Software: Practice and Experience

WILEY

RESEARCH ARTICLE

Leveraging Fog Computing for Security-Aware Resource Allocation in Narrowband Internet of Things

Vamshi Sunku Mohan[1] | Sriram Sankaran[1] | Rajkumar Buyya[2] | Krishnashree Achuthan[1]

[1]Center for Cybersecurity Systems and Networks, Amrita Vishwa Vidyapeetham, Amritapuri, India | [2]The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia

Correspondence: Vamshi Sunku Mohan (vamshis@am.amrita.edu)

Received: 24 June 2024 | **Revised:** 7 October 2024 | **Accepted:** 4 November 2024

Funding: The authors received no specific funding for this work.

Keywords: deep learning | fog computing | narrowband internet of things (NB-IoT) | performance analysis | resource allocation

ABSTRACT

Narrowband Internet of Things (NB-IoT) is LPWAN operating using narrowband spectrum in IoTs, requiring low data rates and long battery life. Since NB-IoT does not support handover, the requirement to sustain network connectivity during mobility may result in fake base station connections, hence applications are limited to stationary use-cases. Researchers propose extending NB-IoT in mobile applications, owing to higher signal quality and battery life, despite DoS attacks due to low bandwidth. As NB-IoT is resource-constrained, resources must be allocated based on application with data processing offloaded to optimise performance. Cloud servers, being centralised and memory-intensive, may result in increased computational delay, lower throughput, and DoS attacks in NB-IoT. Hence, in this paper, we implement fog computing, a decentralised technology, providing scalability, reduced bandwidth, and enhanced privacy, to provide distributed processing. Additionally, we develop secure handover protocols for private and service-provider-controlled fog networks under normal and cell-splitting conditions to minimise fake base station attacks and provide seamless handover. We introduce reputation-based mechanisms to determine device integrity and differentiate faulty behaviour and attacks. Further, we implement real-time application-aware resource allocation and QoS-based load-balancing using deep learning to distribute data processing between devices, fog and cloud servers. We simulate and prototype protocols on iFogSim2 and Raspberry Pi 4. Security of the fog computing framework is validated against various attacks and formally verified using Scyther. Evaluation shows that our approach consumes 12% and 43.75% lower power and communication overhead and approximately 6 and 16 times lower execution time and memory compared with existing solutions, thus making our approach lightweight.

1 | Introduction

Narrowband Internet of Things (NB-IoT) is a Low Power Wide Area Network (LPWAN) radio technology standard developed by 3GPP [1] operating at 180 kHz to provide low latency, broader coverage, extended battery life, etc. NB-IoT operates on a single antenna and half-duplex communication channel capable of handling 32 channels each with a gain of 23 dBm [2], thus optimising the signalling cost and increasing channel capacity and spectrum efficiency. Various studies have been conducted to minimise its power consumption by optimizing Extended Discontinuous Reception (eDRX) and Power Saving Mode (PSM) [3]. These works propose to deploy NB-IoT in stationary applications such as smart metering, smoke detectors, health trackers [4]

etc., owing to its easy integrability with the existing cellular network, ability to send a small quantity of data at infrequent intervals of time and ability to communicate in areas with low signal strength [5].

Since NB-IoT does not support handover, devices deployed in mobile applications such as public bike-sharing Connected and Autonomous Vehicles (CAV) are required to manually choose the visited base station to maintain connectivity across cells [1]. This may lead to devices connecting to fake base stations, information theft, and device tracking. Additionally, being resource and bandwidth-constrained, NB-IoT requires higher power and time to process data and allocate channel resources suitable for specific applications. When offloaded to centralised cloud servers, these operations delay system processes as the cloud incurs higher latency since its speed depends on the connectivity of virtual machines [6]. Hence, NB-IoT remains unimplemented in mobile applications. Owing to NB-IoT's compatibility with the existing communication spectrum, long battery life, ON-demand services, and extended coverage, researchers envision its application in mobile IoT use cases. A decentralised architecture such as fog computing, which introduces an intermediary layer consisting of fog servers between the edge devices and the cloud, can overcome these limitations.

Fog computing offers a distributed decentralised architecture by extending the cloud's features to the edge of the device network [7]. Fog servers, having smaller storage and computation capacity, act as intermediaries between the device and the cloud. Since fog servers are close to devices, the system architecture reduces communication time and requires lower network bandwidth to transfer data. Further, if NB-IoT devices need excess resources such as memory, bandwidth, CPU processing power, etc., to process data or run complex operations in real-time, the fog computing framework allocates excess resources in the virtual memory space in the fog servers [8]. Hence, this technology reduces the processing time and computational overhead on the NB-IoT and the dependency on the cloud for data storage, thereby improving its performance.

Further, cloud storage requires the device to be always connected to the internet, which requires the devices to be always in '*Active*' state. NB-IoT requiring optimised ON-OFF periods have intermittent internet connectivity, which may result in unreliable data processing and communication when connected directly to the cloud. Since fog servers can operate without the internet [9], device data can be synchronized with the cloud when the internet connection is re-established, thus ensuring uninterrupted operation. Additionally, as the cloud is centralised, the number of device connections may be limited to ensure good performance and low response time. Fog architecture being decentralised overcomes this limitation by allowing multiple NB-IoT connections, thereby improving scalability [10]. In cases of low latency requirements for real-time applications such as smart cities and healthcare or where NB-IoT devices are required to make rapid decisions, fog computing reduces the round-trip time to the cloud, thus enabling faster data processing as it brings computing resources closer to the devices.

In this paper, we implement a fog computing architecture to connect NB-IoT devices, as shown in Figure 1, by placing the fog servers in the resource-adequate base stations. Our work can be broadly classified into three categories, as listed below.

- Application-aware resource allocation—Applications such as resource prediction, attack detection, etc., are simulated on iFogSim 2 and assigned among NB-IoT devices, fog and cloud servers based on memory requirements. Device resources, such as memory, CPU usage, etc., are assigned based on the device's application and updated in real-time based on device requirements.
- Authentication and handover protocols—Protocols are designed on Raspberry Pi 4 to authenticate devices, enable handover and eliminate the possibility of fake base station connections as NB-IoT devices travel across cells.
- Reputation-based mechanism—A reputation-based mechanism, implemented on the Raspberry Pi 4, differentiates between faulty devices and attackers and blacklists attackers from transmitting messages and re-authenticating as they enter visiting base stations.

The resource-constrained NB-IoT devices form the lowermost layer (Layer-0), followed by the fog servers (Layer-1) and the cloud (Layer-2). Each successive layer has increased processing power and memory compared to the previous layers. We design the fog architecture to be interoperable [11], that is, enabling easy movement of devices between fog environments controlled by independent fog servers and various service providers. Since the fog architecture includes fog servers deployed by independent entities, such as industries, to provide enhanced connectivity, they are not completely trustworthy [12].

Additionally, attackers may eavesdrop and impersonate devices to steal sensitive information. To eliminate these attacks, we introduce a reputation mechanism to track device and base station behaviour. If the reputation is less than the threshold value, the device or base station is blacklisted. To provide uninterrupted communication and handover when base stations are blacklisted, we propose an authentication mechanism to split the particular cell into microcells and enable devices to join the adjacent base stations at the earliest.

Further, we implement a group authentication mechanism to enable rapid verification of multiple devices situated in the range of independent fog servers and registered with these servers as their home base stations. Application-specific resource allocation required to determine the load distribution between the devices, fog nodes and cloud servers, improve Quality of Service (QoS), and reduce implementation cost is achieved by leveraging deep learning algorithms. To differentiate between normal system operation and degraded performance due to poor channel quality, we design base stations to monitor the packet delay. If the packet delay exceeds the mean of the previous delays recorded, the base station varies the bandwidth within the allocated QoS range for the specific device application. Further, we model the secret generation, device authentication and handover algorithms on iFogSim2 [13] and Raspberry Pi 4 [14] to study their performance.

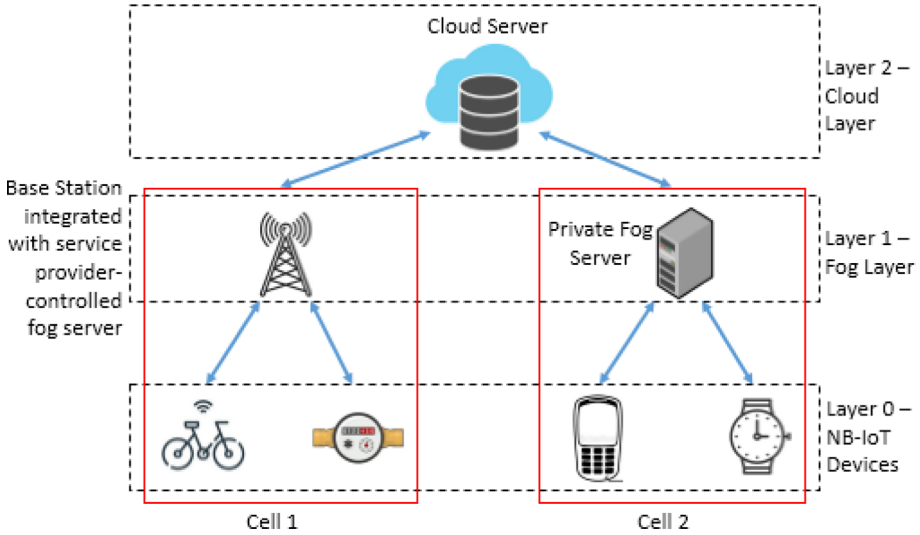

FIGURE 1 | Fog computing architecture.

Additionally, we model the device clustering mechanisms on iFogSim2 to define the cell limits of base stations and private fog servers and enable seamless device handover. We formally evaluate our approach using Scyther [15] and compute execution time, memory and energy-security trade-offs at each node. Further, we evaluate the NB-IoT channel performance in terms of throughput, packet latency and packet drop rate for varying authentication requests initiated in one '*Active*' state. We observe that our framework requires an average of 12% and 43.75% less power and communication overhead compared to existing solutions. Additionally, memory and execution time measured were 16 times and 6 times lower than the latter, resulting in an energy and performance-efficient fog computing framework.

1.1 | Primary Contributions

The primary contributions of our paper are listed as follows.

- We propose implementing fog servers in iFogSim 2 as an intermediary layer between the cloud and NB-IoT devices to reduce the communication time between the cloud and devices and minimise the data processing delay by analysing data even when devices are offline.
- We propose to implement lightweight protocols on Raspberry Pi 4 to achieve seamless handover, authentication and interoperability to enable device movement across cells and micro-cells controlled by service providers and independent fog servers.
- Based on application-specific resource requirements of devices, we propose to develop deep learning algorithms to predict the Classes of Service (CoS) [16], determine the load distribution [17] between the devices, fog layer and cloud and reallocate the device's Quality of Service (QoS) parameters.
- We propose a feedback-based mechanism to monitor packet delay and regulate the bandwidth to ensure bandwidth is within the permissible QoS limits across varying CoS.
- We propose a reputation-based mechanism to track the behaviour of the devices and base stations and blacklist attackers.
- Security of the fog computing architecture is validated for various attacks and formally verified using Scyther.

The rest of the paper is organized as follows. Section 2 discusses the Related Work. Section 3 explains the components of the proposed framework, such as authentication and handover algorithms, Unique ID and authentication secret generation, reputation update, etc. Section 4 presents security analysis of the proposed framework against known attacks. Section 5 discusses the experimental prototype implemented on iFogSim 2 and Raspberry Pi 4. Section 6 evaluates the framework's performance and provides a formal validation of protocols with Scyther. Section 7 discusses inherent security and regulatory issues in fog computing. Section 8 concludes the paper and describes possible future work.

2 | Related Work

Few works focus on developing a fog computing framework to foster handover in NB-IoT. A detailed study showed that a majority of them [18–27] provide support for scalability, whereas only a few of them [16, 21, 22, 24, 28–30] propose application-specific resource allocation. However, none support NB-IoT mobility, behaviour tracking and the elimination of fake devices and base stations. We categorise existing works as follows.

2.1 | Application of Fog Computing in NB-Iot

Ungurean and Gaitan [18] propose a fog architecture for real-time IIoT applications. The architecture is designed to receive data from the devices through field buses to process and share it to the cloud through a middleware system designed based on DDS standards. Peruzzi and Pozzebon [19] propose a hybrid architecture connecting LoRaWAN and NB-IoT nodes to fog gateways

through LoRa and MQTT, respectively. Sensor values thus collected are transmitted to the cloud using the LTE protocol. However, the authors neither discuss the implementation of the architecture for mobile applications nor the effect of its performance on devices' battery life. Hadi and Fadi [31], Qin et al. [32] and Prakash et al. [33] provide an extensive survey of fog computing in various applications such as smart grid, healthcare, agriculture, etc., propose protocols implementable in each of the cases and discuss prospective research directions. Jia et al. [28] propose an NB-IoT-based smart street light operating using fog computing to connect and share information such as light brightness, abnormal operation and requirements for maintenance. Abedin et al. [20] develop a game theoretic approach to model load balancing, resource over-utilisation and optimally schedule packet transmission.

2.2 | Resource Allocation Using Fog Computing

Guevara et al. [16] propose a method to classify the device operations into various modes of operation based on quality of service (QoS) parameters such as bandwidth, security level, data storage capacity, etc. The authors further develop a dataset for various combinations of QoS parameters and use machine learning and deep learning algorithms to classify the device operation efficiently. Gia et al. [34] propose connecting the LoRa devices in remote areas using fog architecture to transmit image data of crops. The authors develop a CNN-based image compression technique to reduce the amount of device data transmitted to the cloud. Zhao, Zou and Boshkani [29] propose a fog architecture based on Open-source Development Model Algorithm (ODMA) to efficiently allocate QoS such as service cost, energy consumption, response time, etc., to achieve resource optimisation. Murtaza et al. [21] develop a feedback-based resource allocation technique for devices connected using a fog architecture. The work aims to minimise the end-to-end delay, processing time and power consumed by continuously monitoring the device's behaviour. A similar approach has been presented by Wang et al. [35], where deep reinforcement learning is used to optimise IoT response time and re-distribute the load in fog servers. Apat et al. [11], Joao et al. [36] and Kashani et al. [37] present a survey of QoS parameters and evaluation tools required to achieve efficient resource allocation in fog architecture. Further, they present research challenges in implementing interoperability, scalability, etc. Tuli et al. [38] and Iftikhar et al. [39] review various AI-based methods to study the device QoS connected using fog servers, study the integration of AI in multiple applications such as healthcare, smart homes etc., and propose the latest trends and challenges in this direction. Lu et al. [40] propose a heuristic approach of allocating resources to the fog servers by successively skipping nodes that do not need excess resources and providing them to nodes based on their impact in the framework. Taghizadeh et al. [41] propose a sorting genetic algorithm to optimize resource usage by automatically deploying data replicas in the fog environment to ensure efficient data management and improved performance. Khan et al. [42] propose a similar approach where the cache stores and retrieves frequently allocated device QoS based on the application instead of repeatedly computing them in real time to reduce excessive resource consumption.

Existing works do not integrate fog computing with NB-IoT to enable handover in mobile applications. They do not address security issues in NB-IoT, such as fake base station connection, repeated packet retransmission and impersonation, replay attacks, differentiate system faults from attacks and detect the change in device behaviour for varying CoS and degraded channel quality. Further, some works do not propose application-specific resource allocation for ensuring longer battery life and authentication protocols to securely generate device IDs and authenticate the devices while moving across cells controlled by varying service providers.

2.3 | Our Proposed Method

In contrast to the existing approaches, we propose a fog computing architecture to implement seamless handover in mobile NB-IoT applications while ensuring scalability, interoperability, device-specific resource allocation and behavior tracking. We introduce a feedback-based mechanism to continuously monitor packet delay, vary bandwidth within the device's QoS limits and improve channel quality. Authentication protocols are proposed to enable seamless handover between cells and microcells controlled by various service providers. CoS of the device is determined based on its application and QoS is allocated according to the limits set by CoS. If the device requests excess resources, its CoS is updated and virtual hardware is allocated at the fog servers' memory. A reputation-based mechanism is introduced to track the behaviour of the fog servers and devices and blacklist them if reputation reduces below the threshold. We analyse energy-performance-security trade-offs of the handover and authentication protocols and formally evaluate them using Scyther. Performance is measured in terms of power, memory, execution time and computational complexity to show that our approach is energy efficient and secure. A detailed comparison of our approach with some of the existing works with respect to primary contributions of our paper, such as memory optimisation, support for scalability, resource allocation, etc., are given in Table 1.

3 | Proposed Framework

In this section, we describe the fog computing framework enabling resource allocation and handover in mobile NB-IoT applications shown in Figure 2. The framework is designed to contain the following entities.

a. *NB-IoT device*: Resource-constrained devices are designed only to store the permanent address, Unique ID and *Authentication secrets* required to prove the integrity and non-repudiation of the messages transmitted. Devices are designed only to send data to the fog servers upon authentication and receive QoS recommendations from the fog servers to maintain channel quality but do not respond to optimise battery requirements. In this work, we assume that when the device reaches the edge of the cell, the connection requests for handover from visited base stations in the form of visited base station address are made available. The devices choose the required address and send the handover request.

TABLE 1 | Comparison of existing works with our approach (em-dash is written in place of parameters not investigated by authors).

Proposed approach	Detection of rogue nodes/base	Performance optimisation scheme	Lightweight architecture support	Device security	Support for scalability	Support for mobility	Resource allocation	Support for interoperability
Guevara et al. [16]	No	Yes	—	No	No	No	Yes	No
Ungurean and Gaitan [18]	No	Yes	—	No	Yes	No	No	No
Peruzzi and Pozzebon [19]	No	No	—	No	Yes	No	No	No
Jia et al. [28]	No	No	—	No	Yes	No	Yes	No
Abedin et al. [20]	No	Yes	—	No	Yes	No	No	No
Zhao, Zou and Boshkani [29]	No	Yes	—	No	No	No	Yes	No
Murtaza et al. [21]	No	Yes	—	No	Yes	No	Yes	Yes
Goudarzi et al. [22]	No	Yes	—	No	Yes	No	Yes	No
Amanlou et al. [23]	No	No	Yes	Yes	Yes	No	No	No
Ali et al. [43]	No	Yes	Yes	Yes	No	Yes	No	No
Cui et al. [24]	No	No	Yes	Yes	Yes	No	Yes	No
Kalaria et al. [44]	No	Yes	Yes	Yes	No	Yes	No	No
Guo et al. [25]	No	Yes	Yes	Yes	Yes	Yes	No	No
Al-Mekhlafi et al. [26]	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Mohammed et al. [45]	No	Yes	No	Yes	No	Yes	No	No
Mohammed et al. [46]	Yes	Yes	Yes	Yes	No	Yes	No	No
Singh et al. [27]	No	Yes	Yes	Yes	Yes	Yes	No	No
Pallavi et al. [30]	No	Yes	Yes	No	No	Yes	Yes	No
Our Work	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

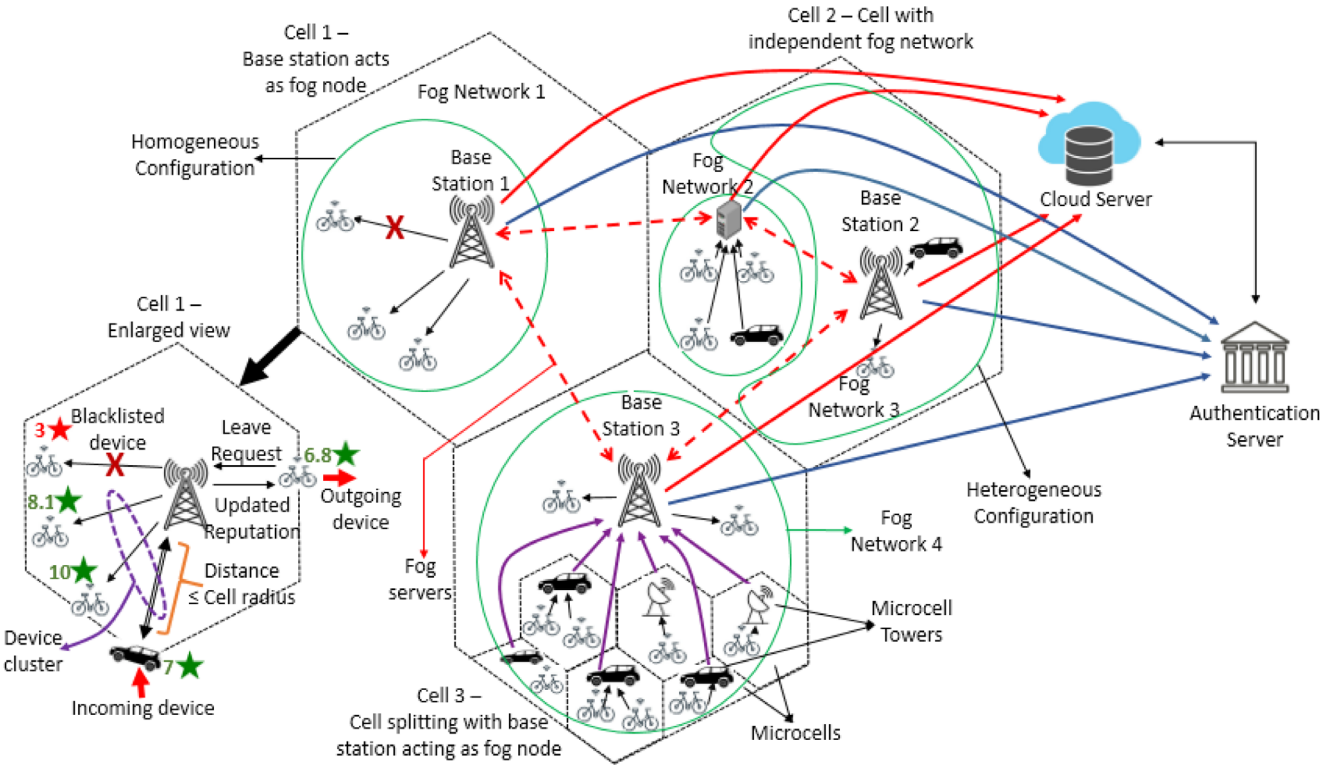

FIGURE 2 | Proposed fog computing framework.

b. *Fog Server*: Fog servers are installed at the base station of every cell. They may be either controlled by the internet service providers or may be established independently by private entities for better connectivity at their premises. Fog servers collect the device QoS in real-time and analyse it using deep learning to re-allocate system resources, regulate channel quality and update CoS based on the device's applications. They enable authentication and handover

protocols by communicating between the device and the cloud. To avoid data sharing with fake base stations, fog servers are designed to only upload device QoS to the cloud and not share among themselves. However, they are allowed to download QoS from the cloud for accurate analysis of device behaviour.

c. *Microcell Tower*: Microcell towers comprise microcell antennae and NB-IoT devices with higher-than-average resources. They are designed to relay device communications to the visited base stations during device handover in microcells. When base stations are offline, communications handled by the microcell towers are stored locally. When the base stations are back online, communications are synchronised, records are updated and deleted locally at microcell towers.

d. *Authentication server*: Authentication server handles a diverse amount of requests, such as handover, authentication and secret generation, at different points in time. Further, Authentication server verifies the details of devices initiating requests and generates and shares the *Authentication secrets* with the devices required for verification during authentication and handover mechanisms.

e. *Cloud Server*: Cloud server forms the topmost layer of the fog architecture and stores the details of all the devices and fog servers. When the device moves across cells, it generates a *Cloud Secret* using the *Seed* shared by the Authentication server. The cloud server then transmits them for verification to the visited base station. This design ensures that the device has been authenticated by the Cloud server and the Authentication server, thus avoiding impersonation attacks and session hijacking. In cases where the device's QoS varies continuously, the cloud server analyses the QoS with the device's previous performance using deep learning and blacklists the device if attack behaviour is observed.

Detailed working of the framework is described below.

3.1 | Fog Architecture

In this paper, we design a fog architecture to connect devices with fog servers and the cloud to enable mobility in NB-IoT. The devices are connected to the fog servers present in those particular cells. Fog servers, in turn, are connected to the Cloud servers and Authentication servers to enable authentication, handover, secret generation and storage of device data. Fog servers monitor packet delay in real-time and send feedback to devices to regulate bandwidth [47] by improving channel quality using Equations (1) and (2). This design eliminates the possibility of channel congestion caused by noisy signals or increased packet flow due to a higher number of devices and authentication requests. Hence, the feedback mechanism differentiates between natural causes and attacks such as DoS and jamming attacks, leading to poor channel quality. Fog servers assign the device with CoS based on its application and allocate the required QoS as discussed by [16]. If devices request a change in QoS requirements or an update of CoS, the fog servers use deep learning algorithms such as MLP, LSTM, autoencoders, etc., to predict the appropriate device QoS based on the requirements.

When the device QoS requirements vary continuously, the fog servers send the QoS parameters to the cloud server for verification. The cloud server retrieves the device's transaction data uploaded by the previous base station it traversed and uses deep learning to study the QoS variation behaviour. If the behaviour patterns repeat at specific intervals, the cloud server deduces it as an internal fault. The other cloud server classifies the behaviour as an attack and blacklists the device. This design enables uniform load distribution across devices, fog and cloud servers.

$$T_{\text{Transmit}}\ (s) = \frac{\text{Packet Size}\ (b)}{\text{Throughput}\ (b/s)} \tag{1}$$

$$\text{Throughput}\ (b/s) = BW\ (\text{Hz})\ *\ \log_2(1 + \text{SINR}) \tag{2}$$

where, T_{Transmit} = Transmission time, BW = Bandwidth.

As seen in '*Cell 2*' of Figure 2, the fog layer comprises servers deployed by various service providers and private entities to provide better connectivity in their location. Since service providers are government-registered, in this paper, we assume that the fog and cloud servers they deploy are trusted. However, private fog servers deployed by anonymous entities may not be entirely trusted due to the possibility of fake base station connections [1]. Therefore, we design the fog servers to transfer the device's QoS details and transactions to the cloud instead of the visited base station when the device moves across cells.

To accommodate more devices in a base station and to enable channel reuse [48], the cell is split into multiple microcells. In such cases, devices are either connected to the microcell antennae or to the devices having higher resources to act as a relay during handover between microcells and cells and microcells. Since microcell antennae have much lower memory and data processing capabilities than base stations and are required to operate only during cell splitting, they are designed to update the base station of the communications handled but not store QoS and device information.

3.2 | Application Model

In this subsection, we describe the application model of our fog architecture implemented on iFogSim2, given in Figure 3. The application model defines the execution tasks of the fog computing entities, such as data processing, analysis, etc., maps and schedules the tasks, allocates resources to optimise performance, and simulates and evaluates the fog computing framework. The model defines the workflow, mimics real-world scenarios and validates the feasibility of the application. We classify the application model of our proposed architecture into 3 phases based on the execution tasks designated to the NB-IoT device, fog server and cloud server.

The NB-IoT device contains the *Client module*, which forms the device hardware. It is responsible for processing the sensor input into packets, transmitting them to the fog server, receiving the fog server's QoS recommendations, and varying the actuators' device parameters. The *Main module* forming the hardware of the fog server transmits and receives data from the device and the cloud, respectively. Device performance under standard NB-IoT

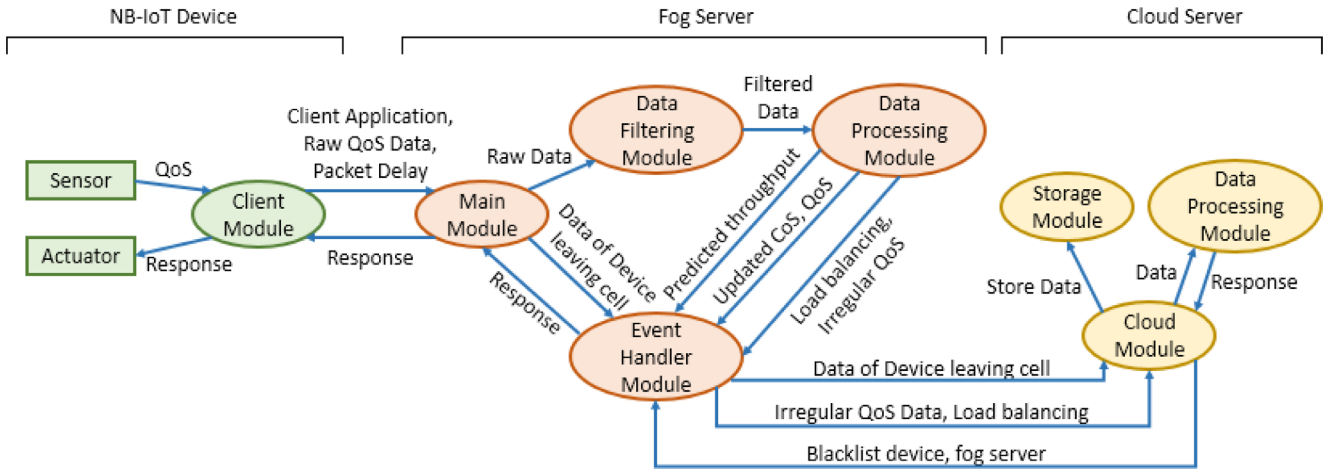

FIGURE 3 | Fog computing application model.

TABLE 2 | NB-IoT device specifications.

Parameters	Value	Parameters	Value
Bandwidth	180 kHz	Battery specifications	1 000 mAh, 3.6 V [49]
Uplink power	36 W	Downlink power	24 W
Uplink gain	70	Downlink gain	40
Uplink current	21.7μA	Transmission mode	Frequency-division duplexing
Packet size–uplink	2–125 B [50]	Packet Size-Downlink	2–85 B
Carrier frequency range	1 920–1 980 MHz [51]	—	—

specifications listed in Table 2 is measured offline for varying CoS and stored as a database in the *Data processing module*. This database acts as the training vector for deep learning models in the *Data processing module*. Real-time sensor data received from the devices is preprocessed to remove null and zero values and split into multiple datasets based on CoS. These datasets, acting as the testing vector, are inputted to the deep learning algorithms. The *Data processing module's* output predicted from real-time sensor input in terms of updated CoS, QoS and load distribution strategy is sent to the *Event handler module*, which interacts with the device and the cloud module. The cloud module stores the data of devices leaving the cells. Further, it analyses the irregular QoS of devices and base stations using deep learning and blacklists them if attack patterns are detected; otherwise, the behaviour is classified as a fault.

3.3 | Assignment of Classes of Service (Cos)

In this subsection, we describe the assignment of CoS and QoS to the devices based on their application as discussed by Guevera et al. [16]. NB-IoT's performance, that is, power, memory and execution time, depends on device operations such as bandwidth requirement, memory usage efficiency, location in the cell, packet delay and data processing capacity. Hence, we select the application requirements discussed in [16] such as *Security*, *Reliability*, *Data Storage*, *Data Location* and *Delay Sensitivity* as QoS for our study representing the device operations. We correlate *Reliability* to memory usage efficiency, *Data Storage* to data processing capacity, *Data Location* to location in cell, *Delay Sensitivity* to packet delay and *Security* to bandwidth requirement depending on handover procedure complexity as explained in Table 3. Other features such as *Mobility*, *Scalability* and *Loss Sensitivity* are not provided by default in NB-IoT [1]. Hence, they have not been considered.

Further, out of all the CoS categories mentioned in [16], we only consider the following cases.

- *Mission-critical (MC)*—Requires highest security and has to be executed at the earliest, for example, blacklisting a device
- *Real-time (RT)*—Requires real-time authentication and verification at the server, for example, handover
- *Interactive (IN)*—Requires the server to respond to the device queries, for example, tuning bandwidth to maintain channel quality
- *CPU-Bound (CB)*—Requires to analyse data and make decisions, for example, resource allocation and updation of CoS
- *Best-Effort (BE)*—Requires the device to have longer responsive delays and is concerned with only securely transmitting messages, for example, normal NB-IoT operation.

Other CoS categories, such as *Conversational (CO)* and *Streaming (ST)* requiring VoIP and video streaming support, are not provided by NB-IoT [1]. Hence, we have not considered them. To study the NB-IoT behaviour in fog computing, we consider the '*Cloud-Fog Computing Dataset*' [52] available on Kaggle. The

TABLE 3 | Correlation between quality of service (QoS), device requirements and dataset features.

QoS	Device requirements	Dataset features	Correlation
Security	Handover complexity	Bandwidth usage cost	Higher security resulting in increased complexity of handshake protocols, rises bandwidth usage cost
Reliability	Memory usage efficiency	Memory usage cost	Reliability defining the memory usage efficiency to store and process QoS is directly proportional to the memory usage cost
Data Storage	Data processing capacity	CPU usage cost	Data storage representing data processing capacity is proportional to CPU Usage Cost
Data Location	Location in cell	Execution time	Devices, fog servers and cloud servers require successively increasing time to collect, process and retrieve stored data from sensors, devices and fog servers respectively
Delay Sensitivity	Packet delay	Execution cost	Delay-sensitive CoS, such as *MC* and *RT*, requiring tasks to be executed in real-time significantly raises execution cost

a

	Number of instructions (109 instructions)	Memory required (MB)	Input file size (MB)	Output file size (MB)
Task_1	40	114	71	28
Task_2	21	57	44	25
Task_3	45	81	64	10
Task_4	71	90	78	99
Task_5	92	58	60	20
Task_6	36	107	90	31

b

	CPU rate (MIPS)	CPU usage cost	Memory usage cost	Bandwidth usage cost
Node_1	3754	0.8044	0.02088	0.0571
Node_2	3585	0.9433	0.0319	0.0806
Node_3	4617	0.8208	0.04455	0.0563
Node_4	1097	0.2464	0.02723	0.0105
Node_5	873	0.136	0.01474	0.018
Node_6	1054	0.1142	0.01232	0.0183

c

	Task_1	Task_2	Task_3	Task_4	Task_5	Task_6
Node_1	10.6553	5.594	11.9872	18.9132	24.5072	9.5898
Node_2	11.1576	5.8577	12.5523	19.8047	25.6625	10.0418
Node_3	8.6636	4.5484	9.7466	15.378	19.9264	7.7973
Node_4	36.4631	19.1431	41.021	64.722	83.8651	32.8168
Node_5	45.819	24.055	51.5464	81.3288	105.3837	41.2371
Node_6	37.9507	19.9241	42.6945	67.3624	87.2865	34.1556

d

	Task_1	Task_2	Task_3	Task_4	Task_5	Task_6
Node_1	16.6043	9.6299	15.5592	27.1997	25.4926	16.8573
Node_2	22.141	12.9053	20.3889	35.819	32.5056	22.6383
Node_3	17.7635	10.1574	15.7748	26.5969	23.4435	17.9792
Node_4	13.1282	6.9935	13.0902	20.2567	23.0837	12.2702
Node_5	9.6937	5.3537	9.5363	15.5733	16.6271	9.3634
Node_6	7.5501	4.2403	7.2278	12.0407	12.1467	7.4331

FIGURE 4 | Fog computing datasets (a) Task memory, (b) CPU and memory cost, (c) Execution time, (d) Execution cost.

dataset contains 1 120 samples of bandwidth, file size, memory required, and execution time for various tasks executed by the fog nodes. A screenshot of the *Task memory*, *CPU and Memory cost*, *Execution time*, and *Execution cost* datasets is shown in Figure 4.

As NB-IoT is resource-constrained, the device specifications at which it operates, as shown in Table 2, are taken as the *Medium range*. Under normal device operation, due to minor power fluctuations or varying hardware designs, device QoS may vary above or below the average specified QoS. To account for these design considerations, the medium range is defined to be between 25% to 75% of the maximum QoS. Values lower and higher than this are taken as *Low range* and *High range*, respectively, and QoS is set as shown in Table 4.

3.4 | Resource Allocation and Memory Optimisation Using Deep Learning

In this subsection, we use deep learning algorithms at the cloud server to study the device QoS in real-time by comparing them with the standard QoS limits and predict the device QoS requirements. In particular, we use LSTM, MLP, CNN, Simple autoencoder and Multilayer autoencoder algorithms built using the tensor layers as shown in Tables 5, 6, 7, 8 and 9 respectively.

Among the various deep learning algorithms, LSTM and MLP use backward propagation to iteratively calculate the weights of hidden layers. This optimises the biases through the gradient descent method and improves prediction accuracy in QoS following time-series patterns. Additionally, as LSTM uses the Forget gate to delete the data features with significantly low weights, it eliminates the vanishing or exploding gradient problem. Attackers, internal faults and user requests for resource updates in devices may result in non-linear QoS. Simple and Multilayer autoencoders using non-linear activation functions and dimensionality reduction help capture outliers due to faults and recognise attack patterns. Further, CNN requires fewer hidden layer determinant computations due to weight sharing and analyses complex QoS data using dimensionality reduction. This reduces overfitting caused by internal faults, enables scalability and improves prediction accuracy.

TABLE 4 | Class of service (CoS) range.

Parameters	Range	Value	Parameters	Range	Value
Reliability	Low	< 25% of Max(reliability)	Security	Low	< 25% of Max(security)
	Important	> 25% & < 75% of Max(reliability)		Medium	> 25% & < 75% of Max(security)
	Critical	> 75% of Max(reliability)		High	> 75% of Max(security)
Data storage	Transient	< 25% of Max(data storage)	Data location	Local	< 25% of Max(data location)
	Short duration	> 25% & < 75% of Max(data storage)		Vicinity	> 25% & < 75% of Max(data location)
	Long duration	> 75% of Max(data storage)		Remote	> 75% of Max(data location)
Delay sensitivity	—	Yes	—	—	—
		No			

TABLE 5 | Tensor layers of LSTM.

Parameters	Values
Input size	1
Sizes of hidden layers	60
Number of LSTM layers	9
Learning rate	0.001
Number of epochs	100
Loss function	Mean square error (MSE)
Optimiser	Adam

TABLE 6 | Tensor layers of MLP.

Tensor layer	Values
Hidden layer sizes	Input shape = (5, 11, 5)
Alpha	0.001
Solver	Stochastic gradient descent (SGD)
Shuffle	True
Random state	5
Verbose	True
Initial learning rate	0.00045

Performance of the devices in terms of QoS operating under NB-IoT specification limits, that is, *Medium QoS range*, are measured before deployment, split into several databases based on CoS, and stored as training vectors for the deep learning algorithms. Real-time QoS input by the sensor is transmitted to the fog server. It is pre-processed, converted into a testing vector, and inputted to deep learning algorithms to predict the corresponding CoS (CoS_{Pred}), detect high packet delays, and count the number of QoS variations (N) observed above T_{max} (maximum QoS limit) and below T_{min} (minimum QoS limit) for each of the CoS ranges as described in Table 4. The fog server then checks for the following cases.

- Case 1: If packet delay exceeds the average of previous packet delays, the fog server sends feedback to the device to vary SINR and improve throughput and bandwidth. This mechanism is introduced to eliminate the chances of poor channel quality caused by low SINR and congested channels.
- Case 2: If CoS_{Pred} is the same as device CoS, device QoS is less than T_{min} and N is significantly less (we consider three QoS variations as the limit), fog servers infer that devices do not have the required QoS to transmit data efficiently. Hence, fog servers send feedback to the devices to increase QoS to T_{min}.

TABLE 7 | Tensor layers of CNN.

Tensor layer	Values
1^{st} Conv2D	Size = 1, Kernel size = (1, 1), Activation = Linear, Input shape = (1, 1, 1), Padding = 'same'
1^{st} LeakyReLU	Alpha = 0.1
MaxPooling2D	Kernel size = (2, 2), Padding = 'same'
2^{nd} Conv2D	Size = 10, Kernel size = (1, 1), Activation = Linear, Padding = 'same'
Reshape	Input shape = (10, 1, 1)
3^{rd} Conv2D	Size = 5, Kernel size = (3, 3), Activation = Linear, Padding = 'same'
Flatten	—
2^{nd} LeakyReLU	Alpha=0.1
Dense	Size = 1, Activation = Sigmoid
Loss	Mean absolute error (MAE)
Optimiser	Adam

TABLE 8 | Tensor layers of simple autoencoder.

Tensor layer	Values
1^{st} Dense	Size = 1, Activation = tanh, Input Shape = (1, 1)
Dropout	0.2
2^{nd} Dense	Size = 1, Activation = Sigmoid
Flatten	—
Loss	Mean absolute error (MAE)
Optimiser	Adam

TABLE 9 | Tensor layers of multilayer autoencoder.

Tensor layer	Values
1^{st} Dense	Size = 11, Activation = Relu, Input shape = (1, 1)
2^{nd} Dense	Size = 9, Activation = Relu
3^{rd} Dense	Size = 11, Activation = Relu
4^{th} Dense	Size = 13, Activation = Relu
5^{th} Dense	Size = 1, Activation = Sigmoid
Flatten	—
Loss	Mean absolute error (MAE)
Optimiser	Adam

- Case 3: In situations when CoS_{Pred} is the same as device CoS, QoS is greater than T_{max} and N is significantly less (i.e., $N < 3$), fog servers infer that devices require resources higher than NB-IoT specifications for a short period. As a result, excess resources are allocated in the fog server's virtual memory.
- Case 4: In instances where QoS is lower than T_{min} or greater than T_{max}, N is moderately higher (i.e., from 4 to 9) and devices request an update in CoS, the fog servers update CoS and provide the corresponding QoS.
- Case 5: In cases where the device behaviour is similar to Case 4 but N is very high ($N > 10$), a continuous variation in QoS is observed. In such cases, fog servers transfer QoS to the cloud servers for further analysis. Cloud servers retrieve device QoS and packet flow data uploaded by the previous fog servers that the device has traversed and analyse for similar behaviour.
  - If either only abnormal packet flows or both the abnormal variation in previous QoS data and packet flows is observed, the behaviour is inferred as an attack and the device is blacklisted.
  - If QoS variation is observed only under the current fog server at repeated intervals of time, behaviour is inferred as a fault. However, some attackers may follow this pattern to avoid being detected. To differentiate between them, cloud servers reduce device's reputation by 1 and send feedback to the device to correct the fault. If reputation reduces to less than 4, the device is blacklisted.

A detailed working of this mechanism is given in Algorithm 1. Further, to enable memory optimisation across the fog layers, the deep learning algorithms study the *'Input file size'* sent by the device during message transfer and predict the *'Memory required (MB)'*. If the predicted memory size is lower than the device memory, the device processes the task; otherwise, it is processed in the fog server. Performance of the deep learning algorithms is evaluated in terms of prediction error rates, such as Mean Square Error (MSE), Mean Average Error (MAE), and Root Mean Square Error (RMSE), in addition to power, memory, and execution time required to execute the program. Output of the model with the best performance is chosen for resource allocation.

ALGORITHM 1 | Allocation of class of service (CoS) and quality of service (QoS).

Let CoS_{STD} = Standard CoS, CoS_{Pred} = Predicted CoS, QoS_D = Present QoS of device, $D[\cdot]$ = Pre-trained QoS dataset measured offline, $D_{Train}[\cdot]$ = Dataset train vector, $D_{Test}[\cdot]$ = Dataset test vector, T_{min} and T_{max} = minimum and maximum QoS limits for each CoS range, N = Number of QoS variations observed above T_{max} and below T_{min}, Rep = Device reputation
Assign CoS_{STD} based on device application 15
Categorise parameters into three CoS ranges, that is, Low (< 25% of T_{Max}), Medium (> 25% & < 75% of T_{Max}) and High (> 75% of T_{Max}), as shown in Table 4
Read QoS_D in real-time and convert to test vector, $D_{Test}[\cdot] \leftarrow QoS_D$
Append: $D_{Train}[\cdot] \leftarrow D[\cdot]$, based on device's CoS
Predict CoS_{Pred} using deep learning. Check the cases.
Case 1: $\{CoS_{STD} == CoS_T\}$ & {Device packet delay > Avg(Packet delays)} → Feedback to improve bandwidth
Case 2: $\{CoS_{STD} == CoS_T\}$ & $\{QoS_D < T_{min}$ & $N < 3\}$ → Feedback to client to increase QoS_D upto T_{min}
Case 3: $\{CoS_{STD} == CoS_T\}$ & $QoS_D > T_{max}$ & $N < 3$ → Allot excess resources in fog server's virtual memory
Case 4: $\{CoS_{STD} \neq CoS_T\}$ & $\{(QoS_D < T_{min})$ || $(QoS_D > T_{max})$ & $(N > 3$ & $N < 10)\}$ || {CoS updation requested by device} → Update CoS, assign new QoS_T
Case 5: $\{CoS_{STD} \neq CoS_T\}$ & $\{(QoS_D < T_{min})$ || $(QoS_D > T_{max})$ & $(N > 10)\}$ → Transfer QoS_D to cloud server. Cloud server retrieves previous QoS_D and packet flow data. Analyse using deep learning
if {Case 5 || Abnormal packet flow || Both} **then**
    Classify behaviour as attack
    Blacklist Device
else
    Classify behaviour as fault
    $Rep = Rep$ - 1
end if
Append: $D[\cdot] \leftarrow D_{Test}[\cdot]$, based on device's CoS
if $(Rep < 4)$ **then**
    Blacklist Device
end if

3.5 | Generation of Authentication Secrets

In this subsection, we discuss the generation of *Authentication Secrets*, that is, the seeds and secret key generation equations required to compute *Cloud Secret* and *Random Value*, by the Authentication server, whose notations are shown in Table 10. Device uses *Cloud Secret* during the handover process to prove its identity to the Authentication server and Cloud server. *Random Value* is used as a verification mechanism by the Authentication server during the authentication mechanism to ensure that the response is being received from the intended device. When the device turns-on each time in its home base station cell, the device requests *Authentication Secrets* from the Cloud server before connecting to the home base station. The Cloud server verifies the device's previous Unique ID and permanent address. It authorises the Authentication server to generate the $Seed_{Secret}$ and $Seed_{Value}$ using '*python uuid library*'. Authentication server then shares them with the device along with the new Unique ID, secret

TABLE 10 | Notations used in secret generation, authentication and handover protocols (Parameters stored in multiple locations are the same. An apostrophe is included to differentiate storage locations).

Storage location			
NB-IoT device	**Fog server**	**Cloud server**	**Description**
Cloud secret	—	*Cloud secret*'	Secret used by devices to prove their identity
Random value	*Random value*'	—	Secret to verify sender's authenticity
Q	—	Q'	Secret to verify base station's authenticity
$Seed_{Secret}$	—	$Seed'_{Secret}$	Seed required to verify the device identity during authentication with the Authentication server
—	—	$Seed_{Value}$	Seed required to verify the device identity during handover with the authentication server
$f_{Secret}(\cdot)$	—	$f'_{Secret}(\cdot)$	Secret key mechanism to compute *Cloud secret*
$f_{Value}(\cdot)$	—	$f'_{Value}(\cdot)$	Secret key mechanism to compute *Random value* and Q
$U_{Secret}(\cdot)$	—	$U'_{Secret}(\cdot)$	Mechanism to update $Seed_{Secret}$ after every usage
Unique ID	*Unique ID*'	*Unique ID*'	Unique ID of the device
Reputation	—	*Reputation*'	Device reputation
—	*Random number*	—	Generated by home base station to approve the device's handover request. Stored in fog server and verified by Authentication server as a proof of home base station's approval
N_S	—	N'_S	Number of successful authentication and handover requests for a particular $Seed_{Secret}$
N_{S_B}	—	N'_{S_B}	Number of successful authentication and handover requests of a base station for a particular $Seed_{Secret}$

key generation and update mechanisms required to generate and update $Seed_{Secret}$ as shown in Figure 5.

After every usage of $Seed_{Secret}$, it is updated using the update rule $U_{Secret}(\cdot)$ to ensure unlinkability of the values. N_S and N'_S are incremented by 1 at both the device and the cloud server to keep track of successful authentications and to maintain a synchronisation between N_S and N'_S. In cases of failed authentication, N_S having been would utilised by the device gets updated; however, N'_S remains the same. Hence, to maintain synchronisation between values, the device requests a new $Seed_{Secret}$ and resets N_S and N'_S to zero. The equations to calculate the *Authentication Secrets* are shown in Equations (3) and (4). The equation to update the $Seed_{Secret}$ is given by Equation (5).

$$Cloud\ Secret = f_{Secret}(Seed_{Secret}) \tag{3}$$

$$Random\ Value = f_{Value}(Seed_{Value}) \tag{4}$$

$$Seed_{Secret} = U_{Secret}(Seed_{Secret}||N_S) \tag{5}$$

Authentication server sends a $Seed_{Value}$ to the device through the visited base station, which in turn calculates the *Random Value* using the mathematical equation $f_{Value}(\cdot)$. The device responds with *Random Value* to prove its identity. Additionally, devices use $f_{Value}(\cdot)$ is used to calculate Q using $h(N_S)$ of the base station as shown in Equation (6) and compare it with Q' transmitted by the Authentication server calculated by Equation (7) to verify the base station's authenticity.

$$Q = f_{Value}(h(N_{S\_B})) \tag{6}$$

$$Q' = f'_{Value}(h(N'_{S\_B})) \tag{7}$$

3.6 | Authentication in Cell

In this subsection, we describe the handshake protocols required to authenticate the device in a cell before sending a message to the base station. The device sends an authentication request along with the nonce (N), timestamp (T), hash H calculated using Equation (8) and a session key (k_{AC}) to the base station. Upon receiving the request from the base station, the cloud server calculates *Cloud Secret*' using the $Seed'_{Secret}$ stored in its database using $CloudSecret' = f'_{Secret}(Seed'_{Secret})$.

Q' is encrypted using k_{AC} at the authentication server to ensure its confidentiality against possible Man-in-the-Middle and Sybil attacks on the base station and make it decryptable only by the device. $Q'_{k_{AC}}$ along with *Cloud Secret*' are then encrypted with the private key of the authentication server and transmitted to the base station as shown in Figure 6. The base station, in turn, generates H' as given by Equation (9), verifies the authenticity of the device, and sends $Q'_{k_{AC}}$, $h(N_S)$ and a session key (k_{AB}) to the device, which in turn verifies Q' with Q and confirms the authentication.

$$H = h(N, T, CloudSecret) \tag{8}$$

$$H' = h(N, T, CloudSecret') \tag{9}$$

3.7 | Authentication in Private Fog Networks

In this subsection, we propose a group authentication mechanism given by Algorithm 2 for devices registered under private fog servers as their home base stations. Since private fog

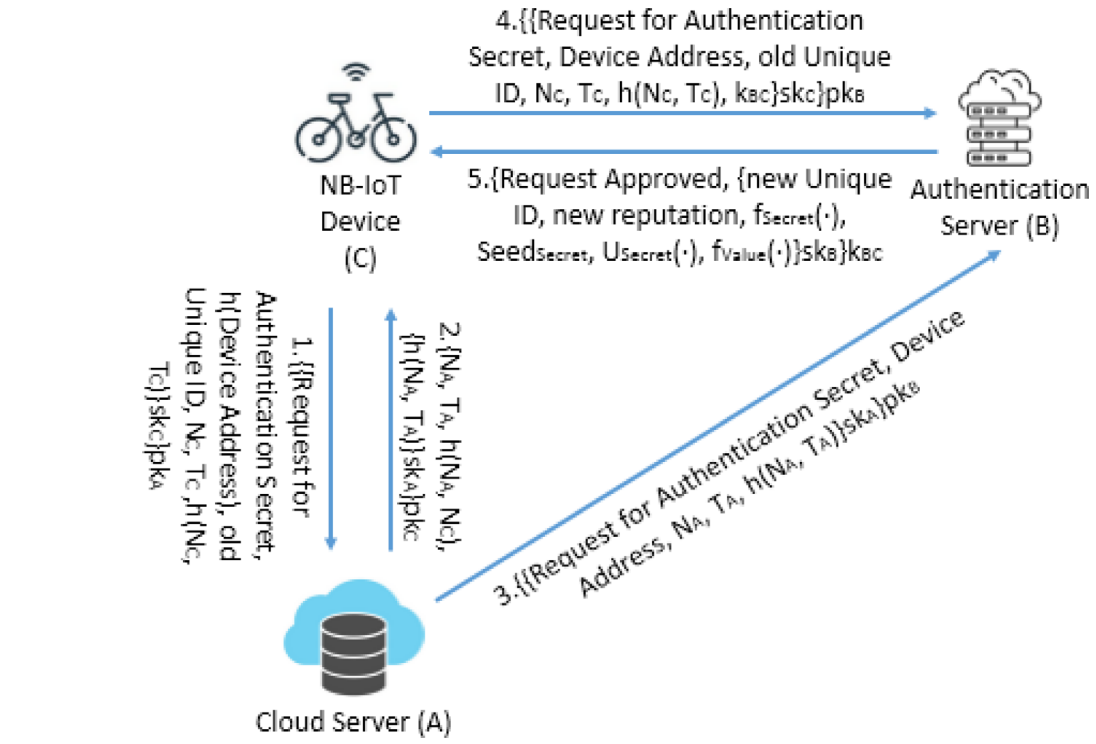

FIGURE 5 | Secret generation.

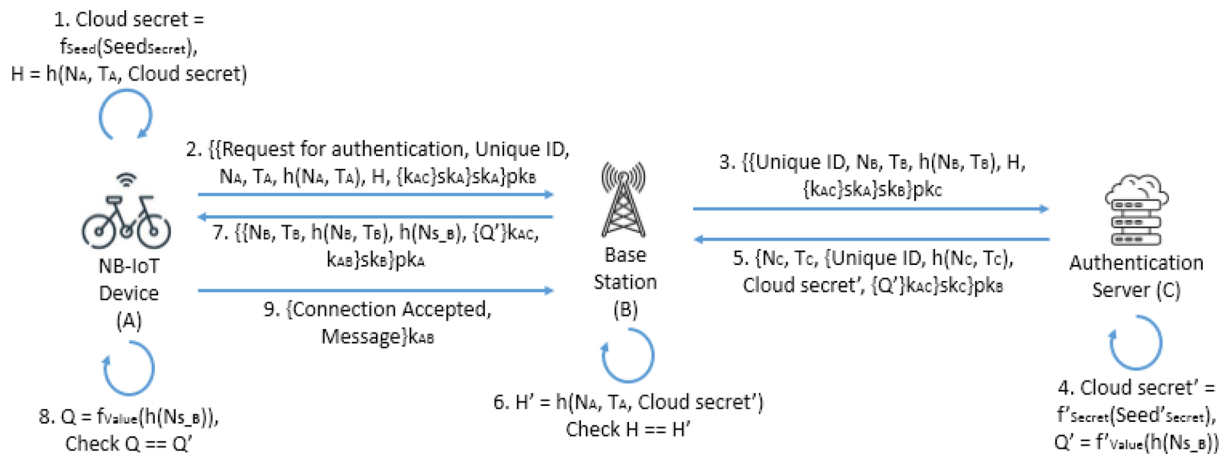

FIGURE 6 | Authentication in cell.

servers are commonly employed by organisations owing multiple devices, group authentication [53] would enable rapid authentication of all the devices simultaneously and blacklist attackers. Group authentication protocols proposed by existing works [54, 55] require the devices to generate a random value, append it to the IMSI of the device's SIM card, encrypt using the group key, and transmit it to the group leader. The group leader calculates the XOR of the data, encrypts it using the session key and forwards it to the base station for verification. However, as NB-IoT is resource-constrained, devices would consume significant batteries to implement these protocols. Hence, we modify the algorithms to meet the requirements of NB-IoT.

To enable group authentication, ensure the authenticity of the device request, and avoid replay of H and blacklisted *Unique IDs*, we calculate H by XORing the hash of *Cloud Secrets* and timestamp of all the devices present in the private network, as given by Equation (10). The fog server verifies the location of the devices by checking if their home base is the private fog server where they are present and checks the legitimacy of devices by comparing their *Unique ID* against the list of blacklisted devices. Further, the private fog server sends an authentication request and a session key to devices, as shown in Figure 7. The device generates the digital signature of Unique ID and T_A and encrypts the communication using the session key. This ensures device legitimacy, integrity, confidentiality and non-repudiation of H and Unique ID. Upon receiving H, the fog server transmits H and *[Unique ID and* T_A*]* pairs to the Authentication server. The authentication server calculates H' given by Equation (11), using T_A transmitted by the device and *Cloud Secret*' generated using $Seed'_{Secret}$,

ALGORITHM 2 | Group authentication in private fog networks.

Let $Seed_{Secret}$, $Seed'_{Secret}$ = Seeds stored in the device and Authentication server respectively, required to calculate *Cloud Secret*, *Rep* = Device reputation
$f_{Secret}(\cdot)$, $f'_{Secret}(\cdot)$ = Mathematical equations stored in the device and Authentication server to calculate *Cloud Secret*
$f_{Value}(\cdot)$, $f'_{Value}(\cdot)$ = Mathematical equations stored in the device and Authentication server to calculate *Random Value*
Calculate *Cloud Secret* at device. *Cloud Secret* = $f_{Secret}(Seed_{Secret})$
Transmit *h(Cloud Secret || T)* to fog server
XOR the *Cloud Secrets* at fog server to give $H = h(Cloud\ Secret_1||T_1) \oplus h(Cloud\ Secret_2||T_2) \oplus ... \oplus h(Cloud\ Secret_i||T_i)$
Check for Unique ID in blacklisted devices' list
Transmit H, Unique ID and {Unique ID, T_i}sk_i to Authentication server
Calculate *Cloud Secret* at Cloud. *Cloud Secret'* = $f'_{Secret}(Seed'_{Secret})$
Calculate H' at Authentication server, $H' = h(Cloud\ Secret'_1||T_1) \oplus h(Cloud\ Secret'_2||T_2) \oplus ... \oplus h(Cloud\ Secret'_i||T_i)$
if $\{H == H'\}$ **then**
    All devices authenticated
else
    Transmit $Seed_{Value_i}$ from Authentication server to all devices through fog server
    Calculate $Random\ Value_i = f_{Value_i}(Seed_{Value_i})$
    Transmit $\{UniqueID,\ h(Random\ Value'_i)\}sk_i$ to Authentication server through fog server
    Calculate $Random\ Value'_i = f'_{Value_i}(Seed_{Value_i})$
    if $\{h(Random\ Value_i) == h(Random\ Value'_i)\}$ **then**
        Device Verified
    else
        $Rep = Rep$ - 1
    end if
end if
if $(Rep < 4)$ **then**
    Blacklist Device
end if

compares with H and authenticates devices if verification is successful.

$$H = h(Cloud\ Secret_1||T_1) \oplus \cdots \oplus h(Cloud\ Secret_i||T_i) \quad (10)$$

$$H' = h(Cloud\ Secret'_1||T_1) \oplus \cdots \oplus h(Cloud\ Secret'_i||T_i) \quad (11)$$

In cases where H is not verified, the authentication server initiates an authentication procedure as shown in Figure 8. The authentication server transmits $Seed_{Value}$ and timestamp (T_C) encrypted using sk_C to the device through the fog server to ensure $Seed_{Value}$ is not replayed and prove its integrity to the device. The device calculates the *Random Value* using Equation (4) and transmits the *Unique ID* and *h(Random Value)* pair to the authentication server through the base station. The authentication server compares it with *h(Random Value')* and verifies the authenticity of the devices individually and blacklists attackers.

3.8 | Handover Between Cells or a Cell and a Private Fog Network

In this subsection, we discuss the mechanism required to provide a seamless handover as devices move across cells controlled by varying service providers and private fog networks owned by individuals. As some private fog networks may have different architectures and paging protocols, seamless hardware integration and handover with the remaining fog servers may not be compatible [56]. To overcome this limitation, our handover mechanism is designed to be interoperable across all fog server architectures as it replaces the standard paging protocols with the verification of device identity using *Random Value*.

The device intending to join the visited base station sends a '*Request to Leave*' to the home base station along with the present unique ID, reputation and address of the visited base station as shown in Figure 9. The home base station, in turn, responds with a *Random number* to be used as an identification by the authentication server for the home base station's approval. The *Random number* is generated using *secrets.token_bytes*() from *Secrets* class [57] in Python. The home base station then calculates H using Equation (12) and transmits to the authentication server. The Authentication server calculates *Random Value*' using Equation

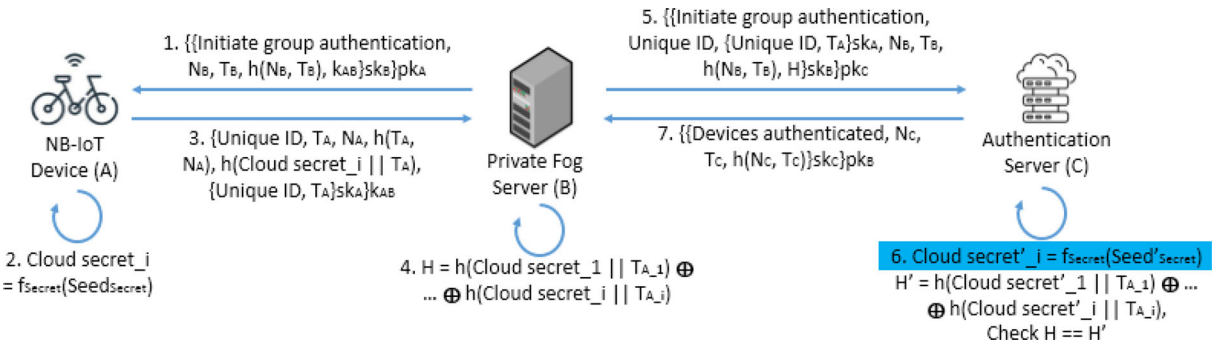

FIGURE 7 | Authentication in private fog networks (devices are verified).

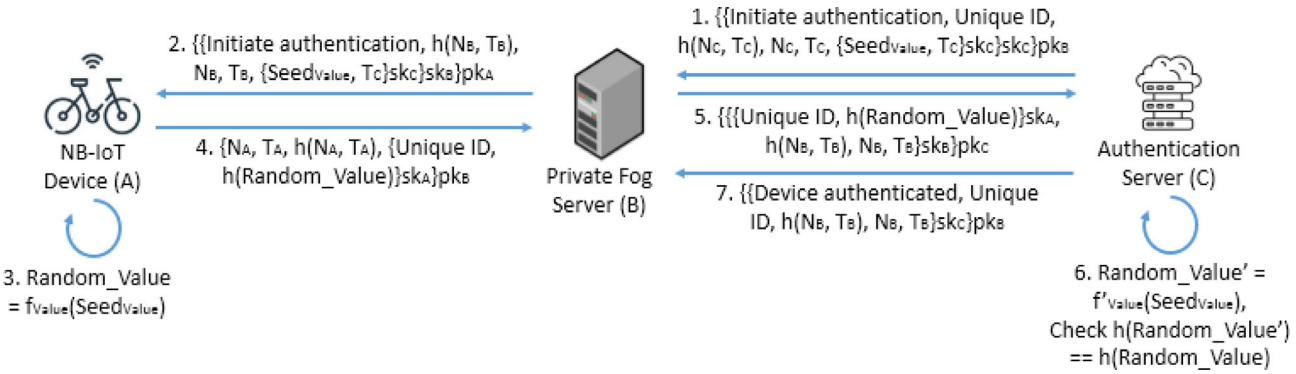

FIGURE 8 | Authentication in private fog networks (devices not verified).

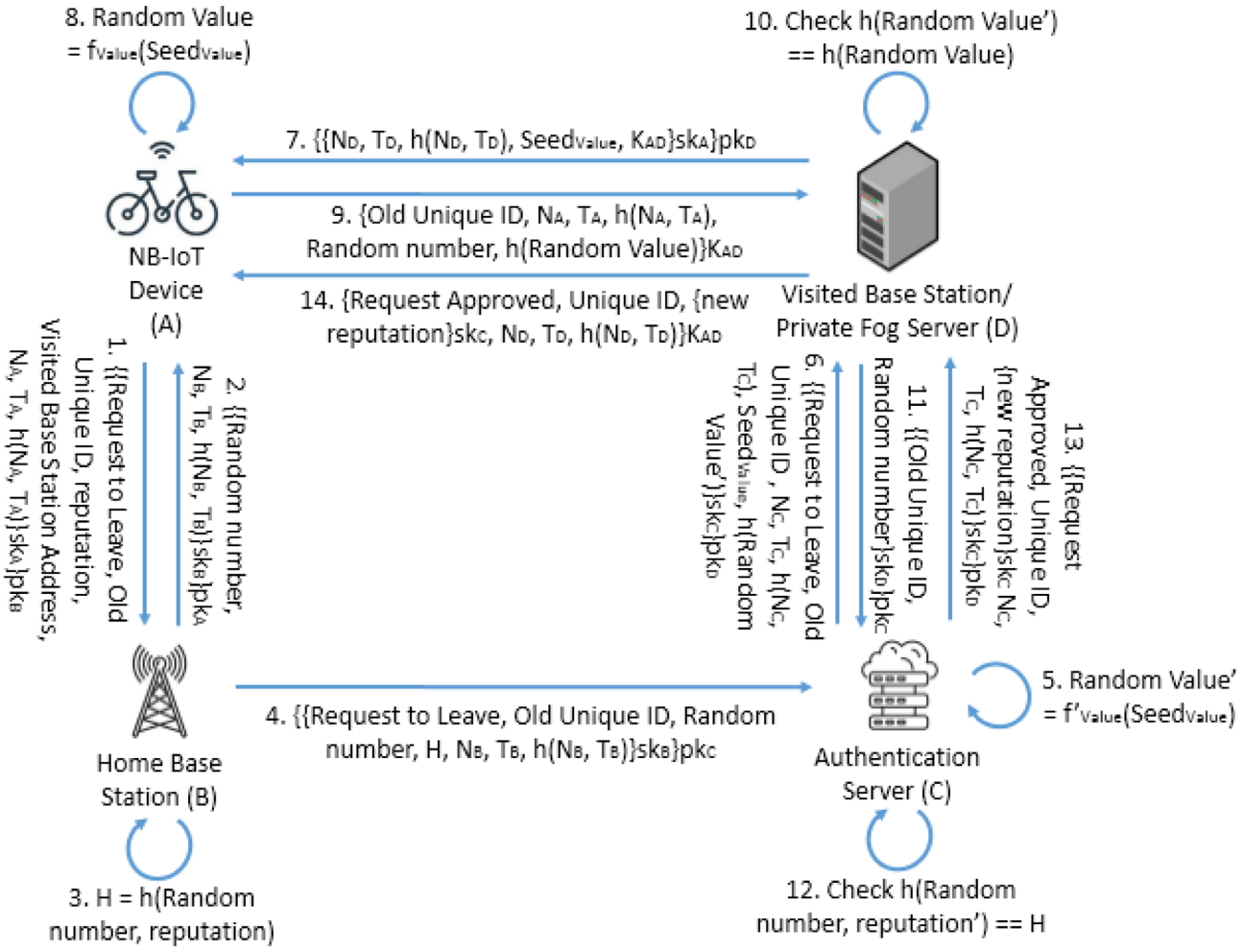

FIGURE 9 | Handover between cells or cell and private fog networks.

(13) and sends *h(Random Value'*) and $Seed_{Value}$ to the visited base station and device, respectively, which are used to verify the device's identity to the visited base station and the authentication server.

$$H = h(Randomnumber, \text{Reputation}) \tag{12}$$

$$RandomValue' = f'_{Value}(Seed_{Value}) \tag{13}$$

The device calculates *h(Random Value*) using Equation (4) and transmits it along with the *Random number* to the visited base station and Authentication server, respectively. The *h(Random number, reputation*) is calculated and verified with H at the Authentication server. *h(Random Value*) is checked with *h(Random Value'*) at the visited base station. If verified, the device's handover request is approved. Once the device enters the visited cell, it is clustered along with other devices based on their proximity to the visited base station. The threshold for proximity is given by the radius of the cell computed using Equation (14) as described in [58].

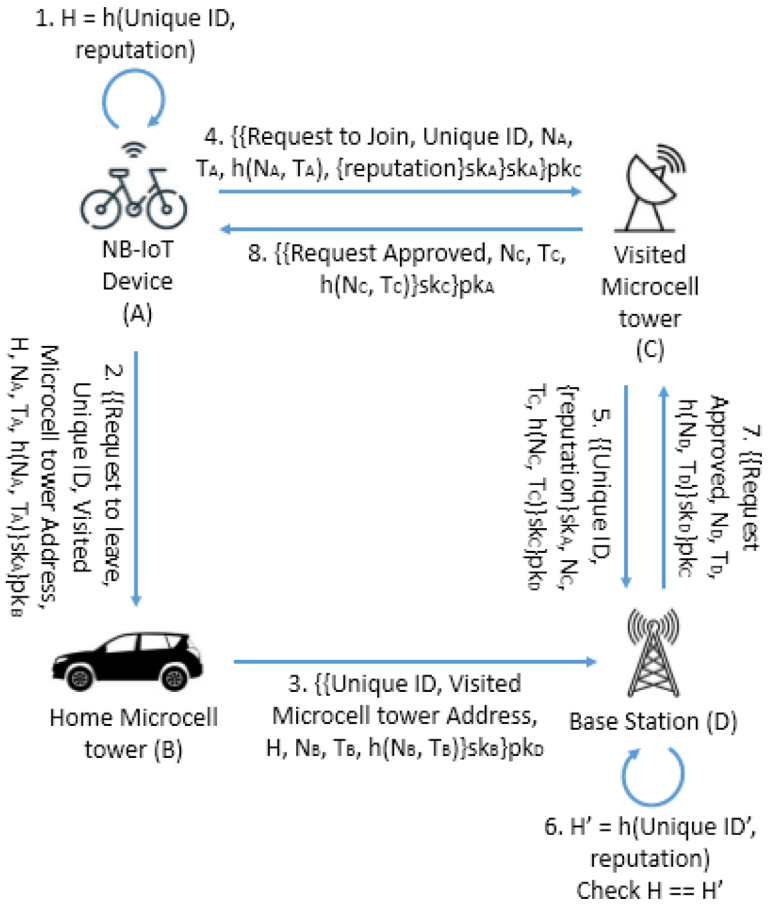

FIGURE 10 | Handover between microcells.

$$R = \sqrt{\frac{P_t}{P_r} * \frac{A_t A_r}{\lambda^2}} \tag{14}$$

where, P_t, P_r = Tx and Rx power, A_t, A_r = Tx and Rx antenna gain, λ = wavelength of radio signal ($\lambda = \frac{c}{f}$), c = speed of light ($3 * 10^8 m/s$), f = carrier frequency of the radio signal.

3.9 | Handover Between Microcells

In this subsection, we propose a handover between microcells when a cell is split to enable channel reuse as shown in Figure 10. Devices in microcells are provided connectivity by the base station through microcell towers comprising either microcell antennae or heavy-load NB-IoT devices having higher resources. However, microcell towers, when used for handover, may affect the quality of the signal [59]. Further, as the microcell towers have fewer resources compared to base stations, they will not be able to analyse and store data, as is the case in base stations. Hence, we design the microcell towers to act only as an intermediary to relay the communications between the device and base station. As the distance travelled by a device in a microcell is less, we simplify the handover procedure to enable rapid verification. The device calculates H using Equation (15) and transmits it to the fog server which in turn compares it with H' calculated using the *Unique ID*' and *Reputation*' stored in its database as shown in Equation (16). If verified, the handover request is approved. Once the device enters a microcell, it is clustered along with other devices based on the signal coverage radius of the microcell tower calculated using Equation (14).

$$H = h(\text{UniqueID}, \text{Reputation}) \tag{15}$$

$$H' = h(\text{UniqueID}', \text{Reputation}') \tag{16}$$

3.10 | Handover When Fog Server is Offline or Blacklisted

In this subsection, we design handover mechanisms to provide connectivity in situations where the home base station is either offline or blacklisted. In such cases, the cell is fragmented into multiple microcells, with microcell towers designed to only forward the device's communication to the cloud server. As microcell towers have lower resources than fog servers and the radius of microcells is low, a simplified handover mechanism is proposed to enable the device to connect with the nearest base station. Distance between the device and the visited based station is calculated using Pythagoras theorem, that is, $\sqrt{(X_2 - X_1)^2 - (Y_2 - Y_1)^2}$, where X_1, Y_1 are X and Y coordinates of the device and X_2, Y_2 are X and Y coordinates of the visited base station. To ensure broader signal coverage, the cell radius of the neighbouring cells is increased by varying the carrier frequency within the range of 1 920 MHz to 1 980 MHz [51] as given in by Equation (14).

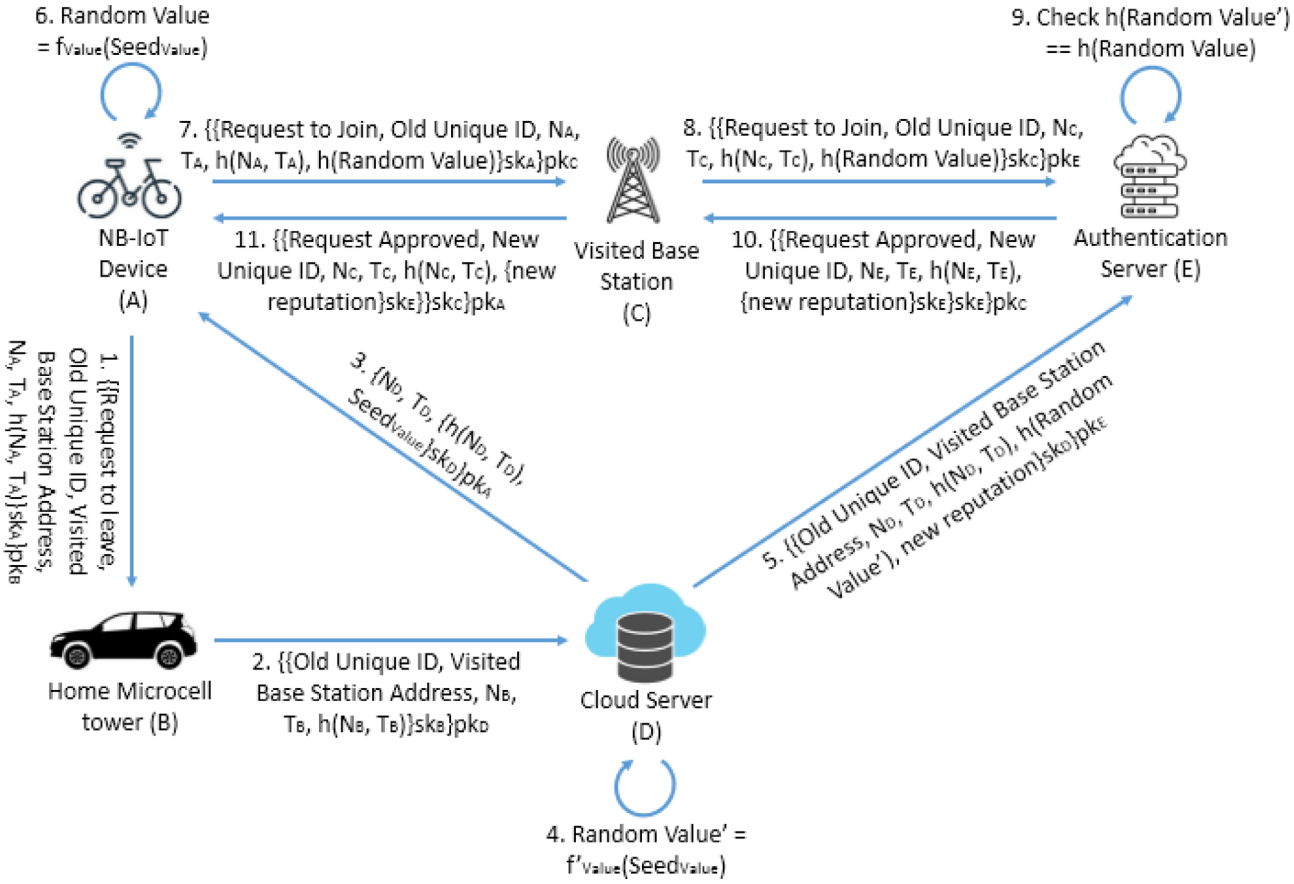

FIGURE 11 | Handover when fog server is offline or blacklisted.

Upon receiving the '*Request to Leave*' from the device through the microcell tower, the Cloud server shares the $Seed_{Value}$ to the device as shown in Figure 11. *h(Random Value'*) calculated using Equation (13) is sent by the Cloud server to the authentication server. The device then calculates *h(Random Value*) using Equation (4) and transmits it to the authentication server through the visited base station, where *h(Random Value'*) is compared with *h(Random Value*). If verified, the handover request is approved. When the home base station is back online, the communications are synchronised by the microcell towers, and the list of transactions stored in the microcell towers' cache is deleted.

3.11 | Reputation-Based Mechanism

In this subsection, we develop a reputation-based mechanism to detect malicious nodes and base stations. Due to the low bandwidth of 180 kHz, attackers can eavesdrop on NB-IoT devices, impersonate legitimate nodes, and misuse fog server resources by continuously varying the QoS requirements. To limit these attacks, we design the cloud server to detect abnormal device behaviour by analysing the QoS variations. Further, fog servers installed by independent entities are only partially reliable, as attackers may impersonate them to gain unauthorised access to devices' information, manipulate stored data, and launch DDoS attacks. Reputation is updated during handover between cells, *Secret Generation* protocol, and in case of attacks. To overcome these limitations and differentiate between attack and abnormal QoS, we introduce a reputation-based mechanism in devices and base stations.

Reputation is ranked between *1* to *10*, where *1* and *10* represent the lowest and highest reputation, respectively. We set *4* as the threshold value. Hence, devices or base stations with a reputation less than *4* or exhibiting attack behaviour are classified as malicious. Their '*Device Address*' is shared with the other base stations, and requests initiated by such devices are not serviced, and base stations are decommissioned as shown in Algorithm 1.

Reputation is calculated by the cloud server as a weighted average of absolute and relative behaviour. Absolute behaviour is determined by the votes of the authentication server, cloud server, base stations and microcell towers, whereas the NB-IoT devices give relative behaviour. The vote value ranges from *-10* to *10*. Equations to calculate the device, base station and final reputation are given by Equations (17) and (18), respectively. Votes collected from the reputation tracking sensors installed on the entities are transmitted to the cloud server using Elliptic Curve Cryptography (ECC) [60] along with a nonce and authentication tag generated using AES-GSM to ensure the confidentiality and integrity of votes. Cloud server calculates the new reputation and transmits it to the authentication server, which then updates the device with this value, as seen in Figures 5, 9 and 11.

In the case of calculating the device reputation, we consider the votes of the Authentication server, base stations and the average of remaining devices present in a particular cell. In the case of base stations, votes of the Authentication server, cloud server, microcell towers and devices are considered. As the Authentication server and base station verify and authenticate the devices, their votes are given a higher weight of 0.4 each. Cloud server analysing the device QoS rates the base station based on its load balancing strategy with a weight of 0.3. The remaining devices in that particular cell reporting the relative behaviour are given a weight of 0.2. Microcell towers rate the base stations based on their rapidity of being back online, with a weight of 0.1.

$$R_D += (0.4 * V_A + 0.4 * V_B + 0.2 * V_D) \quad (17)$$

$$R_B += (0.4 * V_A + 0.3 * V_C + 0.2 * V_D + 0.1 * V_M) \quad (18)$$

where,

R_D = Device reputation,

R_B = Base station reputation,

V_A = Authentication server vote,

V_B = Base station vote,

V_C = Cloud server vote,

V_M = Microcell tower vote,

V_D = Average of votes of all devices present in a particular cell at the time of vote

3.12 | Computational Complexity

In this subsection, we compute the computational complexity of the algorithm to calculate H in a private fog server as shown in Algorithm 2 and memory storage and retrieval of blacklisted unique IDs in terms of the memory and execution time in base stations and the authentication server.

3.12.1 | Group Authentication Algorithm in Private Fog Networks

To verify the authenticity of devices, the private fog server and authentication server calculate H as shown in Equation (8). H is calculated by bitwise XORing $h(Cloud \quad Secret||T_A)$ of size 256 bits, transmitted by all the devices.

Space Complexity

The space required to store the hashes of '*i*' number of devices is '*i*256*' bits. Space required for the resulting 256-bit output is *256 bits*. Therefore, the total space complexity would sum to *O(i*256 + 256)*. Since '*i*256 + 256*' is a constant, the space complexity simplifies to *[(i+1)*256]*O(1)*. Thus, space complexity becomes *O(1)*.

Time Complexity

As each of the bits is XORed in sequence, the time complexity is *O(256)*. Since 256 is a constant, the complexity equation would simplify to *256*O(1)*, resulting in a time complexity of *O(1)*.

3.12.2 | Memory Structure to Store Blacklisted Entities

Blacklisted NB-IoT devices and base stations are stored in the authentication server, and the rest of the base stations are stored in a hash table where the entity's permanent *Device Address* and unique IDs assigned to it are stored in the format of key-value pairs, that is, $Device \quad Address_N$: $[Unique \quad ID_1, Unique \quad ID_2, \ldots, Unique \quad ID_M]$.

Space Complexity:

Space complexity to store '*N*' number of key-value pairs each with '*M*' number of values would be O(N).

Time Complexity:

The time complexity for insertion, deletion, and search in a hash table using linear probing [61] depends on '*Load factor*' (α) given $\alpha = \frac{N}{M}$. Complexity to insert and search each of the unique IDs is $O(\frac{1}{1-\alpha})$. When the number of probes rises exponentially, the worst-case scenario complexity would sum to *O(1)*.

4 | Security Analysis and Mitigation Strategies

In this section, we analyse the security of the proposed fog computing framework against common cyber attacks and examine mitigation strategies.

4.1 | Device Anonymity

NB-IoT devices are assigned a *Device Address* as a permanent ID by the authentication server when they join the fog architecture for the first time. *h(Device Address)* is used by the device only for authentication with the cloud server during the *Secret Generation* process. Devices are assigned a *Unique ID* as an alias generated using '*python uuid library*' with the seed value set to *Nonce* to maintain anonymity. Hence, the devices' anonymity is protected, as attackers will not be able to predict the *Nonce* as the authentication server randomly generates it.

4.2 | Authenticity and Integrity

Message authenticity ensures that the messages transmitted by a sender originated from it. The security requirement is ensured by the usage of *Nonce*, *Device ID*, $Seed_{Secret}$ and HMAC in the authentication protocols. The usage of HMAC ensures the freshness of the message. In the '*Secret Generation*' protocol, the device transmits *[[Request for Authentication Secret, h(Device Address), Unique ID,* N_C, T_C, $h(N_C, T_C)]sk_C]pk_A$ to the cloud server as shown in Figure 5. As *Device Address* is generated by the Cloud

server when the device first joins the fog architecture, it is known only to the device and the cloud server. Hence, attackers eavesdropping on the channel cannot decipher it. Further, in the authentication protocols, the device generates Cloud Secret using $Seed_{Secret}$, calculates $H = h(N_A, T_A, Cloud\ secret)$ and transmits *[[Request for authentication, Unique ID*, N_A, T_A, $h(N_A, T_A)$, H, $\{k_{AC}\}sk_A]sk_A]pk_B$ to the base station as shown in Figure 6. $Seed_{Secret}$ is updated after every usage. The cloud server shares a new $Seed_{Secret}$ in every '*Active*' state. Hence, eavesdropping and reusing them to launch attacks is not feasible, thus ensuring the message's authenticity.

To ensure messages are not modified in transit, integrity is ensured by nonce and timestamp as $h(N_A, T_A)$ in the authentication and handover protocols. Receivers store the previous nonces received. Upon receiving the authentication requests, the receiver calculates $h(N_A, T_A)$ from N_A and T_A sent in the request and compares it with $h(N_A, T_A)$ sent in the request. If the hash is not verified or nonce is re-used, the message is rejected.

In cases of failed authentication, $Seed_{Secret}$, having been calculated to initiate the protocol, is updated at the device end but remains the same at the authentication server. Further, in some cases, attackers may gain access to and modify the $Seed_{Secret}$. Hence, to maintain the integrity of $Seed_{Secret}$ and ensure its value is identical at both the device and authentication server, N_S is appended to $Seed_{Secret}$ to calculate the updated value as seen in Equation (5). If N_S and N'_S vary, the device requests a new $Seed_{Secret}$ to resynchronise its value with the authentication server and resets N_S and N'_S to zero. If still a variation in N_S is observed, the device is blacklisted.

4.3 | Man-In-The-Middle Attack

Attackers launch Man-in-the-Middle attacks by eavesdropping and altering the messages. To prevent these attacks, we introduce nonce (N_A) and timestamp (T_A) in the authentication and handover requests, as seen in Figure 9. N_A, T_A and $h(N_A, T_A)$ are transmitted by the device to the home and visited base stations along with the handover request. Any message tampering would alter T_A, thus making the request invalid as $h(N_A, T_A)$ would not be verified. Each communication between entities is encrypted using the private key of the sender (sk_A) and the public key of the receiver (pk_B) to ensure that unintended entities do not access the message contents. Additionally, *Random Value* and H transmitted by the device calculated using Equations (4) and (12) are verified at the authentication server and the visited base station, respectively, thus authenticating the device identity.

In the authentication protocol proposed in Figure 7, the device calculates and transmits *h(Cloud Secret)* to the Authentication server encrypted using session key (k_{AB}) as shown in *[Unique ID*, T_A, N_A, $h(T_A, N_A)$, $h(Cloud\ \ secret_i, T_A)$, *{Unique ID*, $T_A\}sk_A]k_{AB}$. This ensures end-to-end encryption by limiting the message's visibility only to the receiver. Further N_{S_B}, *Cloud Secret* and *Random_Value* required to verify the identity of devices in authentication and handover protocols are hashed using SHA-256 and transmitted across entities. Probability of sniffing and brute-forcing these packets for 'n' number of trials would be $1 - (1 - 2^{-256})^n$, which is nearly zero, thus protecting its integrity and confidentiality.

4.4 | Impersonation Attack

In impersonation attacks, attackers impersonate devices and base stations to steal personal information. To avoid these attacks on the devices, we design the device to send *h(Device Address)* in the *Secret Generation* protocol. As *Device Address* is maintained a secret by the device and the Cloud server, impersonation attacks aimed at stealing $Seed_{Secret}$, $f_{Secret}(\cdot)$, $f_{Value}(\cdot)$ and $U_{Secret}(\cdot)$ sent by authentication server upon device verification are eliminated. In authentication protocols, the device generates *Cloud Secret* using $Seed_{Secret}$ and transmits $H = h(N_A, T_A, Cloud\ Secret)$ along with the handover request to prove its identity to the Authentication server. $Seed_{Secret}$ is appended to N_S and updated using $U_{Secret}(\cdot)$ after every usage to ensure that it is indecipherable. Further, we design the authentication server to calculate Q' using $h(N_{S\_B})$ and transmit it to the device. The device in turn calculates Q using $f_{Value}(\cdot)$ and verifies the base station identity. As N_{S_B} is a secret between the Authentication server and the base station and that $f_{Value}(\cdot)$ is known only to the device and Authentication server, impersonation attacks on the base stations can be efficiently detected. In the case of handover protocols, as seen in Figure 11, the cloud server transmits a digital signature, that is, $\{Seed_{Value}\}sk_D$ to the device to generate *Random Value* using $f_{Value}(\cdot)$. *Random Value* proves the device's identity to the visited base station and $\{Seed_{Value}\}sk_D$ preserves non-repudiation, thus limiting impersonation attacks.

4.5 | Replay Attack

Attackers initiate replay attacks by eavesdropping and replaying a certain portion or the entire conversation to the receiver to gain illicit access to the device information. To avoid these attacks, we design the protocols to include a nonce (N_A) and timestamp (T_A) as shown in *[[Request for authentication, Unique ID*, N_A, T_A, $h(N_A, T_A)$, $H]sk_A]pk_B$. If the attacker replays the message with new N_A and T_A, it will not match h(N_A, T_A), leading to the message's rejection. Additionally, in authentication protocols, H calculated using $h(N_A, T_A, Cloud\ Secret)$ is transmitted by the device in the authentication request. *Cloud Secret* is calculated using $Seed_{Secret}$ shared by the authentication server in every '*Active*' state and $Seed_{Secret}$ is updated after every usage using $U_{Secret}(\cdot)$. Hence, the messages replayed will not be able to guess the current $f_{Secret}(\cdot)$, $U_{Secret}(\cdot)$ and $Seed_{Secret}$, thus avoiding replay attacks.

In the case of authentication in private fog networks proposed in Figure 7, to prevent attackers from eavesdropping and replaying the hashed secrets, we hash the appended timestamp and the cloud secret as seen in {*Unique ID*, T_A, N_A, $h(T_A, N_A)$, $h(Cloud\ \ Secret_i||T_A)$, {*Unique ID*, $T_A\}sk_A\}k_{AB}$. If attackers replay the $h(Cloud\ \ Secret_i||T_A)$, the authentication server will be able to reject the request by checking its authenticity using T_A from $\{Unique\ ID, T_A\}sk_A$, where T_A is encrypted with the private key of the device. Further, attackers creating a dictionary of unique IDs and hashes and replaying them would not be successful as the *Device Address* is known only to the device, cloud

and authentication server and it is used only during *Secret Generation* requests, as shown in Figure 5. Unique ID, pseudonym, is updated during handover, and every time authentication and handover requests fail, the probability of brute-forcing hashes is nearly zero, as discussed in Section 4.3.

4.6 | Denial of Service (Dos)

Attackers can attempt to disrupt network traffic by launching DoS and Distributed Denial of Service (DDoS) attacks to flood transactions in the fog architecture. To avoid these attacks, we design the authentication and handover protocols to include N_A, T_A and $h(N_A, T_A)$. When attackers initiate multiple transactions, the receiver checks $h(N_A, T_A)$, ignores attack packets and blacklists them. A variation of this attack can lead to selfdenial of service, where a blacklisted device or a base station initiates authentication requests in private fog networks and negates *H == H'* and *h(Random_Value') == h(Random_Value)* checks. This results in an infinite authentication loop. To alleviate this limitation, the Unique IDs of the blacklisted entities are stored and updated regularly at the base stations, Cloud and Authentication servers, which verify the unique ID of the entity against the record of blacklisted entities and reject them. In cases where devices manipulate the permanent *Device Address* to evade blacklisting, the authentication server checks the authenticity of *h(Device Address)* during *Secret Generation* and rejects the request.

4.7 | Sybil Attack

Attackers launch Sybil attacks by creating multiple fake identities of a single node to inject fake data into the database and exhaust resources in the fog servers. The cloud server checks the *Device Address* to overcome these attacks. It authorises the authentication server to assign the devices with unique ID and *Authentication secrets* when devices enter a new cell. Hence, Sybil nodes within a particular base station are prevented. Further, when devices move across cells, $Seed_{Value}$ is sent by the cloud server to the device to calculate *Random Value*. *h*(*Random Value*) is verified at the visited base station to permit the device into the cell. The fog server at the home base station sends device data to the cloud and deletes data locally at its database, thus preventing multiple entities across various cells. In the case of private fog servers, group authentication is used to verify device identity. The devices generate *h*(*Cloud Secret*) and transmit it to the fog server, which in turn calculates $H = h(Cloud\ \ Secret_1\ \ ||\ \ T_1) \oplus \cdots \oplus h(Cloud\ \ Secret_i\ \ ||\ \ T_i)$. If multiple *h(Cloud Secret)* are detected, a Sybil attack is inferred, and the Cloud server verifies individual devices using $Seed_{Value}$ as shown in Figure 8 and blacklists Sybil nodes.

4.8 | Lateral Movement

Lateral movement is defined as the attackers' ability to move horizontally across the fog computing infrastructure to gain unauthorized access to devices and fog servers. To prevent such attacks across cells, we design authentication and handover mechanisms ensuring the authenticity of the devices joining the visited base stations and the integrity of the messages transmitted. Additionally, the authentication secrets are generated by the cloud every time the device turns on and are updated after every usage to ensure unlinkability. Fog servers are designed to share the device QoS with only the cloud and not the neighbouring fog servers as described in Section 3.1 since fog servers controlled by private individuals may not be entirely trusted. To prevent attacks within cells, the fog servers are designed to transfer device QoS to the cloud and clear their cache when devices leave cells. Further, we introduce a reputation-based mechanism to track the device and base station behaviour. When attacks are detected, reputation is reduced and devices are blacklisted.

4.9 | Forward Secrecy

Forward secrecy attacks are initiated by attackers to gain access to the long-term private keys and decrypt past encrypted data. These attacks can be limited by sharing device secrets and encryption keys using mutual authentication, generating temporary private keys and updating the secrets to ensure unlinkability. In our proposed architecture, we design the authentication server to generate and share $Seed_{Secret}$, $f_{Secret}(\cdot)$, $f_{Value}(\cdot)$ and $U_{Secret}(\cdot)$ every time the device enters '*Active*' state and authenticates itself in the home base station as shown in Figure 5. These secrets are encrypted using a digital signature to ensure non-repudiation. They are transmitted to the device along with '*Request Approved*' encrypted using the session key k_{BC} to ensure the non-reusability of secrets. Further, the seeds are updated after every usage using $U_{Secret}(\cdot)$ to ensure freshness and unlinkability of the value, thus avoiding forward secrecy.

4.10 | Resource Depletion Attacks

Resources in the fog computing framework can be depleted in the following ways.

4.10.1 | Malicious Nodes

To deplete fog server resources such as bandwidth, memory, CPU processing, etc., attackers repeatedly send requests for an update in QoS or may continuously vary the QoS requirements. To detect these attacks and to differentiate this behaviour from faulty devices, the fog server transmits the device QoS to the cloud. The cloud server retrieves the device's QoS and packet flow data uploaded by the previous fog server and checks for similar behaviour and abnormal packet flows. If either QoS variation or abnormal packet flows are detected, the cloud classifies it as an attack, and the cloud blacklists the device. Else, behaviour is inferred as a fault, and reputation is reduced to isolate attacks mimicking similar behavior.

4.10.2 | Resource Exhaustion

Resource exhaustion occurs when the QoS demands of the devices saturate the available computational resources in the fog server. This leads to the inability of fog servers to cater to the devices' needs. To eliminate this limitation, we introduce load

balancing, where the fog nodes analyse QoS in real-time using deep learning algorithms and predict the resources required to process a particular device request. If the predicted QoS is within the limits of device specifications, the fog server sends feedback to the device to allocate the required QoS. If the requirement exceeds the device specifications, excess resources are allocated at the fog server's virtual memory. Further, when a device leaves the cell, its QoS data is transmitted by the base station to the cloud and deleted locally at the fog server to enhance scalability.

5 | Implementation and Experimental Set-Up

In this section, we discuss the software simulator and the hardware set-up required to design the fog computing framework. We simulate the performance of device movement, load distribution, and device clustering and visited the base station choosing process in fog architecture using the iFogSim 2 simulator [13]. We implement the handover and authentication protocols on Raspberry Pi 4 [14]. Further, we formally validate the robustness of the authentication and handover protocols using Scyther [15] to prove that they are immune to known attacks. Performance of the proposed architecture is measured in terms of cloud execution time, energy, power and memory consumption on the iFogSim 2 and Raspberry Pi 4 platforms. A brief description of implementation platforms and performance metrics is given below.

5.1 | Implementation on Raspberry Pi 4

In this subsection, we describe the implementation of the secret generation, handover and authentication protocols as a UDP client-server program on the Raspberry Pi 4, where the Raspberry Pi 4 and laptop were programmed as client and server, respectively, and vice versa. A screenshot of the implementation setup is shown in Figure 12.

Since authentication, cloud and fog servers are designed to store and process device data and monitor and predict CoS in real time, significant RAM is required. Hence, we consider Raspberry Pi 4 with 4GB RAM for implementing IoT gateways, fog and edge servers due to its built-in Ethernet and wireless connectivity [14] instead of previous low-memory versions such as Raspberry Pi 3, Raspberry Pi Pico, etc.

5.2 | Implementation on IFogSim 2

In this subsection, we implement the fog architecture on iFogSim 2. We initialise the cloud as Layer-0, fog servers as Layer-1 and NB-IoT devices as Layer-2 and simulate the performance for homogeneous and heterogeneous device configurations. Homogeneous configurations refer to the ideal conditions where private fog servers are not present and all devices and fog servers have the same system configurations, such as MIPS (computational power), RAM (memory requirement), Tx and Rx bandwidth, busy (Active) and idle power. On the contrary, heterogeneous configuration refers to the non-ideal conditions where private fog servers are present and devices and fog servers have varying system configurations. Hence, multiple functions required to create fog servers based on individual service provider requirements are defined using the *createFogDevice function* present in *FogDevice.java*. Similarly, sensors and actuators required to input and execute data are initiated separately to cater to various sensor and actuator requirements using *Sensor class* and *Actuator class* defined in *Sensor.java* and *Actuator.java*, respectively. The time interval with which sensors sense the inputs is set using *sensor.setLatency*().

To schedule packet transmission time, the tuple emission rate is set using the *DeterministicDistribution() function*. In *Secret Generation* protocol, to enable the device to communicate with the cloud directly, *addModuleToDevice* ("*picture-capture*", "*cloud*"), that is, object detector module, and *addModuleToDevice* ("*slot-detector*", "*cloud*"), that is, object tracker module, a part of *moduleMapping function* are declared and set to *TRUE*. Similarly, device-to-fog server communication is initiated by setting the above-specified functions to *FALSE*. To facilitate mobility, devices and fog servers are assigned X and Y coordinates; a *mobilityMap function* tracking the continuous variation in (X, Y) is defined and *mobilityDestinationId function* representing the visited base station ID is set. To cluster devices present inside cells and to enable new connection to a base station based on its distance from the device, cell radius and distance between the device and the base station are calculated using Pythagoras theorem and Equation (14).

5.3 | Verification and Validation Using Scyther

In this subsection, we formally verify the authentication and handover protocols in the proposed fog computing framework using

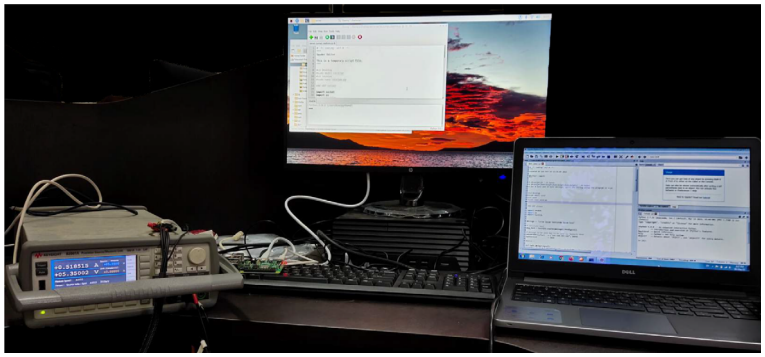

FIGURE 12 | Hardware setup.

Scyther [15]. Scyther is written using Python 3 and works based on the OpenSSL library [62] to define cryptographic primitives and functions. Scyther employs Satisfiability Modulo Theories (SMT) solvers to check the consistency of the protocol satisfying various security properties. The tool outputs the results in terms of '*No attacks*', '*No attacks within bounds*' and '*Attacks found*'. '*No attacks*' shows that the protocol is secure under all possible scenarios and conditions without relying on specific assumptions about the capabilities of attackers. '*No attacks within bounds*' suggests that the protocol is secure under the given assumptions or within the defined security boundaries. The tool verifies claims such as *Weakagree*, *Commit*, *Secret*, *Alive*, *Niagree* and *Nisync* and shows that the receiver is active during protocol execution, the receiver has initiated the response, protocol parameters are secret from eavesdroppers, and the protocol operates with data consistency on all execution steps.

Verification report of the '*Secret Generation*' given by Figure 13a, while those of authentication protocols comprising '*Authentication in Cells*', '*Authentication in Private Fog Networks when Devices are Verified*' and '*Authentication in Private Fog Networks when Devices are not Verified*' are shown in Figures 13b and 14a,b respectively. Output of handover protocols, that is, '*Handover between Cells or a Cell and a Private Fog Network*', '*Handover between Microcells*' and '*Handover when Fog Server is Offline or Blacklisted*' are given in Figures 15a,b and 16 respectively.

a

Scyther results : verify

Claim				Status		Comments
Block	C	Block,C2	Secret ncc	Ok		No attacks within bounds.
		Block,C3	Alive	Ok		No attacks within bounds.
		Block,C4	Niagree	Ok	Verified	No attacks.
		Block,C5	Nisynch	Ok	Verified	No attacks.
	A	Block,A3	Secret na	Ok	Verified	No attacks.
		Block,A4	Niagree	Ok	Verified	No attacks.
		Block,A5	Nisynch	Ok	Verified	No attacks.
	B	Block,B1	Secret nb	Ok		No attacks within bounds.
		Block,B2	Alive	Ok		No attacks within bounds.
		Block,B3	Niagree	Ok		No attacks within bounds.
		Block,B4	Nisynch	Ok		No attacks within bounds.

Done.

b

Scyther results : verify

Claim				Status		Comments
Block	A	Block,A2	Secret na	Ok		No attacks within bounds.
		Block,A3	Secret kAB	Ok		No attacks within bounds.
		Block,A4	Alive	Ok	Verified	No attacks.
	B	Block,B2	Secret nb	Ok		No attacks within bounds.
		Block,B3	Secret kAB	Ok		No attacks within bounds.
		Block,B4	Alive	Ok		No attacks within bounds.
		Block,B5	Niagree	Ok		No attacks within bounds.
		Block,B6	Nisynch	Ok		No attacks within bounds.
	C	Block,C1	Secret na	Ok		No attacks within bounds.
		Block,C2	Alive	Ok		No attacks within bounds.
		Block,C3	Niagree	Ok		No attacks within bounds.
		Block,C4	Nisynch	Ok		No attacks within bounds.

Done.

FIGURE 13 | Scyther verification results (a) secret generation, (b) authentication in cells.

a

Scyther results : verify

Claim				Status		Comments	Pattern
Block	A	Block,A3	Secret na	Ok	Verified	No attacks.	
		Block,A4	Secret kAB	Ok	Verified	No attacks.	
		Block,A5	Niagree	Ok	Verified	No attacks.	
		Block,A6	Nisynch	Ok	Verified	No attacks.	
	B	Block,B2	Secret nb	Ok	Verified	No attacks.	
		Block,B3	Secret kAB	Ok	Verified	No attacks.	
		Block,B2	Alive	Ok		No attacks within bounds.	
		Block,B5	Niagree	Ok	Verified	No attacks.	
		Block,B6	Nisynch	Ok	Verified	No attacks.	
	C	Block,C1	Secret nc	Ok		No attacks within bounds.	
		Block,C2	Alive	Ok		No attacks within bounds.	

Done.

b

Scyther results : verify

Claim				Status		Comments
Block	B	Block,B2	Secret nb	Ok		No attacks within bounds.
		Block,B3	Secret kAB	Ok		No attacks within bounds.
		Block,B4	Alive	Ok		No attacks within bounds.
		Block,B5	Weakagree	Ok		No attacks within bounds.
		Block,B6	Niagree	Ok		No attacks within bounds.
		Block,B7	Nisynch	Ok		No attacks within bounds.
	A	Block,A2	Secret na	Ok	Verified	No attacks.
		Block,A3	Secret kAB	Ok	Verified	No attacks.
		Block,A4	Weakagree	Ok	Verified	No attacks.
		Block,A5	Niagree	Ok	Verified	No attacks.
		Block,A6	Nisynch	Ok	Verified	No attacks.
	C	Block,C1	Secret nc	Ok		No attacks within bounds.
		Block,C2	Alive	Ok		No attacks within bounds.

Done.

FIGURE 14 | Scyther verification results for authentication in private fog networks (a) devices verified, (b) devices not verified.

a

Scyther results : verify

Claim				Status		Comments
Block	A	Block,A1	Secret na	Ok		No attacks within bounds.
		Block,A2	Secret kAD	Ok		No attacks within bounds.
		Block,A3	Secret {SeedValue}fValue	Ok	Verified	No attacks.
	B	Block,B1	Secret nb	Ok	Verified	No attacks.
	C	Block,C2	Secret nc	Ok	Verified	No attacks.
		Block,C3	Niagree	Ok	Verified	No attacks.
		Block,C4	Nisynch	Ok	Verified	No attacks.
	D	Block,D2	Secret nd	Ok	Verified	No attacks.
		Block,D3	Secret kAD	Ok	Verified	No attacks.
		Block,D4	Alive	Ok	Verified	No attacks.
		Block,D5	Weakagree	Ok	Verified	No attacks.
		Block,D6	Niagree	Ok	Verified	No attacks.
		Block,D7	Nisynch	Ok	Verified	No attacks.

Done.

b

Scyther results : verify

Claim				Status		Comments
Block	A	Block,A2	Secret na	Ok	Verified	No attacks.
		Block,A3	Niagree	Ok	Verified	No attacks.
		Block,A4	Nisynch	Ok	Verified	No attacks.
	B	Block,B1	Secret nb	Ok	Verified	No attacks.
		Block,B2	Secret h(UID,Rep)	Ok		No attacks within bounds.
		Block,B3	Niagree	Ok	Verified	No attacks.
		Block,B4	Nisynch	Ok	Verified	No attacks.
	C	Block,C1	Secret nc	Ok	Verified	No attacks.
		Block,C2	Alive	Ok	Verified	No attacks.
		Block,C3	Weakagree	Ok	Verified	No attacks.
		Block,C4	Niagree	Ok	Verified	No attacks.
		Block,C5	Nisynch	Ok	Verified	No attacks.
	D	Block,D1	Secret nd	Ok		No attacks within bounds.
		Block,D2	Secret h(UID,Rep)	Ok		No attacks within bounds.
		Block,D3	Alive	Ok		No attacks within bounds.
		Block,D4	Weakagree	Ok		No attacks within bounds.
		Block,D5	Niagree	Ok		No attacks within bounds.
		Block,D5	Niagree	Ok		No attacks within bounds.
		Block,D6	Nisynch	Ok		No attacks within bounds.

Done.

FIGURE 15 | Scyther verification results for handover between (a) cells or a cell and a private fog network, (b) microcells.

Scyther results : verify

Claim				Status		Comments
Block	A	Block,A1	Secret na	Ok		No attacks within bounds.
		Block,A2	Secret SeedValue	Ok		No attacks within bounds.
	B	Block,B1	Secret nb	Ok	Verified	No attacks.
	C	Block,C1	Secret nc	Ok	Verified	No attacks.
	D	Block,D1	Secret nd	Ok		No attacks within bounds.
	E	Block,E1	Secret ne	Ok		No attacks within bounds.

Done.

FIGURE 16 | Handover when fog server is offline or blacklisted.

5.4 | Performance Metrics

The performance of the fog computing framework is measured in terms of power, memory, execution time and network usage. In contrast, the performance of deep learning algorithms is measured in terms of MSE, MAE and RMSE in addition to the above parameters. A brief description of the performance parameters is given below.

5.4.1 | Power/Energy

Energy in iFogSim 2 is measured using the *UpdateEnergyConsumption*() function present in the *PowerDatacenter* class. This function outputs the energy of all the devices during execution. While implementing on a Raspberry Pi 4 [63], K2901A Source Unit [64] was used to power the device at 5.42V and 3A. Power consumed was recorded by observing the variations during run-time.

5.4.2 | Memory

Memory consumed to execute authentication and handover protocols in Raspberry Pi 4 was measured using the *Task Manager*.

5.4.3 | Execution Time

Execution time in Raspberry Pi 4 is defined as the cumulative time required for packet formation, transfer and verification. It

was measured by computing the difference between the start time (T_{start}) and stop time (T_{stop}) taken to execute the protocols as given by, $T_{Execution} = T_{Stop} - T_{Start}$.

In the case of iFogSim 2, execution time is measured in terms of '*Application loop delay*' defined as the total delay encountered during workflow execution encompassing the time taken to transmit authentication and handover requests to the fog or cloud servers, processing time to simulate and arrange the fog architecture components, and the latency encountered on account of generating devices' sensor output, processing and returning it to the actuators. Execution time in iFogSim 2 is calculated using *TimeKeeper.getInstance*() library.

5.4.4 | Network Usage

Network usage is used in iFogSim 2 to measure the bandwidth utilization of information transferred through the network interfaces for running an application on the fog architecture. It is calculated using the *getNetworkUsage*() function, a part of *NetworkUsageMonitor.java* in iFogSim 2.

6 | Performance Evaluation

In this section, we evaluate the performance of our architecture, that is, power, memory and execution time for secret generation, authentication and handover protocols on Raspberry Pi 4. Further, we study the performance of device clustering, system operation under homogeneous and heterogeneous configurations and load sharing during handover in terms of energy, execution time and network usage on iFogSim 2. QoS assigned to devices upon authentication is predicted using deep learning algorithms in terms of MSE, RMSE and MAE in addition to power, memory and execution time. Performance of the NB-IoT channel is studied in terms of RTT, throughput, and packet delivery ratio. A detailed analysis of the results observed for each case is given below. Further, we compare our results with those of the existing approaches and show that our approach is lightweight.

6.1 | Protocol Implementation on Raspberry Pi 4

As NB-IoT does not support seamless handover, we design and implement '*Secret Generation*' protocol as described in Section 3.5, authentication protocols explained in Sections 3.6 and 3.7 and handover protocols as shown in Sections 3.8, 3.9, 3.10 on Raspberry Pi 4. Performance values thus measured along with those of ECC used to transfer votes from devices and base stations to the cloud server are recorded in Table 11. We observe that the performance of algorithms depends on the operation performed, that is, handover, authentication or secret generation; communication of the device with a cloud or fog server; number of entities required to execute the protocol; type of fog servers, that is, service provider-controlled or private entities and base station communication range, that is, cell or microcell.

Among the protocols implemented, we observe that *'Secret Generation'*, requiring only three interacting entities and having lower protocol complexity as seen in Figure 13a consumes the lowest power, memory and execution time. It is followed by authentication and handover protocols in order as they are designed to be immune to Impersonation and Sybil attacks as discussed in Sections 4.4 and 4.7. *'Vote transfer using ECC'* consumes the highest execution time of 3.5029 ms. However, the time required by the devices and base stations to generate the keys and transfer the votes was observed to be 0.0099 ms. While the Cloud server required 3.4930 ms to receive and process the keys and messages. Hence, ECC was observed to be lightweight on resource-constrained devices.

Further, we observe that the *'Handover between Cells or a Cell and a Private Fog Network'* protocol require high power, memory and execution time due to four entities interacting using protocols with high complexity to verify device and visited base station identity while moving between cells as shown in Figure 9. However, we see an exception in the case of *'Handover between Microcells'*. As the radius of microcells is small and the communications are relayed through resource-constrained microcell towers, the handover protocol has been designed to verify the Unique ID and reputation of the device and permit its movement. Hence,

TABLE 11 | Performance Values of Secret Generation, Vote transfer using ECC, Handover and Authentication Protocols implemented on Raspberry Pi 4.

Parameter	Execution time (ms)	Memory (MB)	Power (W)
Vote transfer using ECC	3.5029	0.1	0.44
Secret generation	0.1540	0.15	0.45
Handover			
Between cells or a cell and a private fog network	0.4724	0.35	1.51
Between microcells	0.1835	0.2	0.91
Fog server is offline or blacklisted	0.2653	0.5	1.52
Authentication			
Cell	0.1646	0.15	0.83
Private fog networks (devices verified)	0.1626	0.2	0.6
Private fog networks (devices not verified)	0.1592	0.2	0.53

power, memory and execution time are lower than other handover protocols.

In the case of *'Secret Generation'* protocol, the device communicates with the cloud server to generate *Authentication secrets*. However, as the cloud server is designed only to verify device details and authorise the authentication server to share the *Authentication secrets*, the protocol consumes the lowest power, memory and execution time of 0.45 W, 0.15M·B and 0.154 ms, respectively.

Among the handover protocols implemented, *'Handover between Microcells'* and *'Handover when Fog Server is Offline or Blacklisted'* are designed to rapidly relay device communication to the home base station and the Cloud server, respectively, through the resource-constrained microcell towers. Hence, they have the lowest execution time. Although the devices alone were observed to consume 0.28 W and 0.05 MB, respectively, of power and memory to execute, the requirement of cloud servers and base stations to relay communication, verify devices and authenticate the requests within a short period results in higher overall battery and power consumption of 0.5 MB and 1.52 W, respectively.

In the case of *'Handover between Microcells'* protocol, as the home base station approves the handover request, authentication using $Seed_{Value}$ generated by the Cloud server is not required. Therefore, this protocol consumes lower power and memory of 0.91 W and 0.2 MB, respectively. As the protocol needs to be executed rapidly to lower the communication burden on the microcell towers, we design the devices only to transmit *h(Unique ID, reputation)*, nonce and timestamp as shown in Figure 10. As Unique ID is unique in each cell and reputation and timestamp are unique for each message request, the protocol ensures device authentication and non-repudiation as shown in the scyther verification report, Figure 15b.

To ensure message confidentiality, device authenticity and non-repudiation in the *'Handover when Fog Server is Offline or Blacklisted'* protocol, we design the devices to transmit *h(Random Value)* calculated using $Seed_{Value}$ shared by the Cloud server. Due to the absence of the home base station, the device transmits '*Request to Leave*' to the Cloud server through the microcell tower. Hence, this algorithm consumes high power and memory of 0.2653 W and 0.5 MB, respectively.

In the case of *'Authentication in Private Fog Networks (Devices are Verified)'* protocol using group authentication, devices generate $Seed_{Secret}$ and the private fog server calculates H given by Equations (3) and (10) respectively, and transmits them to the cloud server. The cloud server calculates H' using Equation (11) and sends it to the fog server for verification. Hence, this process consumes high power, memory and execution time of 0.6 W, 0.2 MB, 0.1626 ms as seen in Table 11. When devices are not verified, each device is authenticated individually based on *h(Random Value)* generated using $Seed_{Value}$ as seen in *'Authentication in Private Fog Networks (Devices not Verified)'* protocol. Hence, this process consumes 2.09% and 11.66% lower execution time and power to authenticate individual devices compared to the *'Authentication in Private Fog Networks (Devices are Verified)'* protocol. However, these values increase proportionally with additional devices in the private fog network.

Further, in the *'Authentication in Cell'* protocol, the device is designed to generate H using *Cloud Secret* as shown in Equation (8). Device details are verified by the authentication server *Cloud Secret*, which is transmitted to the base station to calculate H' and verify the device identity. Hence, the protocol consumes a high power and execution time of 0.83 W and 0.1646 ms, respectively. However, the memory consumed was 0.15 MB, slightly lower than that observed in private fog networks. Similarly, in the '*Handover between Cells or a Cell and a Private Fog Network*' protocol, device identity is verified at the visited base station and the authentication server using H calculated using Equation (12) and *h(Random Value)*. Hence, the protocol consumes the highest power, memory and execution time of 0.4724 ms, 0.35 MB and 1.51 W, respectively.

6.2 | Qos Prediction Using Deep Learning

Among the deep learning algorithms implemented, Simple autoencoder, comprising only two dense layers as shown in Table 8 consumes the lowest power, memory and execution time of 0.04W, 0.1MB and 0.065ms, respectively. This behaviour, seen in Table 12, is observed as the amount of power, memory and execution time consumed is proportional to the number and the complexity of tensor layers employed to build the model. Simple autoencoder is followed by multilayer autoencoder, MLP, LSTM and CNN. Considering the performance values such as lower power consumption, execution time and RMSE, simple autoencoder and multilayer autoencoders can be used to predict the bandwidth required to improve channel quality in the feedback-based mechanism.

MLP, being a feedforward neural network, does not backpropagate output to adjust weights assigned to hidden layers. As the equations to calculate the output are a linear multiplication of input and weight of hidden layer as described in [65], MLP

TABLE 12 | Performance of QoS prediction using deep learning algorithms.

Algorithm	Power (W)	Memory (MB)	Execution time (ms)	MAE	MSE	RMSE
LSTM	0.2	0.2	0.09	0.47	0.22	0.47
MLP	0.15	0.2	0.0024	7.054	49.764	7.054
CNN	0.29	0.5	0.1576	38.427	1476.664	38.427
Simple autoencoder	0.04	0.1	0.065	0.268	0.072	0.268
Multilayer autoencoder	0.06	0.1	0.0728	0.315	0.099	0.315

was observed to have the lowest execution time of 0.0024 ms. However, as output calculation involves the multiplication of high-dimensional matrices, MLP consumes a higher power of 0.15 W. Hence, it may be used to update device CoS based on device application and excess QoS requirements.

Further, as LSTM involves the usage of exponential moving weight average to decide the importance of data and uses *'Forget Gate'* to delete old data, it consumed high power and memory of 0.2 W and 0.2 MB with a moderate execution time and RMSE of 0.09 ms and 0.47, respectively. Hence, it may be implemented in cases where high power consumption is tolerable, such as analysing and detecting irregular QoS variations to differentiate between attacks and device internal faults. As seen in Table 12, simple autoencoder, multilayer autoencoder and MLP have the lowest MAE, MSE and RMSE. In contrast, CNN has the highest MAE, MSE and RMSE of 38.427, 1 476.664 and 38.427, respectively. CNN, primarily used for analysing images, overfits the non-image data as it constructs feature maps of input, analyses using kernels, and downsizes the output to the input size. Hence, CNN is not suitable for predicting device QoS.

6.3 | Channel Performance

Channel performance is measured in terms of Round-Trip Time (RTT), packet delivery ratio and throughput for varying numbers of authentication requests initiated by a single NB-IoT device in one *'Active'* state, as shown in Figure 17. RTT and throughput are measured for varying packet drop rates of 0, 0.25, 0.5, 0.75 and 1.

In Figure 17a, we observe that the rise in RTT is proportional to the increase in packet drop rate and authentication requests. For eight authentication requests, RTT is approximately 0.1589 s, 0.1689 s and 0.1799 s for packet drop rates of 0.25, 0.5 and 0.75, respectively, which is much less than the maximum latency limit of 1 s for NB-IoT [66]. For a packet drop rate of zero, RTT is 0.069 s for the varying number of authentication requests. Further, for a drop rate of 0.75, as the number of authentication requests reaches 9 and 10, RTT is 1 000 s, indicating the loss of all packets. Similar behaviour is observed for a packet drop rate of 1.

We plot the packet drop rate for varying authentication requests with the resulting throughput to be 13.51 packets/s in Figure 17b.

We infer that the packet drop rate reduces as authentication requests increase. This behaviour is observed because devices are only designed to transmit *'Request for Authentication'* along with *Cloud Secret* during the authentication process and only improve channel quality based on QoS feedback from fog servers but not respond back. Due to this, transmission channels and devices will not be overwhelmed with data packets with increased authentication requests. As NB-IoT devices are designed to provide on-demand service, they enter *'Active'* state to process authentication requests, after which they enter *eDRX* state, where devices only receive data but do not transmit it. Hence, in case of lower authentication requests, devices would have entered eDRX by the time re-transmission requests are received. As these re-transmission requests are not processed, packet drop rate is high. However, for increased authentication requests, the device remains in *'Active'* state longer, leading to the re-transmission of lost packets, thus reducing packet drop rate.

Figure 17c studies the channel throughput for varying authentication requests plotted for packet drop rates of 0, 0.25, 0.5, 0.75 and 1. As NB-IoT operates on low bandwidth, throughput decreases for increasing authentication requests as devices require higher execution time and memory to process them. For a drop rate of 0, throughput is constant at 14.285 packets/s. Throughput progressively decreases for increasing drop rates and reaches 0 packets/s for a drop rate of 0.75 and the number of authentication requests equal to ten, indicating the loss of all packets. Similarly, for a drop rate of 1, throughput is observed to be 0 packets/s.

As observed in Figure 17b, our model has a drop rate of 0.01 for ten authentication requests sent. This value is within the acceptable threshold of 1% packet drop rate for real-time applications [67]. For a drop rate of 0.01, throughput and RTT were observed to be 14.285 packets/s and 0.069 s, as seen in Figure 17a,c, respectively. RTT thus obtained is within the standard NB-IoT's RTT range of 0 to 100 ms [68]. Considering the NB-IoT packet size of 125 bytes as shown in Table 2, throughput of 14.285 packets/s would translate to 1.743 kbps, which is well within the range of 20 kbps downlink limit [69]. Hence, our model provides good performance under normal operation conditions.

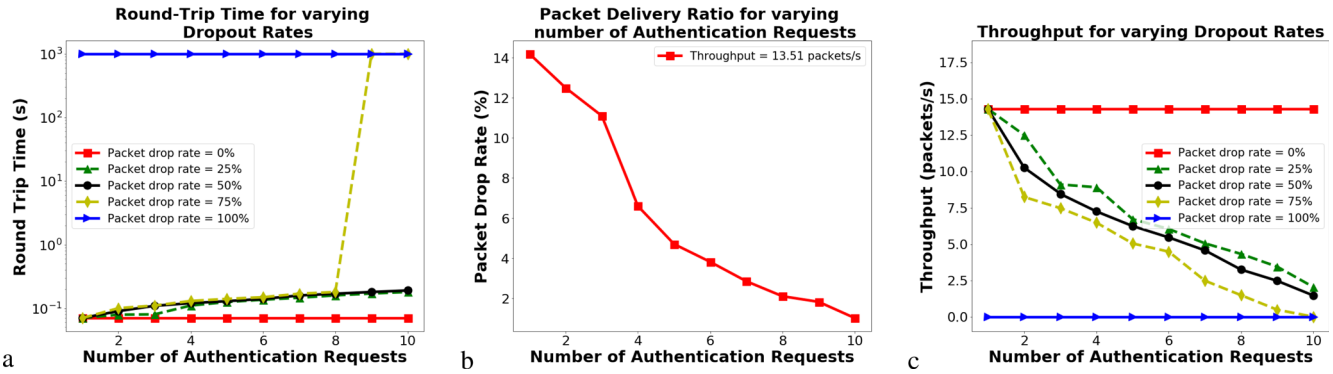

FIGURE 17 | Performance of fog computing architecture (a) Trade-off between round-trip time and packet drop rate, (b) Packet delivery ratio for varying authentication requests, (c) Trade-off between throughput and packet drop rate.

6.4 | IFogSim 2

Performance values of handover protocols, regular device operation and device clustering under a base station and private fog server to define cell limits and provide connectivity to devices in their proximity for varied NB-IoT device parameters, such as network usage execution time and battery simulated on iFogSim 2, are recorded in Table 13. Among the various algorithms implemented, we observe that clustering and regular device operation with both homogeneous and heterogeneous configurations require the lowest power, memory and network usage. We observe that heterogeneous configurations require approximately 0.81% higher CPU, battery and channel resources to operate compared to homogeneous configurations. This behaviour is observed because fog servers categorise QoS details specific to individual device configurations, determine CoS and individually study their system requirements to improve channel quality and distribute the load among the device, fog and cloud servers.

Among the clustering mechanisms, *'Device clustering in private fog servers'* requires slightly higher energy and network usage of 3.048 J and 1 06 800.5 Hz compared to *'Device clustering in base stations'*. This behaviour is observed as the private fog servers are required to verify the devices registered under it as the home base station and cluster them under its range. The rest of the devices are provided service by the service provider-controlled base station in that cell. On the contrary, the service provider-controlled base stations must constantly keep track of the devices entering and leaving the cell and cluster them to provide seamless service. Hence, execution time was observed to be 1 111 ms, which is slightly higher than that of the private fog servers with 1 104 ms.

From Table 13, we observe that among handover mechanisms, *'Device Handover across Microcells'* requires the lowest execution time, network and energy of 1 132 ms, 1 06 060.5 Hz and 3.029 J respectively, as the handover has to be implemented rapidly to reduce resource consumption on the resource-constrained microcell towers. This protocol is followed by *'Device Handover between Cells and Microcells'*, requiring approximately 9% higher resources. However, we observe that *'Device Handover between Cells'* and *'Device Handover between Cells and Private Fog Servers'* protocols consume the highest resources as they are designed to provide a secure handover, updated reputation and new Unique ID by the authentication servers.

6.5 | Comparison With Existing Approaches

In this subsection, we compare the performance of our approach with that of the existing works proposing fog computing frameworks operating on low bandwidth. These works focus on performance optimization while introducing low-bandwidth standards in the fog computing environment. However, authors neither predict device resources using machine learning nor propose protocols modifying the fog structure and communication mechanisms. Hence, we compare our works operating on normal bandwidth in terms of communication overhead representing protocol complexity and prediction error/classification accuracy of the machine learning and deep learning algorithms, as shown in Table 14.

Further, we simulate our protocols on a modified architecture without including fog servers. In such scenarios, devices communicate directly with the cloud server through the base stations, acting only as a relay to transmit messages, and the cloud server is designed to authenticate the devices and analyse QoS in real time. Communication overhead in [25, 26, 43, 44] is defined as the overhead (in bits) incurred in calculating nonce, timestamp, hashing algorithms, etc., at the device required to initiate handshake. While computational overhead is defined as the time taken (in ms) to XOR hashes, encrypt/decrypt messages with symmetric and asymmetric keys, generate hashes and generate random secrets. The approximate time required for various cryptographic functions is given in Table 15.

In our work, the communication overhead to calculate nonce, timestamp, $h(N, T)$ and unique ID during authentication or handover protocols totals 1 024 bits. This value is about 43.75% lower compared to those presented by Ali et al. [43], Kalaria et al. [44] and Guo et al. [25] requiring 3 344.475 bits, 1 505 bits and 1 472 bits, respectively. Similarly, the computational overhead of our approach calculated by $8T_h+5T_{Sec}+2T_{XOR}+9T_{ASE}+T_{SE}+T_{ECC}$,

TABLE 13 | Performance values of device clustering, normal fog operation and load sharing during handover implemented on iFogSim 2.

Parameter	Execution time (ms)	Network usage (Hz)	Energy (J)
Device clustering			
Base stations	1 111	1 06 404.0	3.012
Private fog servers	1 104	1 06 800.5	3.048
Configuration of devices in fog servers			
Homogeneous configuration	1 107	1 04 882.0	2.979
Heterogeneous configuration	1 116	1 06 533.5	3.008
Load sharing during device handover			
Across cells	1 187	1 08 002.5	3.085
Across microcells	1 132	1 06 060.5	3.029
Between cells and microcells	1 140	1 07 119.0	3.057
Across cells and private fog servers	1 182	1 07 898.5	3.026

TABLE 14 | Performance values of frameworks in proposed approaches.

Proposed approach	Execution time (s)	Power (W)	Memory (MB)	ML prediction error	ML classification accuracy (%)	Communication over head (bits)	Computational complexity	Computational overhead (ms)
				Normal bandwidth				
Guevara et al. [16]	0.035	—	—	—	100	—	—	—
Ali et al. [43]	—	0.528	—	—	—	2 816	$5T_{ecm}+1T_{eca}+1T_{fe}+26T_h$	528.475
Cui et al. [24]	0.062	—	—	—	—	—	—	—
Kalaria et al. [44]	—	—	—	—	—	1 504	$T_{ase}+T_{se}+2T_m+2T_h$	68
Guo et al. [25]	—	—	—	—	—	1 472	$14T_h+2T_{sp}$	1.498
Al-Mekhlafi et al. [26]	—	—	—	—	—	672	$(n+1)A_{ecc}+(n+5)M_{ecc}$	0.6749n+3.3621
				Data intensive/low bandwidth				
Amanlou et al. [23]	4.25	0.71	—	—	—	—	—	—
Peralta et al. [72]	—	0.528	—	—	—	—	—	—
Cheng et al. [70]	2.3	—	3.8	—	—	—	—	—
Ibrahim et al. [71]	1.058	—	—	—	—	—	—	—
Our work (with Fog servers)	0.1540	0.6	0.2	0.268	—	1024	$8T_h+5T_{Sec}+2T_{XOR}+9T_{ASE}+T_{SE}+T_{ECC}$	3.8422
Our work (without fog servers)	0.2865	1.04	0.25	0.268	—	1 024	—	—

TABLE 15 | Approximate Time required for Cyptographic Operations.

Notation	Operation	Approximate time (ms)	Notation	Operation	Approximate time (ms)
T_h	Hash function	0.005	T_{se}	Symmetric encryption/decryption	0.009
T_{ase}	Asymmetric encryption/decryption	0.006	T_{XOR}	XOR hashes	0.0001
T_{Sec}	Random secret generation	0.237	T_{ECC}	ECC execution	3.5029
T_{ecm}	ECC point multiplication	63.075	T_{eca}	ECC point addition	10.875
T_{fe}	Fuzzy extractor function	63.075	T_m	Message transmission	67.975
T_{SP}	Symmetric polynomial MAC	0.102	A_{ecc}	Point addition operation	0.0031
M_{ecc}	Point multiplication operation	0.6718	—	—	—

as seen in Table 14, amounts to 3.8422ms, where T_{ECC} being 3.5029ms forms the major factor.

We observe that the existing works operating on normal bandwidth applications have the lowest execution time of around 0.035s, which is around 77.27% lower than our approach. In contrast, the architectures working on low bandwidth, that is, Amanlou et al. [23], Cheng et al. [70] and Ibrahim et al. [71], are recorded to consume a higher execution time of 4.25 s, 2.3 s and 1.058 s, respectively, which is 6 times higher than that consumed by our framework of 0.154 s. However, power consumed by various approaches was observed to be similar in the range of 0.6 W, being approximately 12% higher than that of our architecture. Further, the memory required to execute the protocols proposed by Cheng et al. [70] was recorded to be 3.8 MB, which is approximately 16 times higher than that consumed by our approach of 0.2 MB.

As seen in Section 3.4, we utilise deep learning algorithms in regression mode to analyse real-time QoS data and predict the updated CoS and QoS suitable based on the device's application. Hence, we use prediction errors, that is, MSE, MAE and RMSE, to determine the error rate of the predicted values as opposed to the percentage accuracy used in the case of classification. Our work records an RMSE of 0.268 for the Simple autoencoder (having one hidden layer), indicating good prediction accuracy. Additionally, Guevara et al. [16] propose several neural network models with varying numbers of hidden layers to classify CoS. Hence, in Table 14, we have included the classification accuracy of 100% for the neural network having one hidden layer, which resembles the structure of the simple autoencoder implemented in our paper. Further, we see that our architecture, when implemented without fog servers, consumes 1.04 W, 0.25 MB and 0.3865 s of power, memory and execution time, respectively. These values are approximately 73.33%, 25% and 86.03% higher than that consumed by the architecture with fog servers, as shown in Table 14.

7 | Discussion

This section highlights the security challenges and regulatory issues in implementing fog computing architecture, particularly in NB-IoT. A detailed discussion of these issues is given below.

7.1 | Security Risks

Some security risks encountered while implementing fog computing architecture are authentication and trust issues, black holes, side channel attacks, DDoS, and node cloning. Of these, only the vulnerabilities arising due to authentication and trust issues are addressed in our proposed framework by implementing secret generation, authentication and handover protocols as described in Section 3.

In the case of black hole attacks, the attackers can send fake routes to the sender, trying to determine the best route to the receiver. The attacker nodes act as a sinks and route packets to their addresses to eavesdrop on or disrupt packet flow. These attacks can be prevented by routing verification, user authentication and trust-based mechanisms. In our framework, we use *Cloud Secret* and *Random Value* at the base stations and cloud server, respectively, to verify sender and receiver identity and ensure non-repudiation. Additionally, trust-based mechanisms in the form of device and base station reputation are implemented to analyse device behaviour, detect attack nodes and blacklist them. However, our design is still vulnerable to packet flow disruption as verification of routes is beyond the scope of our work.

Side-channel attacks are launched by attackers using the information leaked by IoTs, such as power consumption, timing information, data cache, etc., to gain access to passwords and cryptography keys. Of these possibilities, attackers will not be able to access data caches as the $Seed_{Secret}$, $f_{Secret}(\cdot)$, $f_{Value}(\cdot)$ and $U_{Secret}(\cdot)$ is generated and shared by the cloud server each time the device switches-on in its home base station. Additionally, $Seed_{Secret}$ is updated after every use to ensure unlinkability. However, the fog architecture is still susceptible to attacks caused by the abovementioned factors.

DDoS attacks caused by cryptographic weakness in protocols are limited by the use of N, T and $h(N, T)$ in handshake protocols as described in Section 4.6. However, there might arise situations where attackers launch variations of DDoS attacks [73] such as ICMP flood, Ping of death or SYN flood to overwhelm the network with excess ping or SYN requests for an undefined period. Additionally, in the case of Slowloris DoS attacks, attackers keep the channel busy for a long time by progressively launching more attack packets to avoid detection until the channel is entirely flooded. Designing our fog architecture to be resilient to these attacks, which are challenging to mitigate, is beyond the scope of our research.

Attackers introduce fault injection attacks by injecting malicious packets, inserting a trojan during IC manufacturing to cause device malfunction and manipulating the IC clock to disrupt the authentication process and access the device's secrets. Of these attacks, only DoS attacks caused due to malicious packet injection can be prevented, as discussed in Section 4. Further, node cloning attacks where attackers can either physically damage and retrieve device hardware or eavesdrop on data can be launched to impersonate or create multiple device identities. Of these, only eavesdropping can be prevented by the proposed fog architecture as discussed in Section 4. However, the proposed architecture is still vulnerable to other factors mentioned.

7.2 | Regulatory Issues

Some stakeholders involved in implementing a fog architecture to connect NB-IoT devices are government organisations, law enforcement and business investors. To provide a secure and scalable framework, base stations and data servers controlled by service providers and private individuals must seek clearance from government and law enforcement agencies to limit the possibility of fake base stations. Researchers must work with the stakeholders to understand the industry-specific regulations and continuously update the protocols per the international and national laws to make them resilient to the ever-growing security challenges. The researchers must also be liable for security violations and be responsible for regularly fine-tuning the protocols to enhance trust and data secrecy in the fog architecture.

7.3 | Feasibility of Edge Computing

Edge computing offers decentralised data processing capabilities similar to fog computing. Edge servers are located physically on the device or in proximity to a group of deployed devices. Hence, they are ideal for making rapid decisions in real-time applications. However, edge servers are designed to transmit unprocessed or partially processed data to the cloud for further analysis [74].

In our architecture, the fog servers installed on base stations analyse real-time CoS and QoS, determine the resource allocation and only transmit QoS to the cloud if abnormal behaviour is observed. The cloud compares the QoS with previous device data and classifies the abnormality as an attack or internal fault. Edge servers might be incapable of completely processing the data when placed at base stations. Hence, they may be unable to replace the fog servers. Additionally, considering the edge servers' placement nearer the device clusters, they may not be implementable on the base stations serving the cells with a few kilometers of radius. However, edge servers may be effectively used in place of private fog servers and microcell towers servicing the device requests within a small radius of private organisations and microcells, where edge servers verify the authenticity of devices by validating the hashes and transmitting real-time data directly to the cloud.

8 | Conclusion and Future Work

In this paper, we developed a fog computing framework to provide secure and seamless network connectivity across cells and microcells controlled by service providers and private individuals in mobile NB-IoT applications. We alleviate the limitations of fake base station connections by designing secret generation, authentication and handover protocols to be implementable irrespective of the hardware requirements varying across service providers. The device QoS parameters are set based on its application and updated regularly based on the device requirement or updated application. QoS was analysed in real-time using deep learning algorithms to improve channel quality and detect abnormal behaviour. A reputation-based mechanism is introduced to track the device and fog server behaviour and blacklist them if their reputation reduces below the threshold. Further, we

implemented the framework on iFogSim 2 and Raspberry Pi 4 and analysed energy-performance-security trade-offs with those of the existing works. Scyther was used to verify the security of the proposed framework formally. Our approach consumes an average of 12% and 43.75% lower power and communication overhead and approximately 16 times and 6 times lower memory and execution time compared to the existing solutions, thus making it a lightweight alternative. We propose extending our research to develop algorithms capable of predicting attacks in advance using QoS and game-theory models and implementing the necessary preventive measures for application-specific NB-IoT.

Author Contributions

Vamshi Sunku Mohan: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization. **Sriram Sankaran:** Conceptualization, Software, Writing - Review & Editing, Visualization, Supervision, Project administration. **Rajkumar Buyya:** Conceptualization, Writing - Review & Editing, Supervision, Project administration. **Krishnashree Achuthanv:** Supervision, Project administration.

Data Availability Statement

The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

References

1. "NB-IoT Deployment Guide to Basic Feature Set Requirements," https://www.gsma.com/iot/wp-content/uploads/2019/07/201906-GSMA-NB-IoT-Deployment-Guide-v3.pdf.

2. M. Chen, Y. Miao, Y. Hao, and K. Hwang, "Narrow Band Internet of Things," *IEEE Access 5,* (2017), https://doi.org/10.1109/ACCESS.2017.2751586.

3. A. K. Sultania, C. Blondia, and J. Famaey, "Optimizing the Energy-Latency Tradeoff in NB-IoT With PSM and eDRX," *IEEE Internet of Things Journal* 8, no. 15 (2021): 12436–12454, https://doi.org/10.1109/JIOT.2021.3063435.

4. "Narrowband IoT (NB-IoT)," https://www.u-blox.com/en/narrowband-iot-nb-iot.

5. "Advantages and Disadvantages of NB-IoT," https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-NB-IoT.html.

6. A. Choudhary, M. Govil, G. Singh, et al., "A Critical Survey of Live Virtual Machine Migration Techniques," *Journal of Cloud Computing* 6 (2017): 23, https://doi.org/10.1186/s13677-017-0092-1.

7. S. Charitha and S. Manishankar, "An Efficient Workflow Management Model for Fog Computing," https://doi.org/10.1109/ICIRCA51532.2021.9544673, in *2021 Third International Conference on Inventive Research in Computing Applications (ICIRCA*), (Coimbatore, India, 2021), 1323–1328.

8. H. A. Alharbi, T. E. H. Elgorashi, and J. M. H. Elmirghani, "Energy Efficient Virtual Machines Placement Over Cloud-Fog Network Architecture," *IEEE Access* 8 (2020): 94697–94718, https://doi.org/10.1109/ACCESS.2020.2995393.

9. "Fog Computing vs. Cloud Computing for IoT Projects," https://www.sam-solutions.com/blog/fog-computing-vs-cloud-computing-for-iot-projects/.

10. C.-L. Tseng and F. Lin, "Extending scalability of IoT/M2M platforms with Fog computing," 825–830 (2018), https://doi.org/10.1109/WF-IoT.2018.8355143.

11. H. K. Apat, B. Sahoo, P. Maiti, and P. Patel, "Review on QoS Aware Resource Management in Fog Computing Environment," https://doi.org/10.1109/iSSSC50941.2020.9358897, in *2020 IEEE International Symposium on Sustainable Energy, Signal Processing and Cyber Security (ISSSC*), (Gunupur Odisha, India, 2020), pp. 1–6.

12. V. Meena, M. Gorripatti, and T. Suriya Praba, "Trust Enforced Computational Offloading for Health Care Applications in Fog Computing," *Wireless Personal Communications* 119 (2021): 1369–1386, https://doi.org/10.1007/s11277-021-08285-7.

13. R. Mahmud, S. Pallewatta, M. Goudarzi, and R. Buyya, "IFogSim2: An Extended IFogSim Simulator for Mobility, Clustering, and Microservice Management in Edge and Fog Computing Environments," *Journal of Systems and Software* 190 (2022): 1–17, Elsevier Press, Amsterdam, The Netherlands, August.

14. "Raspberry Pi 4," https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

15. "The Scyther Tool," https://people.cispa.io/cas.cremers/scyther/.

16. J. C. Guevara, R. d. S. Torres, and N. L. S. da Fonseca, "On the Classification of Fog Computing Applications: A Machine Learning Perspective," *Journal of Network and Computer Applications* 159 (2020): 102596, https://doi.org/10.1016/j.jnca.2020.102596.

17. J. Uma, V. Ramasamy, and A. Kaleeswaran, "Load Balancing Algorithms in Cloud computing Environment – A Methodical Comparison," *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET 2014)* 3, no. 2 (2014): 79–82.

18. I. Ungurean and N. C. Gaitan, "Software Architecture of a Fog Computing Node for Industrial Internet of Things," *Sensors (Basel)* 21, no. 11 (2021): 3715, https://doi.org/10.3390/s21113715.PMID:34073598;PMCID:PMC8198567.

19. G. Peruzzi and A. Pozzebon, "Combining LoRaWAN and NB-IoT for Edge-to-Cloud Low Power Connectivity Leveraging on Fog Computing," *Applied Sciences* 12, no. 3 (2022): 1497, https://doi.org/10.3390/app12031497.

20. S. F. Abedin, A. K. Bairagi, M. S. Munir, N. H. Tran, and C. S. Hong, "Fog Load Balancing for Massive Machine Type Communications: A Game and Transport Theoretic Approach," *IEEE Access* 7 (2019): 4204–4218, https://doi.org/10.1109/ACCESS.2018.2888869.

21. F. Murtaza, A. Akhunzada, S. u. Islam, J. Boudjadar, and R. Buyya, "Qos-Aware Service Provisioning in Fog Computing," *Journal of Network and Computer Applications* 165 (2020): 102674, https://doi.org/10.1016/j.jnca.2020.102674.

22. M. Goudarzi, H. Wu, M. Palaniswami, and R. Buyya, "An Application Placement Technique for Concurrent IoT Applications in Edge and Fog Computing Environments," *IEEE Transactions on Mobile Computing* 20, no. 4 (2021): 1298–1311, https://doi.org/10.1109/TMC.2020.2967041.

23. S. Amanlou, M. K. Hasan, and K. A. A. Bakar, "Lightweight and Secure Authentication Scheme for Iot Network Based on Publish–Subscribe Fog Computing Model," *Computer Networks* 199 (2021): 108465, https://doi.org/10.1016/j.comnet.2021.108465.

24. M. Cui, D. Han, and J. Wang, "An Efficient and Safe Road Condition Monitoring Authentication Scheme Based on Fog Computing," *IEEE Internet of Things Journal* 6, no. 5 (2019): 9076–9084, https://doi.org/10.1109/JIOT.2019.2927497.

25. Y. Guo and Y. Guo, "FogHA: An Efficient Handover Authentication for Mobile Devices in Fog Computing," *Computers & Security* 108 (2021): 102358, https://doi.org/10.1016/j.cose.2021.102358.

26. Z. G. Al-Mekhlafi, M. A. Al-Shareeda, S. Manickam, et al., "Efficient Authentication Scheme for 5G-Enabled Vehicular Networks Using

Fog Computing," *Sensors* 23, no. 7 (2023): 3543, https://doi.org/10.3390/s23073543.

27. S. Singh and V. K. Chaurasiya, "Mutual Authentication Scheme of Iot Devices in Fog Computing Environment," *Cluster Computing* 24 (2021): 1643–1657, https://doi.org/10.1007/s10586-020-03211-1.

28. G. Jia, G. Han, A. Li, and J. Du, "SSL: Smart Street Lamp Based on Fog Computing for Smarter Cities," *IEEE Transactions on Industrial Informatics* 14, no. 11 (2018): 4995–5004, https://doi.org/10.1109/TII.2018.2857918.

29. D. Zhao, Q. Zou, and M. Boshkani Zadeh, "A QoS-Aware IoT Service Placement Mechanism in Fog Computing Based on Open-Source Development Model," *Journal of Grid Computing* 20 (2022): 12, https://doi.org/10.1007/s10723-022-09604-3.

30. K. N. Pallavi and V. Ravi Kumar, "Authentication-Based Access Control and Data Exchanging Mechanism of Iot Devices in Fog Computing Environment," *Wireless Personal Communications* 116 (2021): 3039–3060, https://doi.org/10.1007/s11277-020-07834-w.

31. H. Zahmatkesh and F. Al-Turjman, "Fog Computing for Sustainable Smart Cities in the Iot Era: Caching Techniques And Enabling Technologies - An Overview," *Sustainable Cities and Society* 59 (2020): 102139, https://doi.org/10.1016/j.scs.2020.102139.

32. B. Qin, X. Lin, F. Zheng, X. Chen, C. Huang, and R. Pan, "Research and Application of Key Technology of NB-IoT Based on Fog Computing," https://doi.org/10.1109/CCET48361.2019.8989397, in *2019 IEEE 2nd International Conference on Computer and Communication Engineering Technology (CCET)*, (Beijing, China, 2019), pp. 291–296.

33. P. Prakash, K. G. Darshaun, P. Yaazhlene, M. Ganesh, and B. Vasudha, "Fog Computing: Issues, Challenges and Future Directions," *International Journal of Electrical and Computer Engineering* 7 (2017): 3669–3673, https://doi.org/10.11591/ijece.v7i6.pp3669-3673.

34. T. N. Gia, L. Qingqing, J. P. Queralta, Z. Zou, H. Tenhunen, and T. Westerlund, "Edge AI in Smart Farming IoT: CNNs at the Edge and Fog Computing with LoRa," https://doi.org/10.1109/AFRICON46755.2019.9134049, in *2019 IEEE AFRICON*, (Accra, Ghana, 2019), 1–6.

35. Z. Wang, M. Goudarzi, M. Gong, and R. Buyya, "Deep Reinforcement Learning-based scheduling for optimizing system load and response time in edge and fog computing environments," *Future Generation Computer Systems* 152 (2024): 55–69, https://doi.org/10.1016/j.future.2023.10.012.

36. J. Bachiega, B. Costa, L. R. Carvalho, M. J. F. Rosa, and A. Araujo, "Computational Resource Allocation in Fog Computing: A Comprehensive Survey," *ACM Computing Surveys* 55, no. 14s (2023): 336, https://doi.org/10.1145/3586181.

37. M. H. Kashani and E. Mahdipour, "Load Balancing Algorithms in Fog Computing," *IEEE Transactions on Services Computing* 16, no. 2 (2023): 1505–1521, https://doi.org/10.1109/TSC.2022.3174475.

38. S. Tuli, F. Mirhakimi, S. Pallewatta, et al., "AI Augmented Edge and Fog Computing: Trends and challenges," *Journal of Network and Computer Applications* 216 (2023): 103648, https://doi.org/10.1016/j.jnca.2023.103648.

39. S. Iftikhar, S. S. Gill, C. Song, et al., "Ai-Based Fog and Edge Computing: A Systematic Review, Taxonomy and Future Directions," *Internet of Things* 21 (2023): 100674, https://doi.org/10.1016/j.iot.2022.100674.

40. S. Lu, J. Wu, N. Wang, et al., "Resource Provisioning in Collaborative Fog Computing for Multiple Delay-Sensitive Users," *Software: Practice and Experience* 53, no. 2 (2023): 243–262, https://doi.org/10.1002/spe.3000.

41. J. Taghizadeh, M. Ghobaei-Arani, and A. Shahidinejad, "A Metaheuristic-Based Data Replica Placement Approach for Data-Intensive Iot Applications in the Fog Computing Environment," *Software: Practice and Experience* 52, no. 2 (2022): 482–505, https://doi.org/10.1002/spe.3032.

42. O. A. Khan, S. U. R. Malik, F. M. Baig, et al., "A Cache-Based Approach Toward Improved Scheduling in Fog Computing," *Software: Practice and Experience* 51 (2021): 2360–2372, https://doi.org/10.1002/spe.2824.

43. Z. Ali, S. A. Chaudhry, K. Mahmood, S. Garg, Z. Lv, and Y. B. Zikria, "A Clogging Resistant Secure Authentication Scheme for Fog Computing Services," *Computer Networks* 185 (2021): 107731, https://doi.org/10.1016/j.comnet.2020.107731.

44. A. S. M. Rudri Kalaria, W. R. Kayes, and E. Pardede, "A Secure Mutual Authentication Approach to Fog Computing Environment," *Computers & Security* 111 (2021): 102483, https://doi.org/10.1016/j.cose.2021.102483.

45. B. A. Mohammed, M. A. Al-Shareeda, S. Manickam, Z. G. Al-Mekhlafi, A. M. Alayba, and A. A. Sallam, "ANAA-Fog: A Novel Anonymous Authentication Scheme for 5G-Enabled Vehicular Fog Computing," *Mathematics* 11, no. 6 (2023): 1446.

46. B. A. Mohammed, M. A. Al-Shareeda, S. Manickam, et al., "FC-PA: Fog Computing-Based Pseudonym Authentication Scheme in 5g-Enabled Vehicular Networks," *IEEE Access* 11 (2023): 18571–18581.

47. "Shannon's Theorem," http://www.inf.fu-berlin.de/lehre/WS01/19548-U/shannon.html.

48. M. V. Ramesh and M. S. Nisha, "Design of medium independent handover based emergency communication scheme using vehicular technology," https://doi.org/10.1109/ICCSN.2011.6014286, in *2011 IEEE 3rd International Conference on Communication Software and Networks*, (Xi'an, China, 2011), 355–358.

49. "Analyzing NB IoT and LoRaWAN Sensor Battery Life," https://tech-journal.semtech.com/analyzing-nb-iot-and-lorawan-sensor-battery-life.

50. "NB-IoT - Data Rates and Latency," https://www.netmanias.com/en/post/blog/12609/iot-nb-iot/nb-iot-data-rates-and-latency.

51. "NB-IoT Frequency Bands," https://www.everythingrf.com/community/nb-iot-frequency-bands.

52. "Cloud-Fog Computing Dataset," https://www.kaggle.com/datasets/sachin26240/vehicularfogcomputing.

53. P. Shabisha, A. Braeken, P. Kumar, and K. Steenhaut, "Fog-Orchestrated and Server-Controlled Anonymous Group Authentication and Key Agreement," *IEEE Access* 7 (2019): 150247–150261, https://doi.org/10.1109/ACCESS.2019.2946713.

54. H. Goswami and H. Choudhury, "Remote Registration and Group Authentication of IoT Devices in 5G Cellular Network," *Computers & Security* 120 (2022): 102806, https://doi.org/10.1016/j.cose.2022.102806.

55. M. A. Mobarhan and M. Salamah, "REPS-AKA5: A Robust Group-Based Authentication Protocol for IoT Applications in LTE System," *Internet of Things* 22 (2023): 100700, https://doi.org/10.1016/j.iot.2023.100700.

56. M. Kolberg, E. Magill, and M. Wilson, "Compatibility Issues Between Services Supporting Networked Appliances," *IEEE Communications Magazine* 41 (2003): 136–147, https://doi.org/10.1109/MCOM.2003.1244934.

57. "Secrets - Generate secure random numbers for managing secrets," https://docs.python.org/3/library/secrets.html.

58. A. Portilla-Figueras, S. Salcedo-Sanz, K. D. Hackbarth, et al., "Novel Heuristics for Cell Radius Determination in WCDMA Systems and Their Application to Strategic Planning Studies," *Journal on Wireless Communications and Networking* 2009 (2009): 314814, https://doi.org/10.1155/2009/314814.

59. "AT&T microcell to be replaced with cell phone signal boosters," https://cellphoneboosterstore.com/tech-news-and-blog/att-microcell-to-be-replaced-with-cell-phone-signal-boosters/.

60. D. Hankerson and A. Menezes, "Elliptic Curve Cryptography," in *Encyclopedia of Cryptography, Security and Privacy*, eds. S. Jajodia,

P. Samarati, and M. Yung (Berlin, Heidelberg: Springer, 2021), https://doi.org/10.1007/978-3-642-27739-9_245-2.

61. "Open Addressing Collision Handling technique in Hashing," https://www.geeksforgeeks.org/open-addressing-collision-handling-technique-in-hashing/.

62. Open SSL, "Cryptography and SSL/TLS Toolkit," https://www.openssl.org/.

63. A. S. Farhan and C. R. Kavitha, "End-to-End Encryption Scheme for IoT Devices Using Two Cryptographic Symmetric Keys," *International Journal of Control Theory and Applications (IJCTA)* 9 (2016): 43–49.

64. "B2901A Precision Source/Measure Unit," https://www.keysight.com/in/en/product/B2901A/precision-source-measure-unit-1-ch-100fa-210v-3a-dc-10-5a-pulse.html.

65. "Multi-Layer Neural Network," http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/0.

66. "NB-IoT explained: a complete guide to Narrowband-IoT," https://www.i-scoop.eu/internet-of-things-iot/lpwan/nb-iot-narrowband-iot/.

67. "What Is Network Packet Loss?," https://www.ir.com/guides/what-is-network-packet-loss.

68. "What Is RTT in Networking?," https://aws.amazon.com/what-is/rtt-in-networking/.

69. "What is narrowband IoT (NB-IoT)?," https://www.techtarget.com/whatis/definition/narrowband-IoT-NB-IoT.

70. B. Cheng, J. Fuerst, G. Solmaz, and T. Sanada, "Fog Function: Serverless Fog Computing for Data Intensive IoT Services," https://doi.org/10.1109/SCC.2019.00018, in *2019 IEEE International Conference on Services Computing* (*SCC*) (Milan, Italy, 2019), 28–35.

71. A. H. Ibrahim, Z. T. Fayed, and H. M. Faheem, "Fog-Based CDN Framework for Minimizing Latency of Web Services Using Fog-Based HTTP Browser," *Future Internet* 13, no. 12 (2021): 320, https://doi.org/10.3390/fi13120320.

72. G. Peralta, M. Iglesias-Urkia, M. Barcelo, R. Gomez, A. Moran, and J. Bilbao, "Fog computing based efficient IoT scheme for the Industry 4.0," https://doi.org/10.1109/ECMSM.2017.7945879, in *2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics* (*ECMSM*), (Donostia, Spain, 2017), 1–6.

73. R. J. Alzahrani and A. Alzahrani, "A Novel Multi Algorithm Approach to Identify Network Anomalies in the IoT Using Fog Computing and a Model to Distinguish Between IoT and Non-IoT Devices," *Journal of Sensor and Actuator Networks* 12, no. 2 (2023): 19, https://doi.org/10.3390/jsan12020019.

74. "Edge Computing vs. Fog Computing: 10 Key Comparisons," https://www.spiceworks.com/tech/cloud/articles/edge-vs-fog-computing/.