AloTtalk
A SIP-Based Service Platform for Heterogeneous Artificial Intelligence of Things Applications
Yang, Shun-Ren; Lin, Yi-Chun; Lin, Phone; Fang, Yuguang

Published in:
IEEE Internet of Things Journal

Published: 15/08/2023

Document Version:
Post-print, also known as Accepted Author Manuscript, Peer-reviewed or Author Final version

Publication record in CityU Scholars:
Go to record

Published version (DOI):
10.1109/JIOT.2023.3265674

Publication details:
Artificial Intelligence of Things Applications. IEEE Internet of Things Journal, 10(16), 14167-14181.
https://doi.org/10.1109/JIOT.2023.3265674

Citing this paper
Please note that where the full-text provided on CityU Scholars is the Post-print version (also known as Accepted Author
Manuscript, Peer-reviewed or Author Final version), it may differ from the Final Published version. When citing, ensure that
you check and use the publisher's definitive version for pagination and other details.

General rights
Copyright for the publications made accessible via the CityU Scholars portal is retained by the author(s) and/or other
copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal
requirements associated with these rights. Users may not further distribute the material or use it for any profit-making activity
or commercial gain.

Publisher permission
Permission for previously published items are in accordance with publisher's copyright policies sourced from the SHERPA
RoMEO database. Links to full text versions (either Published or Post-print) are only available if corresponding publishers
allow open access.

Take down policy
Contact lbscholars@cityu.edu.hk if you believe that this document breaches copyright and provide us with details. We will
remove access to the work immediately and investigate your claim.
AIoTtalk: A SIP-Based Service Platform for Heterogeneous Artificial Intelligence of Things Applications

Shun-Ren Yang, Member, IEEE, Yi-Chun Lin, Phone Lin, Fellow, IEEE, and Yuguang Fang, Fellow, IEEE

Abstract—Recently, several Internet of Things (IoT) service platforms have been proposed to facilitate IoT application deployment. These platforms typically utilize the lightweight MQTT or CoAP application protocol, optimized for massive IoT applications. Unfortunately, these protocols are not suitable for the emerging, more sophisticated Artificial Intelligence of Things (AIoT). Session Initiation Protocol (SIP), in contrast, is a signaling and controlling protocol for real-time multimedia sessions, and has been viewed as a better candidate to provide a full range support of different broadband, critical, and industrial AIoT applications. However, there exists no generic SIP-based AIoT service platform that supports creations and operations of heterogeneous AIoT applications with various quality of service. This paper presents the first SIP-based AIoT service platform, AIoTtalk, that enables rapid development of scalar and multimedia AIoT applications. Moreover, we deploy an experimental testbed and two real SIP-based AIoT applications to demonstrate the applicability and the performance of our AIoTtalk under both the cloud and edge scenarios. The experimental results show that, together with accurate model predictions and edge-virtualization auto scaling, AIoTtalk guarantees low latency and high quality of experience for messaging and streaming-based AIoT applications.

Index Terms—Artificial Intelligence of Things (AIoT), edge virtualization, Internet of Things (IoT), service platform, Session Initiation Protocol (SIP).

I. INTRODUCTION

Internet of Things (IoT) represents a connected world of a multitude of physical and virtual “things”, capable of sensing and communications. Recently, with the advancement of 5G networking, artificial intelligence (AI), and big data analytics, the ever-emerging paradigm of AI of Things (AIoT) further allows IoT systems/applications to analyze, learn, and react to external IoT devices more promptly and intelligently.

In a typical IoT network architecture, an IoT service platform in the cloud provides all fundamental functionalities for IoT applications, such as data storage, device management, and application execution. In the past few years, several generic [1]–[7] and customized [8]–[19] IoT service platforms have been proposed to facilitate IoT application deployment. Such IoT service platforms utilize IoT application protocols to communicate with IoT devices. Widely adopted IoT application protocols range from the early high-overhead HTTP to the more recent lightweight MQTT [20] and CoAP [21] protocols. Indeed, MQTT and CoAP are suitable for massive IoT applications, which require small data volumes and extreme coverage for a large number of low-cost and narrow-bandwidth devices, operating mostly based on the short instant messaging semantic. Unfortunately, the aforementioned protocols are not suitable for the emerging, more advanced IoT applications that are mainly multimedia streaming-oriented: 1) broadband IoT applications, which require high data rate and large data volumes; 2) critical IoT applications, which demand ultra-reliable data delivery and ultra-low latency; 3) industrial IoT applications for automation, which demand the integration of mobile connectivity with wired industrial networking infrastructures for advanced industrial automation [22].

The Session Initiation Protocol (SIP), in contrast, is a protocol used for signaling and control for real-time multimedia sessions within voice, video and messaging applications. Besides instant messaging for one-shot small messages, SIP can also handle the long session semantic for streaming data and the publish-subscribe semantic for event notifications. Thus, to provide a full range of support for various and more sophisticated IoT application scenarios, SIP can be a better choice. Moreover, SIP has been a key protocol in the IP Multimedia Subsystem (IMS) within the 4G LTE/5G NR core networks. More details of the SIP procedures will be given and explained in Sec. V-B. Different SIP semantics have been utilized to implement different IoT applications/systems [23]–[26]. For example, the short instant messaging semantic has been applied to smart home automation [23], and the long session and publish-subscribe semantics have been utilized to realize a SIP-based IoT emergency system [25]. However, to our best knowledge, there exists no generic SIP-based AIoT service platform that integrates relevant capabilities for application developers to flexibly develop/deploy heterogeneous

Manuscript received November 18, 2022; revised February 07, 2023; accepted March 25, 2023. Date of publication X X X; date of current version X X X.

S.-R. Yang is with the Department of Computer Science and the Institute of Communications Engineering, National Tsing Hua University, Hsinchu 30013, Taiwan (e-mail: sryang@cs.nthu.edu.tw).

Y.-C. Lin is with the Department of Computer and Information Engineering, National Taiwan University, Taipei 10617, Taiwan (e-mail: plin@csie.ntu.edu.tw).

Y. Fang is with the Department of Computer Science, City University of Hong Kong, Hong Kong, China (e-mail: my.fang@cityu.edu.hk). "Copyright (c) 2023 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org."
AIoT applications that demand diversified quality of service (QoS). Furthermore, none of the previously proposed SIP-based IoT systems can readily support creations and operations of AIoT applications.

This paper aims to implement a SIP-based AIoT service platform, AIoTtalk, which leverages and enhances IoTtalk [6], a generic IoT application management platform, by equipping it with the SIP signaling capability and with security guarantees. The main contributions of this paper are summarized as follows.

- We propose a SIP-based AIoT service architecture, based on which we design and implement our AIoT service platform, namely, AIoTtalk, that enables rapid development of heterogeneous (scalar or multimedia) IoT, particularly AIoT applications.
- We establish a real-world testbed under both the cloud and edge scenarios, and deploy two real SIP-based AIoT applications, “real-time short-term road traffic prediction” and “neighborhood violence detection”, within AIoTtalk, to validate the applicability of our AIoTtalk service platform.
- We conduct extensive experiments using this testbed and the two real SIP-based AIoT applications to investigate the performance of our AIoTtalk.

This work is extended from our VTC2021-Spring conference paper [27] with the following new features: 1) a thorough related-work survey; 2) the enhanced service architecture to support multimedia IoT/AIoT applications; 3) the enhanced SIP msg handler to process multimedia streaming data; 4) a hybrid blockchain architecture that allows privacy preservation and public accountability; 5) AI devices for AIoT applications; 6) enhanced testbed and experiment results; 7) an intensive future-research discussion.

II. RELATED WORK

This section reviews the existing IoT service platforms, which can be classified into HTTP/MQTT/CoAP-based and SIP-based according to their adopted application protocols. Finally, we provide a critical analysis.

A. HTTP/MQTT/CoAP-Based IoT Service Platforms

We classify these IoT service platforms into generic and customized.

1) Generic IoT Service Platforms

- **Open-Source Platforms**: OpenMTC [1] and FIWARE [2] are two well-referenced open-source IoT platforms. In OpenMTC, the core features and connectivity of the front-end and the back-end implement the functionality for message exchanges between devices and the middleware, providing the core logic of the platform. On the other hand, FIWARE is an improved OpenStack-based cloud, in which the IoT edge includes the IoT gateway to establish and manage the communications between the devices and the IoT back-end.

- **Proprietary Platforms**: Amazon Web Services (AWS) IoT [3] and Microsoft IoT Core [4] are cloud-based IoT platforms, designed to connect IoT devices to AWS and Azure cloud platforms, respectively, enabling quick development, management, and scaling. In [5], by applying the social IoT concept, a cloud-based IoT platform, Lysis, has been proposed, supporting the establishment of social relationships among objects in an autonomous manner. In [6], IoTtalk, a Python-based IoT management platform, is designed to allow users to define or reuse input/output device features and create heterogeneous IoT applications via a user-friendly web-based GUI. In [7], TinyLink has been presented, following a top-down approach to IoT applications’ hardware/software design, where unified APIs allow to specify the interactions with the underlying hardware components.

2) Customized IoT Service Platforms

- **Smart Health**: In [8], iHome Health-IoT, an intelligent home-based platform, has been implemented, which operates a medicine box, pharmaceutical packaging, and a wearable bio-medical sensor device. In [9], Pathinarpothi et al. have presented an IoT-based smart edge system for remote health monitoring, employing two novel software engines to process wearable vital sensors’ data: 1) rapid active summarization for effective prognosis and 2) criticality measure index alerts. In [10], a configurable and adaptable platform is designed for comprehensive healthcare parameter monitoring. Via a flexible IoT gateway, this platform can establish bidirectional communications between end users and medics in real time.

- **Smart City**: In [11], an IoT software infrastructure has been presented, enabling energy management and new control policy simulation in a city district. In [12], the authors have developed PortoLivingLab, an urban-scale, multisource sensing infrastructure, exploiting IoT-inspired data collection strategies to facilitate sharing and joint analysis for the characterization of urban dynamics. In [13], Cirillo et al. have offered methodologies to minimize the costs of developing new smart city solutions while maximizing component reusability, aiming at forming a live technical community of smart city application creators.

- **Smart Home**: In [14], a fog computing platform has been proposed to manage energy consumption with the customized control-as-services, and minimize the implementation cost and time-to-market simultaneously. In [15], an IoT energy management system has been presented for smart homes, which applies off-the-shelf business intelligence and big data analytics software packages to better manage energy consumption and satisfy consumer demands. In [16], a self-learning home management system is given, which integrates home energy, demand side, and supply side management sub-systems for a smart home’s real-time operation.

- **Smart Industry**: Under the project “Enabling Business-Based Internet of Things and Services”, Khaleel et al. [17] have developed an IoT platform and the related prototypes, where IoT middleware is deployed to
TABLE I: Critical literature analysis

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Multimedia AloT</td>
<td>MQTT/CoAP</td>
<td>HTTP-like</td>
<td>SIP</td>
</tr>
<tr>
<td>Application Support</td>
<td>Partial</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Security Guarantee</td>
<td>Partial</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Service Generality</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Transaction Public Accountability</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

support seamless integration of involved heterogeneous components. In [18], Kwon et al. have introduced the concepts of IoT-based prognostics and systems health management, an enabling discipline that utilizes sensors to assess system health, diagnoses anomalous behavior, and forecasts the remaining useful asset performance. In [19], Wang et al. have reported the first effort on the design of an industrial IoT process control system, seamlessly integrating HART and WirelessHART network infrastructure with hardware/software components of cellular IoT architecture.

B. SIP-Based IoT Service Platforms

Since there exist no SIP-based IoT service platforms, we survey SIP-based frameworks and systems for IoT applications, instead.

In [23], Garroppo et al. have proposed a SIP-based domotics framework. This framework applies a SIP-based home gateway (SHG), which addresses the compatibility with the SIP infrastructure and enables a user to control domotics devices via a SIP client. In [24], Andriopoulou et al. have proposed the IoTa framework, which supports bidirectional communication sessions between healthcare providers and end users and exploits intelligent algorithms to detect emergency events from various devices. The work in [25] has proposed SEEK, a SIP-based embedded framework for emergency situations, consisting of body sensors and a haptic device and enabling embedded devices to initiate calls to emergency centers and support IP-based real-time monitoring for elderly and disabled people. In [26], an IoT-based healthcare system based on Next Generation Network IMS has been presented. Via SIP calls and messages, the proposed system can automatically send real-time data for patients and/or alerts to doctors and place calls for ambulance.

C. Critical Literature Analysis

Table I provides a critical analysis of the literature reviewed in this section. In particular, 1) all the generic platforms in [1]–[7] operate mainly based on MQTT and CoAP, and cannot flexibly support the development of more sophisticated AIoT applications; 2) each of the customized platforms in [8]–[19] is aimed at a specific area and cannot be readily applied to the development of other fields of AIoT applications, exhibiting distinct characteristics; 3) similarly, the SIP-based frameworks and systems in [23]–[26] are limited to unique applications, and cannot serve as generic AIoT service platforms. However, recent popular AIoT use cases, such as video surveillance (broadband IoT), autonomous and connected vehicles (critical IoT), and manufacturing (industrial IoT), all heavily rely on multimedia sensing/visioning manipulation. Such AIoT applications motivate us to develop SIP-based generic service platforms that can support real-time multimedia streaming capability via SIP.

III. OVERVIEW OF IOTALK

In this section, we present a brief overview of IoTalk [6]. IoTalk is an IoT service platform that allows application developers to deploy and manage IoT applications easily. In the network domain, IoTalk operates an IoTtalk server (see Fig. 1 (A)), which establishes and maintains relationships among IoT devices to realize IoT applications. The IoTtalk server consists of three components: the IoTtalk Engine (see Fig. 1 (1)), the Graphical User Interface (GUI; see Fig. 1 (2)) and the Database (DB; see Fig. 1 (3)). The IoTtalk engine classifies IoT devices with their features, executes the service logic of each IoT application, and maintains all related data in the DB. The GUI provides a web-based user interface, via which application developers can instruct the IoTtalk engine to set up the desired interactions between input and output IoT devices.

In the device domain, input IoT devices, like sensors (see Fig. 1 (B)), collect sensing data, while output IoT devices, like actuators (see Fig. 1 (C)), turn actuating results into actions. A sensor and an actuator can “interact” with each other to operate their IoT applications as follows: (1) the sensor executes its Sensor/Actuator Application (SA) to compute the input sensing data and transmit the data to the IoTtalk engine; (2) the IoTtalk engine translates the sensing data to the actuating result, which will be received by the SA in the actuator, triggering the corresponding actions. Note that the communication between the SA (in the sensor/actuator) and the IoTtalk engine is achieved via the associated Device Application (DA), which can handle the HTTP-based RESTful APIs (see Fig. 1 (D)).
Inside the IoTtalk engine, IoT devices are managed in the form of “device features (DFs)”, which are specific input or output “capabilities” of IoT devices, e.g., “Temperature” and “Display.” In fact, the IoTtalk engine realizes the interactions between the SAs of two IoT devices through their input DFs (IDFs) and output DFs (ODFs). When the IDFs of the input IoT device produce new values, these values are transmitted to the IoTtalk engine, where the corresponding (automatically constructed) network application (NA; see Fig. 1 (4)) is executed to produce results on the ODFs of the output IoT device.

IV. PROBLEM DEFINITION AND SERVICE ARCHITECTURE

A. Problem Definition

Our problem can be divided into the following two parts.

1) Service Architecture: We should first design a service architecture that can flexibly support heterogeneous AIoT application operations based on the SIP signaling and IoTtalk. This architecture should meet:

- Requirement 1: supporting both scalar and multimedia sensing/actuating operations via SIP messaging and session semantics, respectively;
- Requirement 2: allowing to define/manipulate AI models in the same way as regular sensing/actuating nodes.

2) Service Platform: Then, we should provide a detailed realization for our service platform, AIoTtalk, the key enabler of our IoT service architecture. This service platform should meet the following requirements:

- Requirement 1: handling SIP signaling;
- Requirement 2: providing quality of service (e.g., security);
- Requirement 3: leveraging IoTtalk’s “service switching” functionality.

B. AIoTtalk Service Architecture

Our AIoTtalk is closely related to IoTtalk. In this subsection, we first discuss its SIP-based IoT service architecture. In the next two sections, we will then give the design and implementation details of each relevant network entity.

As shown in Fig. 2, a set of sensing user agents (S-UAs) (see Fig. 2 (A)) interact with another set of actuating user agents (A-UAs) (see Fig. 2 (B)) via SIP signaling over the SIP network (see Fig. 2 (C)), where the service logics of the IoT applications are hosted and executed in an application environment supported by the AIoTtalk server and the IoTtalk server (see Fig. 2 (D)). In what follows, we explain the operations of this service model from the perspectives of the client and the server, respectively.

1) S-UAs/A-UAs: In our proposed AIoTtalk IoT service architecture, an S-UA \( S_x \) contains \( K_{S,x}^{(s)} \) scalar IoT input devices (SIoT IDs) \( I_{x,1}^{(s)} \sim I_{x,m}^{(s)} \) and \( K_{S,x}^{(m)} \) multimedia IoT input devices (MIoT IDs) \( I_{x,1}^{(m)} \sim I_{x,m}^{(m)} \), while an A-UA \( A_y \) contains \( K_{A,y}^{(s)} \) scalar IoT output devices (SIoT ODs) \( O_{y,1}^{(s)} \sim O_{y,m}^{(s)} \) and \( K_{A,y}^{(m)} \) multimedia IoT output devices (MIoT ODs) \( O_{y,1}^{(m)} \sim O_{y,m}^{(m)} \). Note that the scalar devices here mean that they only manage scalar content (i.e., of the data type of integer, floating point or character). Moreover, an S-UA/A-UA can involve in several IoT applications, under each of which a group of IoT IDs \( [S_x, I_{x,1}^{(s)}, \ldots, I_{x,m}^{(s)}] \)’s (where \( u \) can be \( s \) or \( m \)) can currently interact with another group of IoT ODs \( [A_y, O_{y,1}^{(m)}, \ldots, O_{y,m}^{(m)}] \)’s (where \( v \) can be \( s \) or \( m \)). Below, we will use two examples to demonstrate our design.

- Road traffic prediction. For road traffic monitoring, each road side unit (RSU) (as an S-UA) in a section of a big city can control a large number of roadside vehicle speed sensors (i.e., as IDs) on neighboring road segments and periodically retrieve the sensed speed data. These past and current speed data from all the RSUs of the city section can then be “transformed” into a short-term estimated speed over each road segment of the city section, which will be displayed on the electronic map (i.e., as an OD) of each involved vehicle (as an A-UA) for further usage.

- Neighborhood violence detection. A neighborhood violence detection application can conduct audio analytics to constantly and automatically monitor the neighborhood environment. Specifically, a smart home automation safety system (as an S-UA) deployed at each home of a neighborhood can continuously collect the data of its audio sensors (i.e., as IDs) within its home environment. Such audio sensor data from each home can then be jointly “transformed” into an alert notification to the emergency services (i.e., as ODs) of the surrounding police departments (as A-UAs) of the neighborhood for probable reactions.

Under such group-oriented IoT application scenarios, each S-UA (involving in several applications with different A-UAs) periodically formulates a SIP message, encapsulating its IDs’ sensing data that should be mapped to the corresponding ODs scattered over several involved A-UAs. These sensing SIP messages from numerous S-UAs will then be passed to and translated by the AIoTtalk server (collaborating with the IoTtalk server according to the service logics, which can
be further based on/enhanced by AI-model predictions) into actuating SIP messages, each of which is destined to an A-UA to trigger the corresponding reactions of individual ODs. All the above SIP message transmissions are handled by the underlying network of SIP proxies.

2) **AloTalk Server**: In this IoT service architecture, leveraging the IoTtalk framework, the AloTalk server provides an environment to execute the service logics of the SIP-based IoT applications and manage the involved S-UAs/A-UAs together with their IDs/ODs. Technically speaking, from the viewpoint of IoTtalk, the AloTalk server is essentially an IoTtalk device, which is also equipped with SIP UA functionalities for SIP message handling. Following the IoTtalk service model, the AloTalk server, as an IoTtalk device, implements SA and DA functionalities to handle respective tasks. First, in order to support group-oriented IoT application scenarios, the SA functionality of the AloTalk server allows the developer of a specific application to define its input device group (IDG) and output device group (ODG) in the AloTalk server, and formulate, deploy and execute the service logic of the IoT application as an NA in the IoTtalk server (see Sec. III). Then, the DA functionality of the AloTalk server will automatically create a DA for this specified application (via AloTalk SA). Sharing this DA in AloTalk, the IDG and ODG of the application can connect to their corresponding NA in the IoTtalk, playing the roles of an input and an output devices, respectively. That is, through the DA, the IDG and ODG can inter-operate by periodically updating their IDF/ODF, which are actually mapped by the join connection of the respective NA in the IoTtalk engine.

The AloTalk server internally maintains a service table which records the IDG, the ODG, and the associated DA that jointly realize each IoT application. Acting as a SIP UA, the AloTalk server receives and handles those sensing SIP messages from the S-UAs. Whenever it receives a sensing SIP message from an S-UA, via the service table, it finds the applications in which each enclosed ID is involved, and triggers the IoTtalk server to run the service logic of each of these applications with the ID’s data (as the input parameter) to obtain the new values of the mapped ODs. On the other hand, the AloTalk server will also periodically transmit an actuating SIP message to each A-UA, enclosing the latest values of the A-UA’s ODs. This actuating SIP message can then launch the reaction of each referred OD, accomplishing a complete interaction between the two parties of the corresponding IoT application.

Our AloTalk service architecture further supports AloT applications, where, instead of the direct interaction between each pair of IDG and ODG, the IDG ID data (maybe via some transformation) are first passed to an **online/offline** AI model (residing at the server side) for prediction, whose results (again, maybe via some transformation) are then delivered to the ODG ODs, finally triggering the desired actions. In our AloTalk, such an AIoT application is realized via a pair of IoTtalk NAs as follows. The first IoTtalk NA relays the sensing data obtained from the IDG (as an IoTtalk input device) to the associated AI model (as an IoTtalk output device), which takes the sensing data as its input parameters to perform the prediction. Subsequently, the second IoTtalk NA updates the predicted actuating results of the AI model (as another IoTtalk input device) back to the ODG (as an IoTtalk output device), finishing a complete interaction of the AIoT application.

**V. DESIGN AND IMPLEMENTATION OF THE AIOTTALK SERVER**

As explained in Sec. IV-B, our AIoTtalk service model operates based on the close interactions among the AIoTtalk server, AI models, and the IoTtalk server. In particular, as illustrated in Fig. 3, the AIoTtalk server and AI models are treated as IoTtalk devices, referred to as “AIoTtalk device” (see Fig. 3 (I)) and “AI devices” (see Fig. 3 (II)), respectively, exploiting the IoTtalk server’s capability of “service switching” between IDFs/ODFs (see Fig. 3 (III)). In what follows, we will elaborate the detailed designs and implementations of the AIoTtalk device and the AI devices, respectively.

**A. Software Architecture of the AIoTtalk Device**

The AIoTtalk device implements its own SA and DA functionalities to, respectively, 1) offer an application environment to manage those deployed SIP-based IoT applications and their IDGs/ODGs, and 2) map the IDG/ODG of each application to its corresponding NA in the IoTtalk engine. Following such functional splitting, the AIoTtalk device is composed of two main software components: a single AIoTtalk SA and multiple IoTtalk DAs (one for each application). To achieve the assigned goals, the AIoTtalk SA, the core of the AIoTtalk device, further consists of four software modules: the AIoTtalk GUI (see Fig. 3 (A)), the AIoTtalk DB (see Fig. 3 (B)), the SIP msg handler (see Fig. 3 (C)), and the blockchain handler (see Fig. 3 (D)). On the other hand, each IoTtalk DA software module is allocated by the DA manager software module (see Fig. 3 (E)) to handle the communication between its associated SIP-based IoT application and the IoTtalk engine in the IoTtalk server, via HTTP-based RESTful APIs. Please note that blockchains are leveraged to function as shared, encrypted/immutable open databases in our AIoTtalk to implement identity authentication/data protection for our AIoTtalk’s security guarantee. Next, we will introduce the functionalities of these six modules, respectively.
1) *AIoTalk GUI and DB*: The web-based AIoTalk GUI, incorporating a user event handler, 1) handles the AIoTalk device’s own SIP account settings, and 2) allows application developers to create new IoT applications. For the SIP account settings, it will pass the configuration to the SIP msg handler to register the AIoTalk device’s SIP account to a SIP proxy. For an IoT application creation, it also allows the application developer to select the corresponding IDs/ODs within different S-UAs/A-UAs into an IDG/an ODG with the associated sensing data type/actuating result type to join the realization of the IoT application. Note that the service logic of the IoT application should be implemented as an NA in the IoTtalk server (see Sec. V-D). For this, the DA manager will be requested to allocate a DA to handle this newly created IoT application’s communication with its NA in IoTtalk. Afterwards, the account configuration and application information will be stored in the AIoTalk DB. In particular, the AIoTalk DB maintains a service table, which records the triplet (IDG, ODG, DA) for each SIP-based IoT application.

2) *DA Manager and IoTtalk DAs*: Whenever a SIP-based IoT application is developed via the AIoTalk GUI, the DA manager will be instructed to construct an IoTtalk DA for registering the application’s IDG/ODG to the IoTtalk engine, establishing connections between the IDG/ODG and their involved NA (see the description in the AIoTalk GUI) in the IoTtalk engine. After the registration, the IDG and ODG can participate in the IoT application deployed on the IoTtalk server.

3) *SIP Msg Handler*: The SIP msg handler copes with the SIP signaling for 1) the AIoTalk device’s registration to the serving SIP proxy as a SIP UA, in accordance with the SIP account configuration from the AIoTalk GUI, and 2) the messaging and streaming between the AIoTalk device and the S-UAs/A-UAs for both the scalar and multimedia IoT applications. Note that the SIP msg handler can pass the SIP messages to the blockchain handler for security-related manipulations (if activated). The operation of the SIP msg handler will be further discussed in Sec. V-B.

4) *Blockchain Handler*: The blockchain handler applies the blockchain and the smart contract mechanisms to support identity authentication and guarantee data confidentiality between the S-UAs/A-UAs and the AIoTalk device. In particular, we implement a hybrid blockchain architecture, integrating private and public blockchains, that allows privacy preservation and public accountability simultaneously. The operation of the blockchain handler will be further discussed in Sec. V-C.

B. Operation of SIP Msg Handler

To perform its main tasks, the SIP msg handler launches three threads: 1) the account thread registers the AIoTalk device to the serving SIP proxy, 2) the message thread deals with SIP messaging, and 3) the session thread handles SIP streaming. Note that we enhance the class definitions of the SIP SIMPLE client SDK [28] developed by AG Projects, including *AccountManager*, *SIPMessageApplication* and *SIPSessionApplication*, to implement these three threads as follows.

1) *Account Thread*: To register the AIoTalk device as a SIP UA, the account thread initially creates an object of *AccountManager* to manage the SIP account of the AIoTalk device. Then, whenever the account thread receives an account configuration request, it will store the new configuration and register the SIP account to the desired SIP proxy, following the SIP registration procedure in Fig. 4a.

2) *Message Thread*: Following the SIP messaging procedure in Fig. 4b, the message thread creates an object of *SIPMessageApplication* to handle SIP MESSAGEs from/to S-UAs/A-UAs. For the case of S-UAs, the received SIP MESSAGEs are first stored in the message queue, from which the message thread parses each MESSAGE and extracts the enclosed sensing data in the FIFO order. Afterwards, via the involved applications’ associated IoTtalk DAs (recorded in the service table in the AIoTalk DB), each sensing value will be transmitted to and used in the IoTtalk NAs to compute new actuating results. On the other hand, to periodically send a SIP MESSAGE to an A-UA, containing the freshest actuating results from the respective IoTtalk NAs (via the IoTtalk DAs), the process is similar, but in the reverse direction.

3) *Session Thread*: The session thread creates an object of *SIPSessionApplication* to handle the SIP streaming procedure from an S-UA to the AIoTalk device and from the AIoTalk device to an A-UA, which consists of three phases: 1) session establishment, 2) media transfer, and 3) session teardown. Fig. 4c gives the message flow for the former case. Specifically, the AIoTalk device exercises an INVITE state machine from a session’s creation to the session’s termination. Once a session is established from an S-UA to the AIoTalk device, a media stream will be connected between them. Next, the sensing media packets from the S-UA can be sequentially transmitted to the IoTtalk server, via the AIoTalk device, for the NAs’ operation. Conversely, the latest multimedia actuating results can also be transmitted to an A-UA following such a SIP session/media stream manipulation.

C. Operation of Blockchain Handler

Fig. 5 illustrates the proposed hybrid blockchain architecture between a network of S-UAs/A-UAs and the AIoTalk
device, integrating a private blockchain (see Fig. 5 (A)) with the public Ethereum blockchain (see Fig. 5 (B)). Here, the private Ethereum blockchain connects the S-UAs/A-UAs and a set of blockchain proxies (BC-proxies; see Fig. 5 (C)), while the public Ethereum blockchain connects the set of BC-proxies and the AIoTtalk device. In this architecture, the BC-proxies, in the proximity of the S-UAs/A-UAs, are developed to securely relay the SIP signaling/media packets between the S-UAs/A-UAs and the AIoTtalk device. Moreover, the BC-proxies jointly operate a token ring to prevent the single-point-of-failure (SPOF) problem, where the BC-proxy currently holding the token will be assigned to serve the S-UAs/A-UAs. On the other hand, the blockchain handler in the AIoTtalk device authenticates the S-UAs/A-UAs at the application level and immutably records the handled sensing data and actuating results.

The detailed operation of this hybrid blockchain architecture is further explained in terms of the following three aspects.

1) Smart Contracts: We design and deploy three smart contracts for the proposed hybrid blockchain architecture to enable secure communication and transactions’ public accountability. Specifically, 1) UAInfoStore (see Fig. 5 (D)), operated by the BC-proxies on the private blockchain, stores/logs each S-UA/A-UA’s information, including identity, virtual identity (used by diTKStore on Ethereum), processed request/response messages, and security key, 2) diTKStore (see Fig. 5 (E)), operated by the blockchain handler on Ethereum, stores/logs each S-UA/A-UA’s “de-identified” information, including virtual identity, processed transactions, and security key, and 3) PVCMgn (see Fig. 5 (F)) on Ethereum manages the security keys of the BC-proxies and the blockchain handler for the encryption operations over the private connection pre-established between them.

Note that for security operations, our architecture needs to establish a shared secret for each pair of communicating network entities. For this, we apply the Elliptic-curve Diffie–Hellman (ECDH) key agreement protocol, which is based on one’s own private key and the other’s public key. The relevant exchanged public keys are stored beforehand in the corresponding smart contracts (that is, the above-mentioned security keys).

2) Smart Contract Registration: Each S-UA/A-UA should first compute its ECDH public–private key pair and send a registration request enclosing the public key to the serving BC-proxy and the blockchain handler. When the serving BC-proxy receives the registration request, it retrieves the S-UA/A-UA’s identity and public key, assigns a virtual identity to this S-UA/A-UA, and stores such information together with this registration request itself into UAInfoStore. Then, the serving BC-proxy relays the registration request to the blockchain handler, where the S-UA/A-UA’s original identity is replaced by the mapped virtual identity for privacy preservation in the public Ethereum blockchain. In particular, the serving BC-proxy uses its own private key and the blockchain handler’s public key, pre-stored in PVCMgn, to compute a shared secret, which is in turn applied to encrypt the registration request utilizing the AES algorithm. On the other side, the blockchain handler decrypts the registration request, stores the S-UA/A-UA’s virtual identity, transaction information, and public key in diTKStore, and sends back a successful registration response to the S-UA/A-UA via the serving BC-proxy through symmetric security operations but in reverse order. Note that the public keys of the BC-proxies and the blockchain handler will be attached in the response back to the S-UA/A-UA.

3) Secure Packet Transmission: The SIP signaling/media packet transmissions between an S-UA/A-UA and the AIoTtalk device are securely handled by the serving BC-proxy and the blockchain handler (together with the three smart contracts) through the encrypted private connection between them, the same as the case of the registration request. In addition, authentication and packet integrity verification are also performed between the S-UA/A-UA and the network side. For example, to send a packet, the S-UA/A-UA should first use the BC-proxies’ and the blockchain handler’s public keys (together with its own private key) to compute the corresponding shared secrets, which are in turn applied to derive the Hash-based message authentication codes (HMACs). The two codes are attached in the request to the serving BC-proxy and the blockchain handler, which can likewise use their private-exchanged public key pairs to compute the shared secrets and then their versions of the HMACs for comparison, thus authenticating the S-UA/A-UA and verifying the packet’s integrity.

D. The Procedure for an IoTtalk Network Application Implementation and Deployment

In our AIoTtalk, the service logic of a SIP-based IoT application can be automatically created in the IoTtalk configuration in the AIoTtalk device. First, the IDF/ODF of the IoTtalk application can be readily identified and defined, corresponding to the IDG’s sensing data type and the ODG’s actuating result type, respectively, configured in the AIoTtalk device.

Step 1 “IDF/ODF identification”. First, the IDF and ODF of the IoTtalk application can be readily identified and defined, corresponding to the IDG’s sensing data type and the ODG’s actuating result type, respectively, configured in the AIoTtalk device.

Step 2 “IDF/ODF specification”. Via the GUI of the IoTtalk server, the NA with respect to the SIP-based IoT application can be automatically created in the IoTtalk engine. Specifically, the developer can manually add the identified IDF/ODF for selection by using the GUI to
express the expected functionalities, corresponding to the sensing data type/actuating result type of the IDG/ODG. Also, the developer should create an “input device” and an “output device” in the IoTtalk server with respect to the IDG and ODG created in the AIoTtalk device and associate them with the IDF and ODF. Then, the “input/output devices” together with their IDF/ODF can be selected to be included in the NA.

- **Step 3 “IDF/ODF mapping”**, Via the GUI of the IoTtalk server, a “join” connection (as a part of the NA hosted in the IoTtalk engine) can be established between the newly created IDF and ODF modules. With this “join” connection, the developer can specify (using the GUI) how a new value of the IDF module should be mapped to a new result of the ODF module.

- **Step 4 “DA association”**, Note that, in the AIoTtalk device, a “DA” is allocated to handle the communication between the IDG/ODG of the SIP-based IoT application and the IoTtalk engine using HTTP-based RESTful APIs. With a registration to IoTtalk, this DA can associate the IDG/ODG with the corresponding created “input/output devices” together with their IDF/ODF (see Step 2) in the IoTtalk engine, and can deal with the subsequent interactions between them.

VI. DESIGN AND IMPLEMENTATION OF AI DEVICES

A. Software Architecture of AI Devices

An AI device, which also follows the IoTtalk device structure, can accommodate and execute a specific AI model implemented and pre-trained by its developer to achieve the customized data prediction or conversion in a SIP-based IoT application. As shown in Fig. 3, within an AI device, the AI SA (see Fig. 3 (F)) manages and carries out the hosted AI model, which is saved in the hierarchical data format 5 (HDF5). Note that HDF5, a truly hierarchical, filesystem-like, portable scientific data format, allows to store an AI model’s large amount of data in a single file, including the model weights and configuration. Specifically, when prepared, the pre-trained AI model can then be loaded into the AI SA by using the load_model function of the keras.models package, enabling subsequent prediction operations. On the other hand, the IoTtalk DA (see Fig. 3 (G)) is responsible for acquiring the input parameters and updating the output results of the AI model from/to the corresponding IDG/ODG in the AIoTtalk device.

B. Two Demonstrative AI Devices

For demonstration purposes, we provide two built-in AI devices in our AIoTtalk service platform, each incorporating a deep-learning neural network model. The first one is a long short-term memory (LSTM) model for road speed prediction, while the second one is a convolutional neural network (CNN) model for audio recognition. These two AI models will be applied in two SIP-based IoT applications, whose details will be discussed in Sec. VII.

In the following, we briefly summarize the main ideas of the two AI models.

1) An LSTM Model for Road Speed Prediction: Broadly speaking, LSTM is a kind of recurrent neural networks (RNNs). However, different from the Simple-RNN, LSTM has a unique architecture so that the problem of vanishing and exploding gradients can be resolved. LSTM has been shown to be good at learning long-term dependency information, capable of predicting time series data.

As Fig. 6 illustrates, each LSTM cell at time step $t$ utilizes a forget gate, an input gate, and an output gate to calculate its memory cell $C_t$ and hidden state $h_t$, where the forget gate and the input gate are used to control how to forget the information from the previous time step and how to add in the new information, while the output gate is used to determine how the updated information of $C_t$ should be output.

First, in terms of the memory cell updating, $C_t$ is the weighted sum of $C_{t-1}$ and the candidate memory cell $\hat{C}_t$, which is derived based on the previous hidden state $h_{t-1}$ and the current data $x_t$. In particular, the LSTM cell uses the output $f_t$ of the forget gate and the output $i_t$ of the input gate as the respective weights, determining how much of $C_{t-1}$ and $\hat{C}_t$ should be kept to get the memory cell $C_t$ at time $t$. Next, for the hidden state calculation, the LSTM cell uses the output $o_t$ of the output gate to decide how much of the memory cell $C_t$ should be output as the current hidden state $h_t$.

We create an AI device in AIoTtalk that employs an LSTM model trained for road speed prediction. In this model, we use the twelve average speeds per minute in the past twelve minutes of a road segment to predict the average speed in the next minute on the road segment.

2) A CNN Model for Audio Recognition: CNNs have been commonly used in image and audio classification. As shown in Fig. 7, a CNN model consists of different layers: the convolution layers, the pooling layers, the fully connected layers, and the classification layer. Typically, prior to entering the CNN model, the data should be pre-processed, including cleaning, integration, reduction, and transformation. First, each convolution layer uses filters to perform convolution operations on the data to extract the features of the data, which are represented as the feature maps. Next, the corresponding pooling layer reduces the dimension and the noise of the feature maps while retaining their important information. Then, the fully connected layers will flatten the pooled feature maps into an
one-dimensional element array. Finally, the classification layer will pass the one-dimensional element array into an activation function to classify the input data to one of the possible outcome categories.

We create our second AI device in AIoTtalk that operates a CNN model trained by Salamon et al. [29], [30] for audio recognition. Specifically, the authors used the UrbanSound8K [31] as the dataset, which contains thousands of labeled urban sounds from ten classes, to train their CNN model, capable of analyzing ambient noises around urban environments. Moreover, this model also applied the data pre-processing method in [32] first to transform the audio clips into the segmented spectrograms, and describe how to feed them to this CNN model.

VII. AIO TTALK PLATFORM VALIDATION: REAL DEPLOYMENT OF TWO SIP-BASED IOT APPLICATIONS

This section validates the applicability of our AIoTtalk for heterogeneous IoT application developments by deploying two real SIP-based AIoT applications within AIoTtalk.

A. A Real Testbed Establishment

As illustrated in Fig. 8, we establish a real testbed consisting of several S-UAs/A-UAs, a SIP proxy (see Fig. 8 (A)), our AIoTtalk (including the AIoTtalk server (see Fig. 8 (B)) and AI models (see Fig. 8 (D))), and the IoTtalk server (see Fig. 8 (C)), where the SIP proxy is built using the well-referenced open-source SIP server software, Asterisk [33], for

\[ \text{https://github.com/wmnet741/AIoTtalk} \]

with the copyright protection.

**TABLE II: Computing equipment of different servers in the testbed**

<table>
<thead>
<tr>
<th>Computing Equipment</th>
<th>OS</th>
<th>CPU</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIP Proxy</td>
<td>Ubuntu 18.04.5 LTS</td>
<td>AMD Ryzen 5 2400G with Radeon Vega Graphics (x8)</td>
<td>16384 MB</td>
</tr>
<tr>
<td>IoTtalk Server</td>
<td>CentOS 7</td>
<td>Intel® Core™ i7-4790 CPU @ 3.60GHz (x8)</td>
<td>14336 MB</td>
</tr>
<tr>
<td>Edge Platform</td>
<td>Ubuntu 20.04.3 LTS</td>
<td>Intel® Xeon® Silver 4110 CPU @ 2.10GHz (x32)</td>
<td>131072 MB</td>
</tr>
</tbody>
</table>

Note that, our AIoTtalk is deployed in our proposed network function virtualization (NFV)-based multi-access edge computing (MEC) platform [34]. Moreover, in both scenarios, the SIP proxy, the edge platform, and the IoTtalk server are installed within physical machines whose computing equipment is shown in Table II.

More specifically, in this edge platform, following the ETSI NFV framework,

- **Cloud scenario.** In this case, the S-UAs/A-UAs are located in National Taiwan University, Taipei City, Taiwan, while the SIP proxy, the AIoTtalk service platform, and the IoTtalk server are located in National Tsing Hua University, Hsinchu City, Taiwan. The distance between the two universities is around 80.3 kilometers.

- **Edge scenario.** In this case, all the S-UAs/A-UAs, the SIP proxy, the AIoTtalk service platform, and the IoTtalk server are located in National Tsing Hua University, Hsinchu City, Taiwan.

SIP message/session routing. Furthermore, we consider the following two cases for our testbed setup:

- **Cloud scenario.** In this case, the S-UAs/A-UAs are located in National Taiwan University, Taipei City, Taiwan, while the SIP proxy, the AIoTtalk service platform, and the IoTtalk server are located in National Tsing Hua University, Hsinchu City, Taiwan. The distance between the two universities is around 80.3 kilometers.

- **Edge scenario.** In this case, all the S-UAs/A-UAs, the SIP proxy, the AIoTtalk service platform, and the IoTtalk server are located in National Tsing Hua University, Hsinchu City, Taiwan.

Note that, our AIoTtalk is deployed in our proposed network function virtualization (NFV)-based multi-access edge computing (MEC) platform [34]. Moreover, in both scenarios, the SIP proxy, the edge platform, and the IoTtalk server are installed within physical machines whose computing equipment is shown in Table II.

More specifically, in this edge platform, following the ETSI NFV framework,

- Open Source MANO (OSM) is exploited for the NFV management and orchestration (NFV MANO), which manages the resources and lifecycle;

- OpenStack is exploited for the NFV infrastructure (NFVI) and virtualized infrastructure manager (VIM), which provides the virtualized resources;

- the AIoTtalk service platform is packaged into a customized VM image via OpenStack, which is then formed as a VNF and deployed into the NFV environment by using the OSM VNF onboarding.

This edge platform supports auto scaling and load balancing for our AIoTtalk.

Next, to demonstrate the applicability of our AIoTtalk, in the following two subsections, we implement and deploy the two exemplar AIoT applications “real-time short-term road traffic prediction” and “neighborhood violence detection”, which are discussed in Sec. IV-B1 and exploit SIP messaging and streaming, respectively. Note that, for each involved S-UA/A-UA in our testbed, we actually launch a respective

\[ \text{https://github.com/wmnet741/AIoTtalk} \]

with the copyright protection.
network program that sends/receives the SIP MESSAGES or establishes the SIP SESSION as required.

B. Real-Time Short-Term Road Traffic Prediction: a Demonstrative AoT Application Exploiting SIP Messaging

1) Application Description: As shown in Fig. 8 (I), in this application, each RSU (see Fig. 8 (E)) in a city section periodically sends a SIP MESSAGE to report the sensed speed data of its monitored road segments, collected from the roadside vehicle speed sensors, to a Long Short-Term Memory (LSTM) AI model (see Sec. VI-B1) in the AloTalk service platform. This short-term LSTM AI model has been pre-trained to forecast the vehicle speed over each road segment of the city section in the next five minutes, based on the past and current speed data. Upon receipt of the sensed speed data, the LSTM AI model will estimate the desired speed values. Then, these short-term estimated speed values can be encapsulated into a SIP MESSAGE, a copy of which will be sent to each connected vehicle that will pass through the city section (see Fig. 8 (F)). Eventually, these speed values will be displayed on these vehicles’ electronic maps, allowing to improve the drivers’ journey efficiency.

Following the AloTalk service model, this AoT application is realized via two IoTalk NAs, NA1 and NA2: 1) NA1 relays the sensed speed data from the group of roadside vehicle speed sensors (as the input device) to the LSTM AI model (as the output device) for prediction, and 2) NA2 relays the predicted speed data from the LSTM AI model (as the input device) to the group of electronic maps (as the output device) of the passing vehicles for display. The operation of this application involves the joint collaboration among the AloTalk server, the LSTM AI model, and the IoTalk server. In the following, we explain the corresponding settings required to implement and deploy this AoT application within our AloTalk.

2) The AloTalk Server Settings: The AloTalk server interacts with both 1) the S-UAs/A-UAs and their IDs/ODs, and 2) the IoTtalk server, and thus needs the developer to conduct the following settings, so as to acquire the related application information:

- **Step 1.** First, an application, named “RoadTrafficPrediction”, should be created in the application management page.
- **Step 2.** Next, to interact with the S-UAs/A-UAs and their IDs/ODs: the ID type “RoadSensor” and OD type “ElecMap” for this application, together with their corresponding sensing data type “SensedSpeed-I” and actuating result type “PredictedSpeed-O”, should be defined in the ID/OD management page.
- **Step 3.** Finally, to interact with the IoTalk server: the IDG, named “RoadSensorGroup”, and the ODG, named “ElecMapGroup”, should be created. To define the application in the device group management page by selecting the involved “RoadSensor” IDs with the sensing data type “SensedSpeed-I” into the IDG, and also the involved “ElecMap” ODs with the actuating result type “PredictedSpeed-O” into the ODG.

Note that, after the above steps, the AloTalk server will create a DA for this application to register the IDG “RoadSensorGroup” and ODG “ElecMapGroup” to the IoTalk server as two IoTalk devices.

3) The AI Model Settings: The LSTM AI model participates in two IoTalk NAs, NA1 and NA2: in NA1, it acts as an output device to receive the sensed speed data from the input device, the IDG “RoadSensorGroup”; in NA2, it acts as an input device to transmit the predicted speed data to the output device, the ODG “ElecMapGroup”. To reflect these two roles of the LSTM AI model, in the AI model management page, the developer needs to conduct the following settings:

- **Step 1.** First, to interact with the IoTalk server: the developer should define the device type “AI_LSTM”, which is associated with the actuating result type “SensedSpeed-O” for the “output device” of NA1, and with the sensing data type “PredictedSpeed-I” for the “input device” of NA2.
- **Step 2.** Then, the developer should upload the pre-trained LSTM speed prediction model, which will receive data of type “SensedSpeed-O” as its model input, and compute values of type “PredictedSpeed-I” as its model output.

At last, a DA for the pre-trained LSTM speed prediction model will be automatically created to register this model as an IoTalk output device and an IoTalk input device for NA1 and NA2, respectively.

4) The IoTalk Server Settings: In the IoTalk server, the developer needs to establish the two NAs, NA1 and NA2, via the following settings:

- **Step 1.** First, for NA1, the developer should create an input IoTalk device “RoadSensorGroup” with IDF “SensedSpeed-I” and an output IoTalk device “AI_LSTM” with ODF “PredictedSpeed-O”; for NA2, the developer should create an input IoTalk device “AI_LSTM” with IDF “PredictedSpeed-I” and an output IoTalk device “ElecMapGroup” with ODF “PredictedSpeed-O”. Then, the created IoTalk devices should be selected into NA1 and NA2, respectively.
- **Step 2.** Next, for NA1, the IDF “SensedSpeed-I” should be mapped to the ODF “SensedSpeed-O” to generate a join connection “Join 1”, which will pass the sensed speed data from the IDG “RoadSensorGroup” to the LSTM AI model “AI_LSTM”; for NA2, the IDF “PredictedSpeed-I” should be mapped to the ODF “PredictedSpeed-O” to generate another join connection “Join 2”, which will update the short-term estimated speed values computed by the LSTM AI model “AI_LSTM” to the ODG “ElecMapGroup”.

Then, when the IoTalk engine receives the registration request from the DA of each defined input/output device in the AoTalk server or the AI model, it can finally associate the input/output device with its counterpart “input/output device” in the IoTalk server, finishing the construction of the NAs. The screenshot of the IoTalk server settings is shown in Fig. 9.
C. Neighborhood Violence Detection: a Demonstrative AIoT Application Exploiting SIP Streaming

1) Application Description: As shown in Fig. 8 (II), in this application, a smart home automation safety system (see Fig. 8 (G)) will continuously stream the data collected from its audio sensor to a CNN AI model (see Sec. VI-B2) in our AIoTtalk via a SIP SESSION. This CNN AI model has been pre-trained to recognize any suspicious sound (e.g., gunshot, scream, and siren). Then, a corresponding alert notification can be encapsulated into a SIP MESSAGE, a copy of which will be destined to the emergency service of each surrounding police department (see Fig. 8 (H)) of the neighborhood.

This AIoT application is implemented as two IoTtalk NAs, NA3 and NA4: 1) NA3 relays the audio sensor data from the group of smart home automation safety systems to the CNN AI model, and the IoTtalk server for this application. In the following, we will briefly summarize the corresponding settings of the AIoTtalk server, the CNN AI model, and the IoTtalk server for this application.

2) The AIoTtalk Server, AI Model, and IoTtalk Server Settings:

- **The AIoTtalk server settings.** The developer should define the IDG “AudioSensorGroup”, and the ODG “EmerServGroup” for the application “NeighborhoodViolenceDetection” by selecting the involved “AudioSensor” IDs with the sensing data type “Audio-I” into the IDG, and also the involved “EmerServ” ODs with the actuating result type “Notification-O” into the ODG.

- **The AI model settings.** The developer should create the device type “AI_CNN”, which is associated with the actuating result type “Audio-O” for the “output device” of NA3 and with the sensing data type “Notification-I” for the “input device” of NA4.

- **The IoTtalk server settings.** For NA3, a join connection “Join 3” is established between the input IoTtalk device “AudioSensorGroup” with IDF “Audio-I” and the output IoTtalk device “AI_CNN” with ODF “Audio-O”; for NA4, another join connection “Join 4” is established between the input IoTtalk device “AI_CNN” with IDF “Audio-I” and the output IoTtalk device “EmerServGroup” with ODF “Notification-O”.

VIII. PERFORMANCE EVALUATION

This section conducts experiments to investigate the effects of several input parameters on the AIoTtalk’s performance using the real testbed and applications developed in Sec. VII.

A. Experiment Settings

As illustrated in Fig.10, our experiments consider six relevant events that will occur during each operation cycle of our AIoTtalk.

- **The sensing part:** 1) at \( \tau_1 \), an S-UA sends a sensing SIP MESSAGE/media packet to the AIoTtalk server; 2) at \( \tau_2 \), the sensing SIP MESSAGE/media packet arrives at the AIoTtalk server; 3) at \( \tau_3 \), according to the sensing data, the AIoTtalk server calls an HTTP-based RESTful API to update the ID of each enclosed ID’s IDG in the IoTtalk server; 4) each IDF update will trigger the subsequent execution of the corresponding two NAs in the IoTtalk server and one AI model, which will eventually update the ODF result back to the mapped ODG in the AIoTtalk server at \( \tau_4 \).

- **The actuating part:** 5) at \( \tau_5 \), the AIoTtalk server sends an actuating SIP MESSAGE/media packet to each involved A-UA, reflecting its ODs’ latest actuator statuses; 6) finally, the actuating SIP MESSAGE/media packet is received by the A-UA at \( \tau_6 \).

In the experiments, \( N_a \) S-UAs interact with \( N_a \) A-UAs to operate each of the two demonstrative AIoT applications. For each sensing SIP MESSAGE/media packet transmission, we record the corresponding \( \tau_1 \sim \tau_6 \) time points to compute the following delay segments:

- \( t_{s,S} \) and \( t_{S,a} \): \( t_{s,S} = \tau_2 - \tau_1 \) and \( t_{S,a} = \tau_6 - \tau_5 \) are the sensing and actuating SIP MESSAGE/media packet latencies between the S-UA and the AIoTtalk server and between the AIoTtalk server and an involved A-UA, respectively.

- \( t_{s,I}^{(n)} \) and \( t_{S}^{(n)} \): \( t_{s,I}^{(n)} \) is the processing time consumed by the AIoTtalk server to extract the sensing data from the sensing SIP MESSAGE/media packet and update
the data to the IDF}s within the IoTtalk server, while $t_{S}^{(a)}$ is that consumed by the AIoTalk server to retrieve and pack relevant actuating results into an actuating SIP MESSAGE/media packet, for delivery to an involved A-UA.

- $t_{\text{tot}}$ and $t_{m}$: $t_{\text{tot}}$ is the total operation time for the sensing data and actuating result transmissions between the IoTtalk server and the AIoTalk server/the AI model (via the HTTP-based RESTful API calls), and for the two IoTtalk NA executions, while $t_{m}$ is the AI model processing time.

Note that in our experiments, we apply the Network Time Protocol (NTP) to synchronize the clocks of the involved network entities in the testbed. More specifically, we deploy a central NTP server and an NTP client on each entity of the S-UAs, the AIoTalk server, the IoTtalk server, the AI models, and the A-UAs, where these entities will periodically synchronize their local times with the local time of the central NTP server, allowing us to obtain accurate $\tau_{1} \sim \tau_{0}$ time points.

### B. Experiment Results

In the first three parts, we discuss the performance results for the case that our proposed hybrid blockchain architecture in Sec. V-C is not activated. Finally, the last part presents the results our security-guarantee mechanisms achieve for AIoTalk.

1) Delay Segment Measurement: We consider the cloud scenario, where an S-UA sends a single sensing SIP MESSAGE/media packet to trigger a AIoTalk operation cycle of the road traffic prediction/nearby violence detection application, inter-operating with an A-UA. Figs. 11a and 11b demonstrate the incurred delay segments within the two applications, respectively, in this experiment.

Fig. 11 shows that the $t_{S}^{(a)}$ delay of the first application’s sensing SIP MESSAGE (in Fig. 11a) is larger than that of the second application’s RTP media packet (in Fig. 11b). This is because the SIP MESSAGE is of length 555 bytes (with a message body 94 bytes for the sensing data), while the RTP media packet is of length 218 bytes (encoded with ITU-T G.711 PCMU). Moreover, the parsing times $t_{S}^{(a)}$ of both the sensing SIP MESSAGE and RTP media packet at the AIoTalk server are small and can be neglected. We note that the IoTtalk operation times $t_{\text{tot}}$ are significant, around 165 ms and 185 ms for the first and the second applications, respectively. The first application leads to a smaller $t_{\text{tot}}$ since it has a smaller sensing data, 94 bytes. On the other hand, the length of the AI model processing time $t_{m}$ highly depends on the application type. In our case, the complicated audio recognition in the second application can take a $t_{m}$ as large as 155 ms. The $t_{S}^{(a)}$ delay of each of the two applications is taken for the AIoTalk server to activate a process and encapsulate a corresponding actuating SIP MESSAGE, whose value is around 55 ms. Finally, the actuating SIP MESSAGEs of length 502 bytes (with a message body 40 bytes) and 685 bytes (with a message body 211 bytes) for the first and the second applications, respectively, and thus the $t_{S,a}$ delay of the first application is slightly smaller than that of the second application.

2) Performance of the Road Traffic Prediction Application:

Define the mean end-to-end delay $E[t_{E}]$ as the sum of the mean delay of the sensing part and the mean delay of the actuating part within a AIoTalk operation cycle:

\[
E[t_{E}] = \left( \frac{\sum_{j=1}^{N_{S}} [t_{S}^{(a)}(j) + t_{S}^{(s)}(j) + t_{\text{tot}}(j) + t_{m}(j)]}{N_{m}^{(a)}} \right) + \left( \frac{\sum_{j=1}^{N_{S}^{(a)}} [t_{S}^{(a)}(j) + t_{S}^{(s)}(j) + t_{m}(j)]}{N_{m}^{(a)}} \right),
\]

where $N_{S}^{(a)}$ and $N_{S}^{(s)}$ are the total numbers of transmitted sensing SIP MESSAGEs (which go through the sensing part) and encapsulated actuating SIP MESSAGEs (which go through the actuating part) during the operation of the road traffic prediction application, respectively.

Figs. 12a and 12b show the effects of the number $N_{S}$ of S-UAs and the number $N_{m}$ of A-UAs on the mean end-to-end delay $E[t_{E}]$ for the cloud and edge scenarios, respectively. The two figures present a trivial result that as $N_{S}$ and $N_{m}$ increase, the communication and computational loads increase,
and thus $E[t_E]$ increase accordingly. We note that the $E[t_E]$ performance is mainly dominated by the operation mechanism of the IoTTalk server, which requires an interval of 300 ms between two consecutive RESTful API calls to guarantee successful IDF updates. Moreover, the $E[t_E]$ values under the edge scenario are roughly 10 times smaller than the corresponding values under the cloud scenario. This results from the joint effects of the two facts: 1) the edge scenario allows a lower latency between the S-UAs/A-UAs and the AIoTtalk server, and 2) the edge scenario deploys the S-UAs/A-UAs within a less powerful AMD-based desktop PC (compared with a virtual machine on a Dell EMC PowerEdge R740 rack server in the cloud scenario), which can only support a limited amount of threads simultaneously.

Fig. 12c shows the error (accuracy) of our LSTM model, which, based on SUMO traffic simulator, uses the twelve average speeds per minute in the past twelve minutes of a road segment to predict the average speed in the next minute on the road segment. We perform 100 predictions, and Fig. 12c plots the error between each pair of the predicted result and the actual data. The figure indicates that the prediction error of our LSTM model can roughly be kept below 10% and the average prediction error is 4%. Fig. 12d shows the effects of the OSM auto scaling mechanism of our edge platform on $E[t_E]$ under the cloud scenario, where $N_e=5$. Specifically, we add 25 more S-UAs into the application per half an hour, and measure the corresponding resultant $E[t_E]$. In addition, we configure the OSM auto scaling mechanism to scale out another VM to host another copy of the AIoTtalk server when $N_e$ reaches 100. We note that when $N_e$ is increased to be 100, $E[t_E]$ is nearly 7 s before scaling out is activated. Thus, the OSM auto scaling mechanism can effectively improve the $E[t_E]$ performance to be nearly 5 s even when $N_e$ is as large as 100.

3) Performance of the Neighborhood Violence Detection Application: Similarly, we measure the mean end-to-end delay $E[t_E]$ for the neighborhood violence detection application, where a AIoTtalk operation cycle is triggered by an RTP media packet. Figs. 13a and 13b show the corresponding results for the cloud and edge scenarios, respectively. Likewise, the two figures demonstrate the close-proximity benefit under the edge scenario, and that $E[t_E]$ linearly increases as $N_e$ and $N_a$ increase due to the constraint of a 300 ms interval between two consecutive IoTtalk RESTful API calls to assure IDF updates. To evaluate the prediction performance of the applied CNN model for audio recognition, following [30], we input ten different sounds into the CNN model and collect the returned prediction, which are depicted in Fig. 13c. The results indicate that the CNN model has strong confidence on recognizing most of the recordings, including air conditioner, car horn, engine idling and siren, but has weak confidence when facing the sound of children playing. Moreover, we further compare with the deep neural network (DNN) and LSTM models, which are also well applied to audio recognition in the literature. Containing only the fully connected layers, without the convolution and pooling layers, the DNN model is known for its simple architecture, thus leading to less computation time yet acceptable prediction performance. On the other hand, the LSTM model has demonstrated its capability of extracting audio time-series features while requiring a moderate network size. We note in Fig. 13c that each of the three models has different performance for different sounds and the CNN model achieves the best accuracy in general.

At last, we measure the quality of experience of the sensing RTP voice sessions between the S-UAs and the AIoTtalk server in terms of the five-level mean opinion score (MOS): 5, 4, 3, 2, and 1, representing excellent, good, fair, poor, and bad, respectively. In this experiment, the RTP session between each S-UA and the AIoTtalk server is actually divided into two sub-sessions: one between the S-UA and the Asterisk SIP proxy, and the other between the Asterisk SIP proxy and the AIoTtalk server. In this case, we then use the network packet analyzer Wireshark, installed in the Asterisk SIP proxy, to capture the live packet data and conduct the RTP stream analysis upon the two sub-sessions. Specifically, we collect the mean jitters and the counts of lost packets, which, following [35], can be subsequently mapped to the corresponding MOS values. We consider 2 to 12 RTP sessions between the S-UAs and the AIoTtalk server, and depict the resultant MOS values in Fig. 13d. The statistical results indicate that our testbed can guarantee a packet loss ratio less than 1% and a mean jitter less than 2 ms for the numbers of sessions less than 12, thus achieving MOS 5 (i.e., excellent). However, when the number of sessions is equal to 12, we find that the packet loss ratio is increased up to 90%, and the MOS value is consequently degraded down to 1 (i.e., bad).

4) Performance of the Proposed Hybrid Blockchain Architecture: Based on the cloud scenario, we deploy a ring of three BC-proxies, collocated with the S-UAs/A-UAs in National Taiwan University, to exercise our security mechanisms together with the blockchain handler for the road traffic prediction application. Fig. 14a first lists the properties our architecture exhibits and the corresponding approaches.
Fig. 14 further shows the extra delay costs consumed by our security mechanisms, for the cases: $N_a$ is from 2 to 8, and $N_a$ is 25. The delays include: 1) $\sim150$ ms for the authentication/integrity verification at the serving BC-proxy and the blockchain handler, and 2) $\sim0.8$ ms for two pairs of encryption/decryption between the serving BC-proxy and the blockchain handler. Clearly, compared with the total length of $E[t_{pc}]$’s, the incurred delays are tolerable.

IX. CONCLUSIONS AND FUTURE WORK

In this paper, we have designed a SIP-based IoT service platform, AIoTtalk, which leverages and enhances our previously developed IoTtalk, by equipping it with the SIP signaling ability. We have proposed a SIP-based IoT service architecture, based on which we designed and implemented our AIoTtalk, which enables rapid development of heterogeneous (scalar or multimedia) IoT, in particular AIoT, applications. Moreover, we have established a real testbed under both the cloud and edge scenarios, and deployed two real SIP-based AIoT applications, “real-time short-term road traffic prediction” and “neighborhood violence detection”, within AIoTtalk to validate the applicability of our AIoTtalk. Finally, we have conducted experiments using the real testbed and the two real SIP-based AIoT applications to demonstrate the performance of our AIoTtalk. The experimental results have shown that, together with accurate model predictions and edge-virtualization auto scaling, AIoTtalk can guarantee low latency and quality of experience for messaging and streaming-based AIoT applications.

For future work, the emerging AI-powered next generation computing technologies [36], integrating cloud/fog/edge/serverless/quantum computing paradigms, can be exploited to guarantee the reliable and efficient operation of AIoTtalk. Accordingly, the AIoTtalk service platform can be further shaped into a three-layer hierarchical architecture: 1) the computing layer provides an AI-powered cloud/fog/edge integrated computing infrastructure for AIoTtalk; 2) the control layer exploits the SDN technology to support intelligent traffic control decisions by establishing a global view for AIoTtalk; 3) the service layer contains the AIoTtalk server to handle intelligent service-logic execution and data-related manipulation. Based on this architecture, we identify possible future directions for AIoTtalk.

- Computing layer: 1) service placement and migration mechanisms dynamically arrange AIoTtalk on a suitable cloud, fog, or edge computing environment according to the predicted application requirements and user mobility; 2) resource scaling mechanism automatically allocates adequate cloud, fog, or edge resources for AIoTtalk according to the short-term predicted traffic in real time.
- Control layer: 3) function decoupling moves the SIP network (proxies and registrars) and the AIoTtalk functionalities to a programmable SDN controller as SDN applications; 4) load balancing mechanism through the SDN controller operates at the levels of the SIP proxies and the AIoTtalk servers, respectively, for network scalability.
- Service layer: 5) federated learning paradigm enables the local parameter update/global model aggregation interaction between the central server and each involved end IoT device via two IoTtalk NAs; 6) serverless AI model development pattern supports AIoTtalk application developers to bring their own containers hosting their locally developed AI models, without experiencing execution environment change; 7) edge data preprocessing, taking advantage of edge servers’ proximity to the end sensors, screens unnecessary data locally before being sent to AIoTtalk, thereby reducing network stress; 8) quantum key distribution, combined with a message authentication scheme, allows AIoTtalk to set up cryptographic keys for resistance to the security threats from quantum computers; 9) interoperability gateways support multimedia SIP clients to interact with other traditional scalar HTTP/MQTT/CoAP clients, without having to modify the involved protocols.

ACKNOWLEDGMENT

We would like to extend our appreciation to the anonymous reviewers who have carefully reviewed the boring implementation details and offer many constructive comments to significantly improve the quality of this paper. We would also like to express our sincere appreciation to our mentor Prof. Yi-Bing Lin, National Yang Ming Chiao Tung University, Taiwan, who has not only made critical comments but also offered us his original ideas unselfishly. This paper is the outcome of his suggestion.

REFERENCES


