Scalable IoT architecture for balancing performance and security in mobile crowdsensing systems*

Theodoros Nestoridis
School of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
nestorid@csd.auth.gr

Chrysa Oikonomou
School of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
oikochry@csd.auth.gr

Anastasios Temperekidis
School of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
anastemp@csd.auth.gr

Fotios Gioulekas
School of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
gioulekas@csd.auth.gr

Panagiotis Katsaros
School of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
katsaros@csd.auth.gr

***Abstract*—Crowdsourcing aims to deliver services and content by aggregating contributions from a large user population. For mobile networks and IoT systems, crowdsourcing is used to gather and process sensor data from mobile devices (crowdsensing), in order to deliver real-time, context-aware services and possibly support user collaboration in extended geographic areas. In applications like geonsensitive navigation, location-based activity sharing and recommendations, the challenge of adequate service quality and user experience may be at stake, as the services are provided securely to an ever-growing user population. This happens due to the inherent trade-off between security and real-time performance that ultimately sets in doubt any scalability prospect beyond a certain user-interaction load. This work introduces a publish-subscribe architecture for mobile crowdsensing systems, which can be transparently scaled up to higher usage load, while retaining adequate performance and security by load balancing into multiple MQTT brokers. The security support combines a lightweight TLS implementation with an integrated mechanism for two-level access control: user-device interactions and message topics. We provide proof-of-concept measurements that show how our solution scales to increasing interaction loads through load-balancing the processing cost that includes the overhead of the security mechanisms applied. The system architecture was implemented in a vehicular crowdsensing navigation network that allows to exchange navigation information at real-time, for improved routing of vehicles to their destination.**

***Index Terms*—Internet of Things, crowdsourcing, mobile network, security, load-balancing**

I. Introduction

The system architecture defines the fundamental organization of a system in terms of its software and hardware components, their relationship with each other, and with the environment, as well as the principles guiding its design and evolution. The interactions among the different components impact the overall system performance, the scalability prospects and eventually determine the potential to meet the requirements for a given application. When we specifically focus on IoT systems, it is widely accepted that there is no standardized IoT reference model and architecture that can be applied in a majority of IoT systems [1]. For mobile networks and IoT systems, the common characteristics that affect design decisions usually concern with the inherent tradeoff between performance and security, when a system scales to higher-load usage scenarios. In these cases, we expect the system to retain the same quality of service in terms of performance, while having to face increased computation/communication load due to the cryptographic protocols and authorization requirements that assure adequate security. Moreover, we also expect the system to exhibit high availability (i.e. tolerating faults) and to transparently scale through automated load balancing [2].

Crowdsensing systems depend strongly on how to address the aforementioned challenges, due to the primary concern of success, which is their capacity to support large-scale sensing with mobile devices and humans in the loop [3]. The ultimate purpose of a crowdsensing system is the provision of intelligent services through aggregating and processing user-provided sensor data in the cloud. Today, there are numerous crowdsensing applications based on various mobile devices (smartphones, portable navigation and wearables) and various forms of human participation, including mobile social networking. In the context of this paper, we focus on a vehicular crowdsensing navigation system, as a use case scenario to motivate and evaluate our proposal for a scalable crowdsensing system architecture.

This is not the first research work focusing on the architecture of crowdsesning systems. In [4], the authors emphasize on distinguishing the centralized cloud-based architectures from the so-called edge computing architectures, which aim to provide massive-scale crowdsensing IoT services that would not be feasible before the advent of 5G networks. The edge computing architecture introduces an additional layer of

*This research has been co-funded by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH - CREATE - INNOVATE II (project code:T2EDK-02617).

computation between mobile devices and cloud services that is responsible for data filtering, aggregation, processing and storage. In effect, such a design enables third parties to run their services and applications at the edge of mobile networks. Such an architecture is presented in [5], where a publish-subscribe middleware for mobile crowdsesnsing is introduced that enables management of mobile sensor resources within the cloud and supports filtering and aggregation of sensor data on mobile devices prior to its transmission to the cloud. Based on experimental evaluation, the authors show that their system exhibits scalable processing performance, both on mobile devices and within the cloud. In [6], the authors present a platform for on-demand energy-aware sensing and server-side storage of different types of data. Finally, in [7] the authors propose a mobile crowdsensing approach based on mobile data stream mining, which reduces the amount of data sent and the energy consumption of mobile devices, while providing an acceptable level of sensing accuracy (i.e. comparable with other models of intermittent/continuous sensing).

However, none of the mentioned related works has focused on the scalability issues of crowdsensing systems, due to the tradeoff between performance and security and the vital requirements of transparent load-balancing and high availability. Security mainly refers to the problems of user/device access control and authorisation of interaction rights [8], which are inevitably application-dependent and we therefore treat them at the system architecture level. Maintaining authorisation information in thousands of mobile devices is simply not feasible [9], and therefore we assume the existence of an authorisation service together with a series of additional security properties for the interactions with the authorisation server: mutual authentication, confidentiality and integrity of the credentials for the mobile devices [10]. Moreover, a lighweight security protocol should guarantee the mutual authentication, confidentiality and integrity of publish-subscribe interactions in the crowdsensing system. We therefore introduce a decentralised security architecture and evaluate its scalability prospects in the context of applying high availability and transparent load-balancing for crowdsensing systems.

The rest of the article is structured as follows. The next section discusses the related work. Section III introduces the proposed publish-subscribe system architecture to specifically address the design challenges in crowdsensing systems. Section IV presents the security architecture. Section V discusses how the proposed system architecture was implemented in the context of our traffic/navigation crowdsensing scenario. Finally, Section VI evaluates the use case system performance for increasing user-interaction loads and the paper concludes with a summary on its contributions and a discussion on future research prospects.

II. Related work

In [11]–[13] and [14] the authors review different types of mobile crowdsourcing applications and discuss the key challenges related to the problem of task allocation, i.e. how to allocate a set of outsourced tasks to a set of mobile users. Mobile crowdsourcing is clearly a different field of applications from the crowdsensing systems, on which we focus in this article.

In [15], the authors review and explore the wide range of crowdsensing applications based on smartphones for capturing and sharing the sensed data between many nodes. Moreover, the challenges related to the data collection methodologies and task management are also discussed.

Regarding the security aspect, [16] refers to related problems faced in mobile social networks, while [17] is focused on a particular architecture used for fog-assisted vehicular crowdsensing. In [18] and [19], the authors introduce novel solutions for access control and authentication specifically for IoT architectures, but none of the mentioned articles addresses the scalability problem, which is essential for applying the security measures, while retaining adequate real-time performance in higher-load usage scenarios.

III. Publish-subscribe architecture for crowdsensing

The MQTT (Message Queuing Telemetry Transport) protocol is designed for telemetry data exchange in real-time, using the publish-subscribe messaging pattern that is shown in Figure 1. This pattern has no solid client roles. Clients can take the role of either a publisher or a subscriber, i.e. they can either share some information or request (subscribe) to receive all information concerning a specific topic. The Broker is the equivalent of the server in the traditional messaging terminology. It is responsible to handle the client subscriptions and then distributing the published information to them. Topics, in the publish-subscribe interaction model, are strings that Brokers use in order to filter messages and send them to the corresponding subscribers. They consist of levels, starting with more general terms and then referring to more specific ones. Topic hierarchies are designed based on the interaction requirements between users and devices of an IoT application and its specific security constraints.

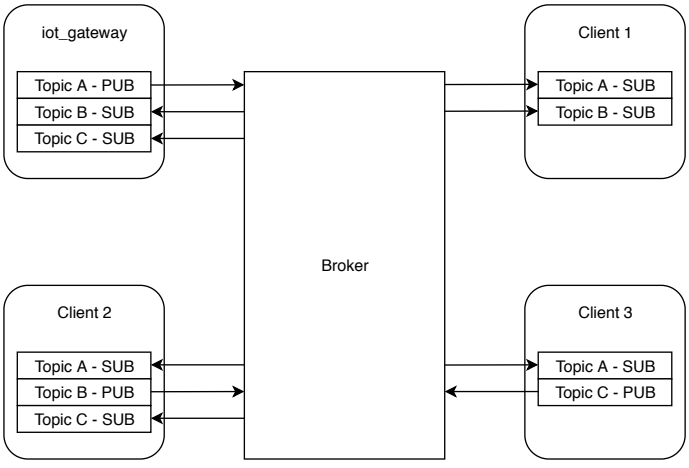

Fig. 1: Publish-subscribe messaging pattern

MQTT is widely used in IoT systems, due to its scalability prospects and its capability to provide differentiated

quality of service (QoS) levels according to the application's interaction requirements. All Broker services are triggered through system events and can be parallelized transparently. Moreover, Brokers can take advantage of cache memory to store and distribute messages to the clients. The overall system architecture and the complete topic hierarchy is transparent to the clients and therefore it does not affect their communication complexity, while at the same it hides vital architectural details of the system's security.

The Broker services can be scaled by using multiple Brokers for load balancing [20], which requires the adoption of appropriate techniques/algorithms in the system architecture [21]. Depending on the particular requirements of a crowdsensing system, the load balancing algorithm may or may not distribute multiple TCP connection instances from the same IP addresses to a single Broker. Also, the system's performance may be affected by whether the load-balancing algorithm reads/uses or ignores the message payload, which may be required by the applied security constraints.

In terms of security, Brokers provide basic configuration options to control the access rights of clients to the topics based on Access Control Lists (ACLs). Publish-subscribe message communications can be authenticated and encrypted using the TLS (Transport Layer Security) protocol, in order to protect their confidentiality/integrity. However, special care should be paid towards using a comparatively light-weight TLS implementation (e.g. based on Elliptic Curve Cryptography (ECC) [22] instead of the classical RSA), in order to avoid any scalability limitations due to the extra overhead implied by the underlying encryption algorithm. The described security options effectively result in a centralized security architecture with important scalability limitations, as they are explained in [10].

An IoT Gateway (Figure 1) enables the integration of third-party systems into the overall system architecture. To allow connection with an external system, the IoT Gateway is a subscriber to the updates of a broker. At the same time, the gateway forwards messages to the topic subscribers either directly or via external systems. Finally, the gateway also acts as a protocol converter, i.e. it converts data from a protocol-specific format to/from another format.

IV. Security architecture for crowdsensing

We describe henceforward the mechanisms used to address the crowdsensing security requirements mentioned in Section I, within the context of a publish-subscribe system architecture.

In a publish-subscribe IoT system, sensing and actuation should be controlled by device authentication and authorisation credentials. A device-centric solution cannot scale to address the needs of a crowdsensing scenario with thousands of mobile devices [9]. Our security architecture adopts a decentralized access control model, where authentication and authorisation services are provided to the Broker(s) by an Authorisation database server and all communications with this server take place through secure channels. Figure 2 describes the steps for a device to establish a connection to the Broker:

1) The device sends its authentication credentials to the Broker.
2) The Broker transmits the credentials to the Authorisation Database.
3) The Authorisation Database responds whether the device is authenticated or not.
4) The Broker sends a confirmation to the device that it has been authenticated for using the Broker's services.
5) The device now can send publish/subscribe messages to the Broker.

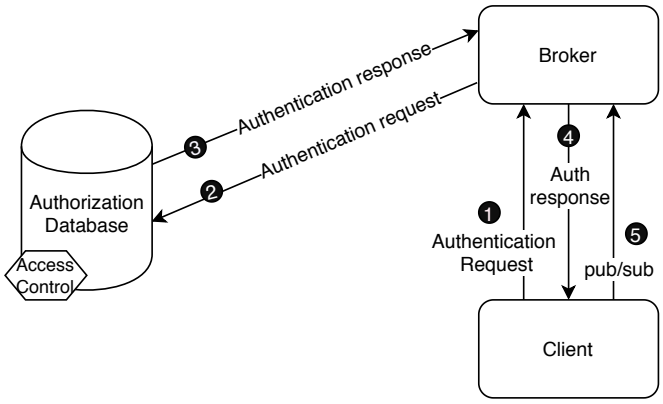

Fig. 2: Authentication Mechanism

In this way, only the authenticated clients can access the Broker, while there is no need for configuring the Broker for each authenticated client, as would be the case in a centralised security architecture. Data sharing through publish/subscribe messages is restricted by ACLs for authenticated clients to the Broker-maintained topics. Thus, only authorised clients can read (subscribe) from and write (publish) to a topic and with respect to this we distinguish two main types of client interactions: the crowdsensing devices and the back-end cloud subsystem (with the mediation of an IoT gateway), which is used to process the crowdsourced data. In order to grant read/write access rights to all interacting clients, we rely on a role-based access control (RBAC) approach.

Figure 3 depicts the procedure according to which the Broker checks a topic's ACL for validating a client's rights to the topic.

Every single IoT gateway maintains information for the topic hierarchies, where new updates are written/published, as well as read/subscribed to/by the clients (including the gateway). This information is used to initialize the ACLs for these topics. In essence, the IoT gateway and the external Authorisation database provide all information needed to grant rights that control how to subscribe/publish to the topics, according to our RBAC scheme.

In the context of our decentralised security architecture, the requirements introduced in Section I are further refined [10] as follows. We guarantee confidentiality and integrity of client credentials through secure communication for the steps 1-5 of Figures 2 and 3, as well as mutual authentication with the Broker and the Authorisation Database based on a light-weight

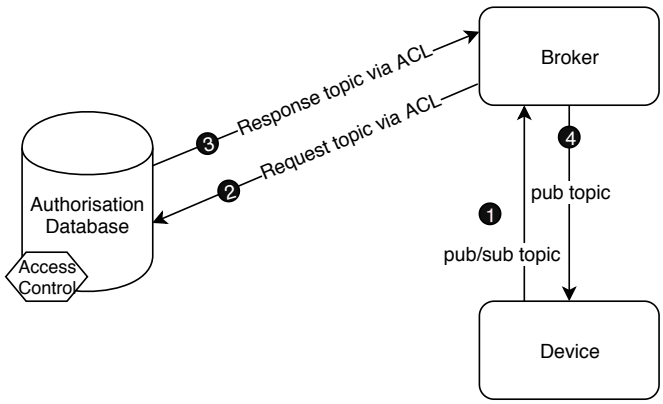

Fig. 3: Access Control List Mechanism

cryptographic mechanism. Finally, all publish/subscribe messages are also protected against possible confidentiality and integrity threats.

To ensure confidentiality and data integrity of client-to-Broker messages, we propose using the Transport Layer Security (TLS) cryptographic protocol with 1-way TLS connections, i.e. connections accepted by the Broker from any client based on public key cryptography. The load-balancing mechanism should be configured to work in the TLS pass-through mode, where the traffic is not decrypted, but it is simply routed through a TCP tunnel between the client and the Broker so as to let them negotiate and handle the TLS session. Public key cryptography is implemented by utilizing digital signatures based on the Elliptic Curve Digital Signature Algorithm (ECDSA) that uses ECC. A key length of 384 bits is more adequate for low-end devices, since we found that it requires relatively small key sizes and operand lengths.

The external Authorisation Database is implemented using the auth plugin option of the Mosquitto broker [23]. This allows using an external system for authentication and access control, towards applying custom username/password mechanisms and access control policies. Communication with the Authorisation Database takes place through a 2-way TLS connection, which allows to mutually authenticate the two parties given that a certificate has been previously issued for both of them.

V. Crowdsensing use case

The architecture described in previous sections was implemented in a traffic/navigation crowdsensing system that aims to the collection, analysis and distribution of traffic data, in order to enhance GPS navigation with real-time updates and optimal vehicle routing. The Navigation mobile app implements common GPS navigation functions, while enabling users to share crowdsensing data (e.g. speed, direction) that are used to infer information at real-time, for the driving context in their current location. This data is processed combined with additional information from other sources, e.g. the weather conditions in a given area, in order to compute optimal routes to the destination of the crowdsensing vehicles.

The system architecture is based on the MQTT protocol for the communication between the Navigation app clients and the backend cloud system. Specifically, we used an open source implementation of the MQTT broker services, the well-known Eclipse Mosquitto broker [23]. Furthermore, we also used the HAProxy open source software service [24] that offers a multitude of load-balancing algorithms for distributing a message load among a series of servers (Mosquitto brokers). Our aim was to achieve higher levels of availability and transparent load-balancing of increasing numbers of simultaneous Navigation app connections assumed to generate publish-subscribe messages in various levels of frequencies. Fault tolerance was implemented using the Keepalived [25] open source service, which performs continuous health checking between the primary and a backup HAProxy service, thus ensuring that the whole system survives single service failures. Upon completing the recovery of a previously failed HAProxy, the same instance takes on again the primary role for servicing incoming MQTT messages. The whole system architecture is shown in Figure 4, where the Authorisation Database is also depicted, which is part of the security architecture, as well as part of the backend data processing engine (Apache Flink for distributed processing of data streams). We omit the details of the rest of the backend system, which do not directly affect the security and scalability issues of our crowdsensing architecture. Finally, an emailing notification mechanism (the POSTFIX service [26]) was also configured, to alert the system administrator for critical events that potentially require him/her to handle.

The HAProxy service receives messages and then filters and forwards them to a broker, while balancing the message load according to the applied algorithm. Our system architecture was configured to work using the so-called *source algorithm*, which is based on a hash function that generates hash codes from the clients IP addresses. Our load-balancing approach was motivated by the need to include support for unicast communication through MQTT. Thus, for each Navigation app client the broker handling its connection will be identified through the hash code, i.e., every Navigation app client is always connected to the system using a single broker and just in case of a broker failure or system reconfiguration, the connection is handled by another broker. In case of a system reconfiguration (e.g. when adding extra broker), the load balancing algorithm updates automatically the weights used by the hash function, such that the load is transparently distributed to all available brokers, without any code modification.

Health-checks based on the KeepAlived service take place periodically (every 2 seconds) with negligible communication costs that cannot affect the overall system's performance. Upon a detected failure of the primary HAProxy, KeepAlived notifies the back-up HAProxy, which is pre-configured exactly as its twin HAProxy service and ready to take on the primary role. In overall, an HAProxy instance may be in one of the following distinct modes: primary, backup and faulty. Through the POSTFIX service, the system administrator is notified for each mode change.

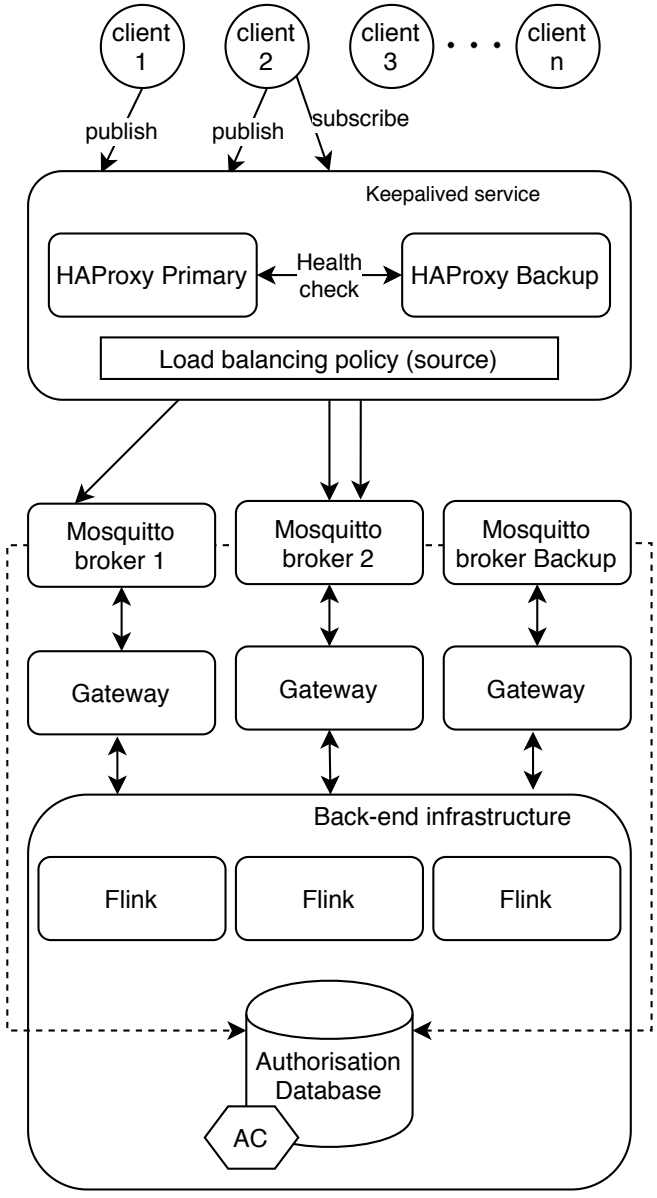

Fig. 4: Traffic/navigation crowdsensing communication subsystem architecture

In the cluster of available MQTT brokers, we employ at least one backup instance (cf. Figure 4), which may be activated upon a detected error to any of the active brokers. The primary HAProxy is responsible for the broker status check and the decision to activate the backup broker(s) when needed.

At the security architecture level, our system was based on the Let's Encrypt Certificate Authority [27], which is a free and open authority that supports our security requirements, i.e. it provides certificates using the ECDSA algorithm with the secp384r1 elliptic curve and a 384-bit key length. The Authorisation Database was connected to the Mosquitto broker using the open-source mosquitto-auth plugin [28], which supports username/password based authentication and authorisation for the access-controlled topics based on an external SQL database. We chose a synchronous multimaster replication architecture (e.g. PostgreSQL Database), thus enabling the provision of authentication and authorisation services over multiple sites. Finally, the gateway functionality in our system architecture was based on the Thingsboard IoT gateway [29], an open source implementation that we extended with additional functionality for the topics synchronization and the initialization of the ACL.

In order to further explain the topics hierarchy, we first need to elaborate on the functional details of our crowdsensing system. For personalized delivery of location-based data updates at real-time and in regular intervals, we would need support for server-side identification of the road segments, on which the Navigation app clients move. However, such a solution would cause redundant data traffic by potentially large groups of clients concentrated in busy urban areas.

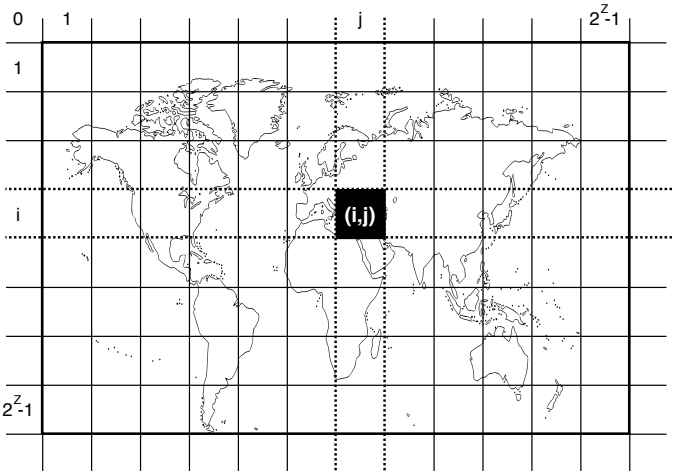

Fig. 5: Map segmentation into tiles

Instead of this, we adopted an alternative approach that is also used by many other navigation systems, such as Bing Maps [30] and OpenStreetMaps [31], which is based on a map segmentation into tiles. For a given zoom level Z, the map is partitioned horizontally and vertically into 2^Z zones, as is shown in Figure 5. Then, an index is assigned to each zone, from 0 to $2^Z - 1$ and the map is partitioned into distinct tiles, each of which can be uniquely identified by its index coordinates (i, j).

We distinguish two topic hierarchies used to route messages with respect to the communication direction. Moreover, according to the structure of the topic hierarchies, each Navigation app client can easily subscribe to one or more neighboring tiles simply by specifying the respective indexes (Figure 6). The categories of messages are the following:

- Navigation app to back-end messages. The app clients send a JSON message including the user's location, the tile's information and crowdsensing data (vehicle's speed, road segment, direction etc.), as well as road events such as traffic incidents.

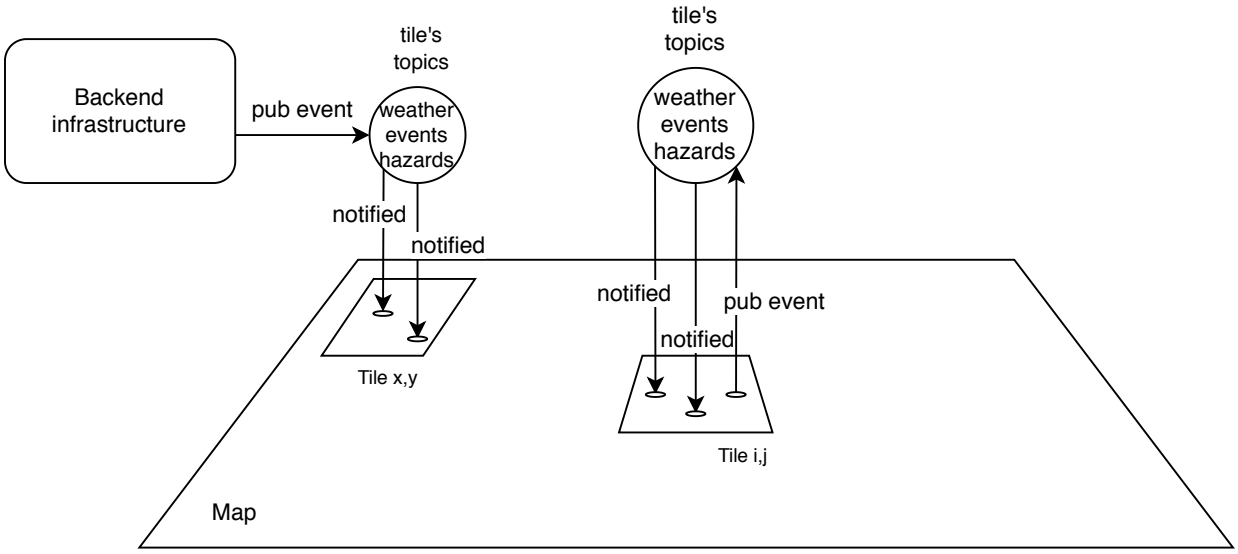

Fig. 6: Tile-based topics hierarchy

- Back-end to Navigation app messages. The back-end broadcasts events that have happened in a given tile or traffic-load estimations, by publishing the respective data to dedicated topics. All users that are currently moving to this specific tile - therefore subscribed to the corresponding topic - are then notified.

Further security protection is provided against Navigation app users trying to abuse the two topic hierarchies, through the pattern option, which adds a 256-bit (32-byte) hash value to the topic string that is given as the result of the SHA256 cryptographic hash function.

Our crowdsensing system supports multitenancy, i.e. the system can simultaneaously serve multiple tenants and each tenant represents a different group of users (Navigation app clients for taxi companies, public transport operators etc) with specific priviliges to publish and subscribe to the available topics. Authorisation management is based on RBAC combined with the access control for the topics hierarchy, thus grouping the management of large numbers of Navigation client access rights into well-defined groups.

VI. Scalability and performance evaluation

Our crowdsensing system architecture was evaluated with respect to its performance and scalability potential in a laboratory environment where the HAProxy and Broker services were installed in a workstation with the following hardware specification:

TABLE I: Machine specs

Processor	Cores	RAM	OS
Intel Core i7-3770K, 3.50GHz	8	16 GB	64-bit Ubuntu

Two sets of experiments took place, with each of them having been implemented using a script for generating MQTT

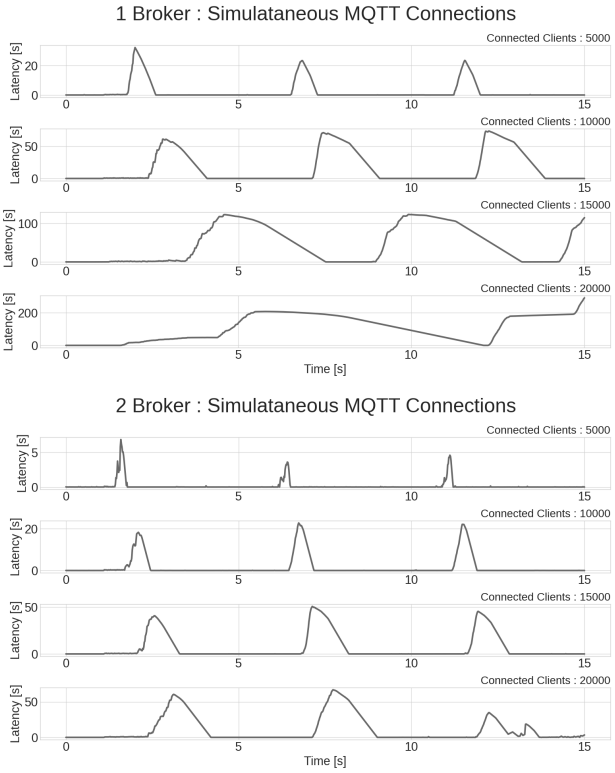

Fig. 7: Experiments with simultaneous MQTT connections.

app clients that connect to the infrastructure with appropriate credentials; connected app clients continuously sent messages of 60 Kbytes, with an average of 30 messages per minute for each app client. The first set of experiments concerns with the worst case scenario of simultaneous connection of multiple app clients to the infrastructure, while the second was implemented to simulate the connection of MQTT app

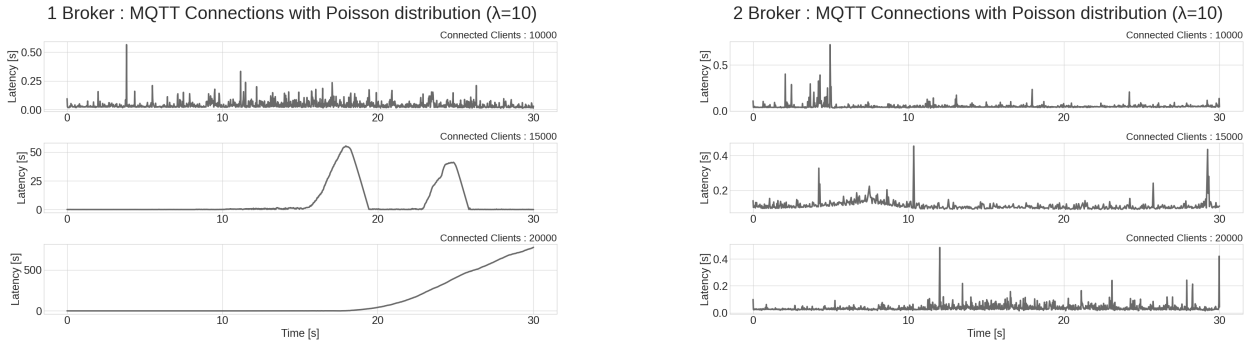

Fig. 8: Experiments with MQTT connections based on Poisson distribution (λ=10).

clients within a certain period of time. The two experiments took place for four different levels with respect to the number of app clients that were increased by a step of 5000 app clients from 5000 to 20000 MQTT app clients. The overall load corresponds to a throughput of 140,000 - 560,000 K messages per minute.

The metric used to validate the smooth operation of our infrastructure for managing real-time data is the observed latency in message communication. The latency is measured from the moment an MQTT app client publishes a message to a topic until the time a subscriber receives it. For a realistic measurement approach, the subscribers network has been implemented based on 3G network metrics [32], which results in different performance, when compared to an ideal cloud network.

The process of an MQTT app client connection is completed when the credentials are successfully sent to the respective broker and it is confirmed that this client is identifiable according to the response of the Authorization Database (authentication part). By sending a publishing message, the MQTT app client declares the topic he wants to send the message, while the broker checks through the Authorization Database, if the client has the rights to write or in case of subscriber to read from the requested topic (authorization part). In any case, the (non-)authorisation of the MQTT app client for each topic is stored in the Broker cache for the next five minutes, so that any mutual request will not be resent at the Authorization Database.

The time constraint of the mosquitto cache memory for the authorization part, results in many more requests being made to the Authorization Database, than if there would be only one at the beginning of the connection (basic option of mosquitto broker). This means that in case of multiple users, if these requests are synchronized at a certain period, there will be an extra delay, since the requests are queued and will have to wait for approval by the Authorization Database. Our experiments examine this specific behavior, while the infrastructure with the load-balancing techniques tries to eliminate it at an acceptable level for our use case scenarios.

The results for the first set of experiments (Figure 7) show the latency of an MQTT app client that connects to the infrastructure, when the number of clients in each case connect simultaneously. The case of simultaneous MQTT app client connection causes simultaneous requests to the Authorization Database, which justifies the first shown peak values in each case. Accordingly, each subsequent peak in latency takes place when the cache time limit of the mosquitto broker expires, with the Authorization Database being bombed with requests causing a service delay for each MQTT app client. We observe that the transition to our load-balancing solution results in a significant reduction in the latencies, which means that the delay is due to the mosquitto broker and the management of requests to the Authorization Database.

The second set of experiments refers to the gradual connection of MQTT app clients to the infrastructure following a Poisson distribution with λ=10 over a period of 24 minutes. As shown in Figure 8, the latency remains at satisfactory levels up to the value of 10,000 MQTT app clients (280 K messages/per minute) per broker, while from then latencies are constantly increased. When using our load-balancing solution, without any other change in the infrastructure, the communication of 20,000 MQTT app clients (560 K messages/per minute) was successfully achieved with the values of latency remaining at acceptable levels.

In overall, the shown experiments indicate that (i) the number of additional messages communicated by our system architecture (including the Authorisation database) is negligible and (ii) the latency remains acceptable in multiple brokers scenarios using our load-balancing solution (Figure 8), while allowing us to retain the same performance and security levels, for significantly higher numbers of connections. Finally, due to our backup HAProxy and broker services and the configured recovery procedures, we also offer a higher level of availability, for the crowdsensing application.

VII. Conclusions

We presented a decentralised security architecture for crowdsensing systems, as an alternative to a straightforward device-centric architecture. The proposed solution is based on an authorisation service (all communications with it take place through secure channels) capable to support user/device

access control of multiple tenants, in the context of a publish-subscribe system architecture that can scale with respect to increasing numbers of crowdsensing devices, while retaining the performance and security levels.

The main motivation behind our design was to devise an approach in order to effectively manage the inherent tradeoff between security and real-time performance, when a system scales to higher-load usage scenarios. Our solution was implemented in a vehicular crowdsensing navigation system that allows to exchange navigation information at real-time, for improved routing of vehicles to their destination. The shown experimental results confirm that our load-balancing approach allows to easily distribute the message load in high-load usage scenarios, while offering a sufficient level of availability.

Our future work plans mainly focus on further developing access control schemes that utilize the integrated role-based mechanism with the publish-subscribe authorisation system towards addressing security requirements of diverse crowdsensing application scenarios.

REFERENCES

[1] G. Gardašević, M. Veletić, N. Maletić, D. Vasiljević, I. Radusinović, S. Tomović, and M. Radonjić, "The iot architectural framework, design issues and application domains," *Wireless Personal Communications*, vol. 92, no. 1, pp. 127–148, Jan 2017.

[2] H. Guo, J. Ren, D. Zhang, Y. Zhang, and J. Hu, "A scalable and manageable iot architecture based on transparent computing," *Journal of Parallel and Distributed Computing*, vol. 118, pp. 5 – 13, 2018.

[3] B. Guo, C. Chen, D. Zhang, Z. Yu, and A. Chin, "Mobile crowd sensing and computing: when participatory sensing meets participatory social media," *IEEE Communications Magazine*, vol. 54, no. 2, pp. 131–137, February 2016.

[4] M. Marjanović, A. Antonić, and I. P. Źarko, "Edge computing architecture for mobile crowdsensing," *IEEE Access*, vol. 6, pp. 10 662–10 674, 2018.

[5] A. Antonić, M. Marjanović, K. Pripužić, and I. P. Žarko, "A mobile crowd sensing ecosystem enabled by cupus: Cloud-based publish/subscribe middleware for the internet of things," *Future Generation Computer Systems*, vol. 56, pp. 607 – 622, 2016.

[6] R. Ben Messaoud, Z. Rejiba, and Y. Ghamri-Doudane, "An energy-aware end-to-end crowdsensing platform: Sensarena," in *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, Jan 2016, pp. 284–285.

[7] W. Sherchan, P. P. Jayaraman, S. Krishnaswamy, A. Zaslavsky, S. Loke, and A. Sinha, "Using on-the-move mining for mobile crowdsensing," in *2012 IEEE 13th International Conference on Mobile Data Management*, July 2012, pp. 115–124.

[8] S. Ravidas, A. Lekidis, F. Paci, and N. Zannone, "Access control in internet-of-things: A survey," *Journal of Network and Computer Applications*, vol. 144, pp. 79 – 101, 2019.

[9] W. Feng, Z. Yan, H. Zhang, K. Zeng, Y. Xiao, and Y. T. Hou, "A survey on security, privacy, and trust in mobile crowdsourcing," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2971–2992, Aug 2018.

[10] V. Beltran and A. F. Skarmeta, "Overview of device access control in the iot and its challenges," *IEEE Communications Magazine*, vol. 57, no. 1, pp. 154–160, January 2019.

[11] J. Ren, Y. Zhang, K. Zhang, and X. Shen, "Exploiting mobile crowdsourcing for pervasive cloud services: challenges and solutions," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 98–105, March 2015.

[12] L. Shu, Y. Chen, Z. Huo, N. Bergmann, and L. Wang, "When mobile crowd sensing meets traditional industry," *IEEE Access*, vol. 5, pp. 15 300–15 307, 2017.

[13] Y. Wang, X. Jia, Q. Jin, and J. Ma, "Mobile crowdsourcing: framework, challenges, and solutions," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 3, p. e3789, 2017.

[14] X. Kong, X. Liu, B. Jedari, M. Li, L. Wan, and F. Xia, "Mobile crowdsourcing in smart cities: Technologies, applications, and future challenges," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8095–8113, Oct 2019.

[15] K. Abualsaud, T. M. Elfouly, T. Khattab, E. Yaacoub, L. S. Ismail, M. H. Ahmed, and M. Guizani, "A survey on mobile crowd-sensing and its applications in the iot era," *IEEE Access*, vol. 7, pp. 3855–3881, 2019.

[16] K. Zhang, "Security and privacy for mobile social networks," 2016. [Online]. Available: http://hdl.handle.net/10012/10418

[17] S. Basudan, "Secure and privacy-preserving fog-assisted vehicular crowdsensing," 2018. [Online]. Available: http://hdl.handle.net/10155/962

[18] N. Fotiou, T. Kotsonis, G. F. Marias, and G. C. Polyzos, "Access control for the internet of things," in *2016 International Workshop on Secure Internet of Things (SIoT)*, Sep. 2016, pp. 29–38.

[19] N. Fotiou and G. C. Polyzos, "Authentication and authorization for inter-operable iot architectures," in *Emerging Technologies for Authorization and Authentication*, A. Saracino and P. Mori, Eds. Cham: Springer International Publishing, 2018, pp. 3–16.

[20] P. Jutadhamakorn, T. Pillavas, V. Visoottiviseth, R. Takano, J. Haga, and D. Kobayashi, "A scalable and low-cost mqtt broker clustering system," in *2017 2nd International Conference on Information Technology (INCIT)*, Nov 2017, pp. 1–5.

[21] A. B. Prasetijo, E. D. Widianto, and E. T. Hidayatullah, "Performance comparisons of web server load balancing algorithms on haproxy and heartbeat," in *2016 3rd International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*, Oct 2016, pp. 393–396.

[22] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, Aug 2001.

[23] "Eclipse mosquitto." [Online]. Available: https://mosquitto.org/

[24] D. Corbett, S. Caus, and W. Tarreau, "The world's fastest and most widely used software load balancer." [Online]. Available: https://www.haproxy.com/

[25] "Keepalived for linux." [Online]. Available: https://www.keepalived.org/

[26] "The postfix home page." [Online]. Available: http://www.postfix.org/

[27] "Free ssl/tls certificates." [Online]. Available: https://letsencrypt.org/

[28] Jpmens, "Authentication plugin for mosquitto with multiple back-ends "mosquitto-auth-plug"," Mar 2019. [Online]. Available: https://github.com/jpmens/mosquitto-auth-plug

[29] "Thingsboard iot gateway." [Online]. Available: https://thingsboard.io/docs/iot-gateway/what-is-iot-gateway/

[30] "Bing maps tile system." [Online]. Available: https://docs.microsoft.com/en-us/bingmaps/articles/bing-maps-tile-system

[31] "Quadtiles." [Online]. Available: https://wiki.openstreetmap.org/wiki/QuadTiles

[32] C. Serrano, B. Garriga, J. Velasco, J. Urbano, S. Tenorio, and M. Sierra, "Latency in broad-band mobile networks," in *VTC Spring 2009 - IEEE 69th Vehicular Technology Conference*, 2009, pp. 1–7.