-only contact tracing by exploiting short-range wireless technologies, namely WiFi and Bluetooth. EPIC follows a diffused approach, with localization data collected and stored at client side and contact tracing data processing transferred to the server upon positive user detection. Much emphasis is therefore placed on cryptographic techniques when data are to be moved from the client to the server side.

WifiTrace is a network-centric solution for contact tracing where user localization is based on passive WiFi sensing and data processing, strongly relying on WiFi enterprise network logs, does not involve any client intervention [9]. It mainly addresses post-processing of device trajectories of campus attenders.

The approach of the above indoor solutions are strongly dependant on users fully trusting central and third-party authorities. None of the two integrates blockchain components. Conversely, Nausica@DApp does not have to face trust concerns about fully reliable central entities, since it is a decentralized solution with regard to contact tracing assessment and relies on strong de-anonymization of user information. It also enhances data authenticity and operation transparency because of the blockchain elements in the solution back-end. Furthermore, for privacy preserving reasons, the configuration information of the WiFi infrastructure will only be used to localize the anonymized users within the campus premises, but by no means to track users.

B. OUTDOOR CENTRALIZED SOLUTIONS

One of the earliest and most controversial contact tracing solutions is South Korea's *Corona-100m*. Its central server collects the absolute locations of users by means of the GPS or cellular networks with the aim of making them available at a public website to help people track where (anonymized) infected people have moved about. These data can be de-anonymized for the benefit of health investigators who can thus bypass the preservation of individual privacy (https://koreaherald.com/view.php?ud=20200311000132).

Singapore's *TraceTogether* (https://tracetogether.gov.sg) is another relevant solution in which contact tracing assessment and contagion risk alerts are centrally carried out. Differently from Corona-100m, relative localization data are collected at user side and cryptographically exchanged with the server. Similarly to the Korean solution, information about reported positive users can be de-anonymized by the Ministry of Health, for more effective epidemiological surveillance actions [6].

These two centralized solutions have very little in common in terms of privacy choices with Nausica@DApp, which has been designed to provide anonymity and authenticity to user data, even though tracing of indirect contacts is obtained centrally in terms of absolute localization data.

C. OUTDOOR DECENTRALIZED SOLUTIONS

BeepTrace [28] is a very sophisticated blockchain-based solution, which makes use of two different chains. The first chain, named tracing blockchain, allows users to store their anonymized localization data, additionally encrypted with sophisticated protocols involving PKI Central Authorities. Upon virus-positive finding, user localization data must be endorsed by a diagnostician, who further de-anonymize them before storing in the second chain, named notification blockchain, to the benefit of specialized players (i.e. geodata solvers) who can, in turn, assess the contagion risk rate available to the involved users from the notification blockchain.

BeepTrace is a very comprehensive technological proposal that adopts quite complex mechanisms to address stringent security issues. However, the complexity of the proposed solution raises performance and scalability issues, so that a lightweight consensus protocol, such as Direct Acyclic Graph, is advised.

An interesting proposal by *Song et al.* [29] exploits the blockchain as a trustless repository to store location-based and individual-based information for both direct and indirect contact tracing, as well as for risk contagion assessment, to be further processed by the central server. This heavy data burden on the blockchain affects the performance and the scalability of the whole system. The solution also features an algorithm that operates on the collected data.

Nausica@DApp, instead, makes a more confined use of the blockchain functionalities; it is in fact designed with the idea in mind that their adoption must be traded off against the overall system performance. Decentralization of contact tracing processing and storing of only reference data in the blockchain are choices that converge towards this target.

Another relevant proposal by *Marbouh et al.* [30] implements a blockchain-based system that makes use of Ethereum smart contracts and oracles (https://docs.ethhub.io/built-on-ethereum/oracles/what-are-oracles) to assess the reliability and trustworthiness of the information received by the public and government agencies. The main goal of the solution is then to allow dashboards and DApps to retrieve aggregate data only coming from registered external sources, which have incrementally obtained a high degree of reputation, according to a specific logic developed on a specific smart contract. This solution does not deal at all with contract tracing and crowd gathering monitoring, hence does not share the same goals as ours. Indeed, it shows some interesting oracle features that can be taken into consideration for further development.

VIII. CONCLUSION

This paper presented Nausica@DApp, a comprehensive solution for epidemiological monitoring within the premises of a University campus (or similar corporate sites). The proposed system suits the overcrowd surveillance needs and complements recommended health-related best practices for community activities, especially as contagion risks presently seem to be not fully contained by vaccine campaigns.

Nausica@DApp is a decentralized app solution capable of tracing both direct person-to-person contacts and indirect, location-based ones, by taking advanced privacy and security measures. These include the adoption of a decentralized and deanonymized design to store and process user data, and the introduction of blockchain functionalities to enforce user data integrity and operation transparency. Data exchange via secure app-to-server and server-to-app channels, rather than the direct app-to-app Bluetooth one, enhances the overall security features of the presented solution.

In our proposal, a public blockchain technology, such as PoW-based Ethereum, has been specifically used as a secured, transparent, immutable shared repository of limited and specific data. Having chosen not to overuse the Ethereum storage capabilities, but only to leverage those operations bringing out the key feature for which the blockchain usage is widely recognized, i.e., data integrity assurance in a publicly consensus-driven environment, it is fair to assert that the adoption of the presented smart contracts design is experimentally viable.

On the network performance side, more enhancements, in terms of scalability, transaction costs and security, can be achieved replacing the public blockchain of Nausica@DApp by one with PoS or PoA consensus mechanisms. E.g., the emerging Ethereum 2.0 (https://ethereum.org/en/upgrades) is a proper choice for applications in the healthcare arena, where the decentralization dimension of the ''blockchain trilemma'' can be downsized in favor of scalability and security. Similar results can be obtained through a consortium blockchain (https://analyticssteps.com/blogs/what-consortium-blockchain), in which the consensus level is managed by validators belonging to multiple organizations operating in the same sector.

Further enhancements of the proposed solution have successfully explored various alternatives to store the tamper-proof bulk data in key-value storage systems, such as a cloud-based, centralized DB or a P2P decentralized architecture, where ad-hoc user authentication policies can be added to guarantee a more regulated and secure access, as adopted in many blockchain-based healthcare solutions.

Overall, the proposed solution leaps forward other existing contact tracing ones, and the implemented improvements are numerous. These include the technology used for localizing users, the ability to timely distribute aggregated data without compromising the decentralized system performance and communication costs, the ability to preserve user privacy, as well as to efficiently notify users whenever a contact with a positive user is detected.

REFERENCES

[1] Q. Feng, D. He, S. Zeadally, M. K. Khan, and N. Kumar, "A survey on privacy protection in blockchain system," *J. Netw. Comput. Appl.*, vol. 126, pp. 45–58, Jan. 2019.

[2] L. Garg, E. Chukwu, N. Nasser, C. Chakraborty, and G. Garg, "Anonymity preserving IoT-based COVID-19 and other infectious disease contact tracing model," *IEEE Access*, vol. 8, pp. 159402–159414, 2020.

[3] Q. Tang, "Privacy-preserving contact tracing: Current solutions and open questions," 2020, *arXiv:2004.06818*.

[4] H. Cho, D. Ippolito, and Y. W. Yu, "Contact tracing mobile apps for COVID-19: Privacy considerations and related trade-offs," 2020, *arXiv:2003.11511*.

[5] M. Zastrow, "South Korea is reporting intimate details of COVID-19 cases: Has it helped?" *Nature*, Mar. 2020, doi: 10.1038/d41586-020-00740-y.

[6] J. Bay, J. Kek, A. Tan, C. S. Hau, L. Yongquan, J. Tan, and T. A. Quy, "BlueTrace: A privacy-preserving protocol for community-driven contact tracing across borders," Government Technol. Agency, Singapore, Tech. Rep., Apr. 2020.

[7] A. Khurshid, "Applying blockchain technology to address the crisis of trust during the COVID-19 pandemic," *JMIR Med. Informat.*, vol. 8, no. 9, Sep. 2020, Art. no. e20477.

[8] T. McGhin, K.-K. R. Choo, C. Z. Liu, and D. He, "Blockchain in healthcare applications: Research challenges and opportunities," *J. Netw. Comput. Appl.*, vol. 135, pp. 62–75, Jun. 2019.

[9] A. Trivedi, C. Zakaria, R. Balan, A. Becker, G. Corey, and P. Shenoy, "WiFiTrace: Network-based contact tracing for infectious diseases using passive WiFi sensing," in *Proc. ACM Interact., Mobile, Wearable Ubiquitous Technol.*, vol. 5, no. 1, 2021, pp. 1–26.

[10] I. Bashir, *Mastering Blockchain: Distributed ledger Technology, Decentralization, and Smart Contracts Explained*. Birmingham, U.K.: Packt, 2018.

[11] L. Ricci, D. D. F. Maesa, A. Favenza, and E. Ferro, "Blockchains for COVID-19 contact tracing and vaccine support: A systematic review," *IEEE Access*, vol. 9, pp. 37936–37950, 2021.

[12] G. Singh and J. Levi, "MiPasa project and IBM blockchain team on open data platform to support COVID-19 response," IBM, Armonk, NY, USA, Tech. Rep., Mar. 2020. [Online]. Available: https://www.ibm.com/blogs/blockchain/2020/03/mipasa-project-and-ibm-blockchain-team-on-open-data-platform-to-support-covid-19-response/

[13] A. Chawla and S. Ro, "Coronavirus (COVID-19)–is blockchain a true savior in this pandemic crisis," SSRN, Jul. 2020, doi: 10.2139/ssrn.3655337.

[14] G. Marotta, F. Billeci, G. Criscione, F. Merola, G. Pappalardo, and E. Tramontana, "NausicaApp: A hybrid decentralized approach to managing covid-19 pandemic at campus premises," in *Proc. Asia Conf. Comput. Commun. (ACCC)*, Sep. 2020, pp. 124–129.

[15] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Tech. Rep., 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[16] A. M. Antonopoulos, *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. Sebastopol, CA, USA: O'Reilly Media, 2014.

[17] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, Apr. 2014.

[18] D. Mingxiao, M. Xiaofeng, Z. Zhe, W. Xiangwei, and C. Qijun, "A review on consensus algorithm of blockchain," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2017, pp. 2567–2572.

[19] A. M. Antonopoulos and G. Wood, *Mastering Ethereum: Building Smart Contracts and Dapps*. Sebastopol, CA, USA: O'Reilly Media, 2018.

[20] L. Luu, J. Teutsch, R. Kulkarni, and P. Saxena, "Demystifying incentives in the consensus computer," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 706–719.

[21] Q. Zhou, H. Huang, Z. Zheng, and J. Bian, "Solutions to scalability of blockchain: A survey," *IEEE Access*, vol. 8, pp. 16440–16455, 2020.

[22] S. Badruddoja, R. Dantu, Y. He, K. Upadhayay, and M. Thompson, "Making smart contracts smarter," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, May 2021, pp. 254–269.

[23] G. Kwon, J. Kim, J. Noh, and S. Cho, "Bluetooth low energy security vulnerability and improvement method," in *Proc. IEEE Int. Conf. Consum. Electronics-Asia (ICCE-Asia)*, Oct. 2016, pp. 1–4.

[24] S. Vaudenay, "Analysis of DP3T," Cryptol. ePrint Arch., Paper 2020/399, 2020. [Online]. Available: https://eprint.iacr.org/2020/399

[25] D. Jaisinghani, R. K. Balan, V. Naik, A. Misra, and Y. Lee, "Experiences & challenges with server-side WiFi indoor localization using existing infrastructure," in *Proc. 15th EAI Int. Conf. Mobile Ubiquitous Syst., Comput., Netw. Services*, Nov. 2018, pp. 226–235.

[26] G. Marotta, A. Fornaia, A. Moschitta, G. Pappalardo, and E. Tramontana, "NausiChain: A mobile decentralized app ensuring service continuity to university life in covid-19 emergency times," in *Proc. 4th Int. Conf. Softw. Eng. Inf. Manage.*, Jan. 2021, pp. 74–81.

[27] T. Altuwaiyan, M. Hadian, and X. Liang, "EPIC: Efficient privacy-preserving contact tracing for infection detection," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.

[28] H. Xu, L. Zhang, O. Onireti, Y. Fang, W. J. Buchanan, and M. A. Imran, "BeepTrace: Blockchain-enabled privacy-preserving contact tracing for COVID-19 pandemic and beyond," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3915–3929, Mar. 2020.

[29] J. Song, T. Gu, Z. Fang, X. Feng, Y. Ge, H. Fu, P. Hu, and P. Mohapatra, "Blockchain meets COVID-19: A framework for contact information sharing and risk notification system," 2020, *arXiv:2007.10529*.

[30] D. Marbouh, T. Abbasi, F. Maasmi, I. A. Omar, M. S. Debe, K. Salah, R. Jayaraman, and S. Ellahham, "Blockchain for COVID-19: Review, opportunities, and a trusted tracking system," *Arabian J. Sci. Eng.*, vol. 45, pp. 1–17, Dec. 2020.

ANDREA FORNAIA received the B.S., M.S., and Ph.D. degrees in computer science from the University of Catania, Italy, in 2011, 2014, and 2018, respectively. He is currently an Adjunct Professor with the Department of Mathematics and Computer Science, University of Catania. In 2009, he won a Scholarship with the National Institute of Nuclear Physics (INFN), Catania, where he mainly worked on monitoring solutions for data centers and on application porting over grid and cloud infrastructures. In 2012, he won a Scholarship with the GARR Consortium, the Italian Academic and Research Telecommunication Network, by proposing a solution for a cloud-based recovery as a service (RaaS service) capable of building an on-demand private cloud over existing grid resources. He is the author of about 30 journals and conference papers. His research interests include software quality, covering subjects, such as software testing, static analysis, and code refactoring. He is also strongly interested in distributed systems technologies and IT infrastructures.

GIOVANNI MAROTTA received the B.S. degree in computer science and the M.S. degree in physics from the University of Catania, in 1987 and 2010, respectively, where he is currently pursuing the Ph.D. degree with the Department of Mathematics and Computer Science, on leave from teaching. He was a Visiting Researcher at AT&T Bellcore, making contribution to metropolitan area networks services, a Resident Researcher at the HP Laboratories, working on network security and communication protocols, and the Project Manager at Nokia Networks, dealing with system integration solutions. His current research interests include the design and prototyping of blockchain decentralized solutions in non-financial sectors, such as document management, healthcare and green energy market, with special focus on decentralized storage, digital identities, and messaging architectures. He is the corresponding author of three papers in the blockchain field.

GIUSEPPE PAPPALARDO received the M.S. degree in electrical engineering from the University of Catania, in 1983, and the Ph.D. degree in computing science from the University of Newcastle, Newcastle upon Tyne, U.K. He began his research activity in the field of computer science with the University of Catania, the Polytechnic of Milan, and the University of Newcastle, where he was a Researcher. Then, he was a University Researcher and an Associate Professor in information processing systems at the Faculty of Engineering, University of Reggio Calabria, and an Associate Professor. Since 2002, he has been a Full Professor in computer science with the Department of Mathematics and Information Technology (DMI), University of Catania. His research interest includes the area of distributed systems. He has been involved in formal specification and verification techniques, fault tolerance, software architectures, image processing, robotic systems, and smart cities. He is the author of over 150 scientific publications in the field of computer science. He was and is the Scientific Manager of the DMI or the University of Catania of various research projects, including PRIN, POR, and PON, in the IT and smart cities sectors.

EMILIANO TRAMONTANA is currently an Associate Professor with the Department of Mathematics and Computer Science, University of Catania. He was a Full Professor with National Agency. He was a Research Associate with the Department of Computing Science, Newcastle University, Newcastle upon Tyne, U.K., for three years. In the past, he was a Visiting Researcher at The University of Tokyo, Japan, and University of Waseda, Japan. He is the Appointed Teacher of software engineering for the degree course in computer science at the University of Catania, and distributed system engineering for the master's degree course in computer science. Since 2005, he has been the Co-Chair of various editions of the thematic tracks dedicated to distributed systems and software engineering techniques at the IEEE WETICE and ACM SAC international conferences. He is the Chief Editor of *Scientific Programming* journal (Hindawi) and the Editorial Board of *Concurrency and Computation Practice and Experience* (Wiley). He is the author of over 150 scientific publications, and he has been responsible for several research projects funded after a competitive selection.

Open Access funding provided by 'Università degli Studi di Catania' within the CRUI CARE Agreement