electronics

MDPI

Article

RESFIT: A Reputation and Security Monitoring Platform for IoT Applications

Ștefan-Ciprian Arseni [1,2], Bogdan-Cosmin Chifor [1], Mihai Coca [1,2], Mirabela Medvei [1], Ion Bica [1,*] and Ioana Matei [1]

[1] Faculty of Information Systems and Cyber Security, "Ferdinand I" Military Technical Academy, 050141 Bucharest, Romania; stefan.arseni@mta.ro (Ș.-C.A.); bogdan.chifor@mta.ro (B.-C.C.); mihai.coca@mta.ro (M.C.); mirabela.medvei@mta.ro (M.M.); ioana.matei@mta.ro (I.M.)

[2] Faculty of Electronics, Telecommunications and Information Technology, University Politehnica of Bucharest, 060042 Bucharest, Romania

* Correspondence: ion.bica@mta.ro

Abstract: The fast-paced adoption of smart devices has changed the Internet of Things (IoT) landscape, leading to the growth of smart environments and inclusion in many aspects of our society. In IoT applications, data collected from sensors and mobile devices are aggregated, processed, and analyzed to extract useful information and develop intelligent services. If the collected data is not trustworthy due to the damage or malicious input of some sensors, the quality of the service will be impacted. For reliable data collection and mining, it is mandatory to define robust security and trust models, suitable for the IoT application context. In this paper, we propose RESFIT, a platform that implements a reputation-based trust mechanism and an advanced application level firewall to cope with the above mentioned issues. Having a gateway-centric architecture, the proposed platform ensures minimal resource consumption at the node layer, and an integrated overview and control of the system state, through the cloud component and smartphone management application.

Keywords: IoT security; reputation-based trust evaluation; MQTT application level firewall; remote command authorization.

Citation: Arseni, Ș.-C.; Chifor, B.-C.; Coca, M.; Medvei, M.; Bica, I.; Matei, I. RESFIT: A Reputation and Security Monitoring Platform for IoT Applications. *Electronics* **2021**, *10*, 1840. https://doi.org/10.3390/electronics10151840

Academic Editors: Sang-Soo Yeo and Damien Sauveron

Received: 29 June 2021
Accepted: 28 July 2021
Published: 31 July 2021

1. Introduction

The IoT will be a key enabling technology to the development of smart environments that will revolutionize the way we live, do business, and manage critical infrastructures. In an IoT environment, a variety of independent devices interact on a collaborative basis to perform different tasks. Data collected from these devices is aggregated, processed, and analyzed to extract useful information and develop intelligent services [1]. This computational model introduces new challenges due to its particular features. Apart from privacy and security, the most critical one is trust. If the aggregated information from different devices is not trustworthy, due to the damage or malicious input of some devices, it is difficult for users to accept that information.

Traditional trust evaluation solutions face difficulties when applied to IoT, due to the different standards, communication stacks, and low computation resources of the entities [2]. Therefore, a flexible approach is needed to be able to deal with these issues in such a complex environment. To overcome this issue, researchers' attention focused on using reputation-based trust models adapted for IoT. As defined in [3], reputation is a measure that is derived from direct or indirect knowledge or experiences on earlier interactions of entities and is used to assess the level of trust an entity puts into another entity. The concept of reputation has been proven useful in many areas, particularly in the context of distributed and collaborative systems, which share many characteristics with IoT.

In this paper, we propose RESFIT, a platform with a multi-layer architecture, addressing the trust evaluation of sensing devices, based on reputation scores calculated

using a Naive Bayes algorithm. The proposed platform consists of interconnected modules that are integrated at each layer of an IoT system: cloud, gateway and device. RESFIT is built on a customized decentralized architecture, empowering middle-layer devices, such as gateways, while having a central point of management through a Cloud component and a smartphone application. Another important capability of our platform is the Message Queuing Telemetry Transport (MQTT) application layer firewall that helps mitigate Denial-of-Service (DoS) attacks.

The platform architecture was also one of our research topics of [4–6], but, while our previous work was focused more on the design of the platform, the current effort is focused on implementing and analyzing the challenges of deploying the system in a real environment. Having a gateway-centric architecture, RESFIT ensures minimal resource consumption at the device layer and good response time in detecting malfunctions or malicious actions. The platform is suited for edge/fog computing IoT architectures and allows building flexible sensing applications [7].

The rest of the paper is structured as follows. Section 2 presents related work regarding trust management in IoT. Then, we present the design and implementation details of our platform in Section 3. In Section 4, we present the testing environment and discuss the evaluation results. Section 5 ends the paper with conclusions and future research directions.

2. Related Work

Trust assessment is a complex endeavor with a multi-faceted assembly with no definitive consensus, integrating multiple measurable and non-measurable variables. Reputation ratings of individual parties in distributed environments are commensurate with trust effects in functional approaches for improving security and promotion of collaboration among nodes. In order to evaluate the trust of an entity, a series of metrics and attributes are interpreted on the basis of reputation, experience and knowledge.

Many reputation systems use the weighted sum techniques to aggregate direct trust (self-observations) with indirect trust (ratings, feedback or recommendations). As many approaches usually offer rigid and inflexible mechanisms to compute reputation scores, the authors in [8] designed a dynamic adaptation to current heterogeneous IoT environments by prototyping a flexible mechanism to select the most suitable trust and reputation model. In the long term, this pliable mechanism proved to output more accurate reputation values than those estimated by static reputation models.

Trust management plays a critical role in IoT for granting solid data mining and gathering context-aware intelligence for user privacy and information security benefits. Commonly, a reputation model in a trust management scheme is a probabilistic and statistical model, where Beta distribution is the most used method. The Reputation-based Framework for high integrity Sensor Networks (RFSN) [9] is one of the first models of this type where each node maintains trust values about other nodes using direct observations about its neighbors and second-hand information from other nodes' observations. RFSN allows only positive observations to be propagated, making the bad mouthing attack impossible. However, this model is unable to detect on-off attacks. In [10], a Beta distribution-based Trust and Reputation Evaluation System (BTRES) is proposed to solve the vulnerable state generated by compromised nodes' attacks. The authors used the beta function to calculate the trust value, based on the interaction information between the nodes. However, the proposed system does not take into account energy consumption when estimating the trust of an IoT device. As BTRES is vulnerable to internal attacks like DoS and data tampering attacks, research efforts [11–13] were conducted in solving these concerns. In [11], authors developed a Beta and link quality indicator-based trust model (BLTM), which additionally employs an analysis mechanism to maintain the accuracy and stability of the trust value of nodes. A trust model using binomial distribution for computing the node's trust value integrated with a secure routing protocol used for security stability, transmission performance and energy efficiency is defined in [12]. Binomial distribution describes an interaction between two nodes involving cooperation or non-

cooperation. The Dirichlet Distribution-based Trust Management Scheme (DDTMS) [13] was proposed exclusively to defend against the internal attacks and demonstrated better results than Beta and Gaussian distribution reputation models. The Dirichlet distribution is a multivariate version of the Beta distribution, modeling events with more than two distinct results, unlike the Beta distribution, which takes only two distinct results.

An Adaptive Trust Estimation Scheme (ATES) was suggested in [14], which combines the communication history and stereotypical reputation of an IoT device by using both personal and non-personal trust values. When a user-device interaction happens, the user's personal trust value is calculated by mapping the current situation of the device and experience history extraction for the same types of devices into a M5 tree regression model. Direct observations can adjust the trust of a node by evaluating the services provided by it in terms of time. This model performs well to assess selective attacks in a trust management model, but it augments the occasions for bad-mouthing attacks.

The highly dynamic nature of the IoT environment requires a general and flexible architecture, capable of managing trust and reducing challenges associated with heterogeneity and scalability. IoTrust architecture [15], a Soft Defined Network (SDN) with a cross-layer authorization protocol integration proposes two reputation evaluation schemes to establish the trust, at node level, respectively, at organization level. Although it achieves high detection accuracy of attacked nodes, this scalable architecture is incapable in detection of malicious user and organization behaviors.

Contribution trust has been studied in the participatory sensing systems [16–18] in order to maintain high-quality data. The Reputation System to Evaluate Participants (RSEP) methodology [16] validates node contributions using their reputation values. The system clusters participants into groups based on the similarities of the contribution values. Then, the highest group weight, calculated on the basis of members' reputation values, decides the most accurate contributions. Ultimately, the RSEP scheme filters the sensed data to determine the winners and the losers of the participatory sensing applications. Without assigning a reward mechanism in reputation computation, data quality and reputation history composite produces lower reputation values than what deserved for only a few faulty contributions. Authors of [17] have proposed a trust-based framework, called FIDES, to measure contributions accuracy. Data reliability is guaranteed through deployment of mobile security agents (MSAs), which gather surrounding information. A participant's reliability is estimated using the frequency of transmission of reports. However, the necessity to set a large number of system parameters makes the actual performance of the FIDES framework difficult to predict in reality. Anonymity and trust, two conflicting objectives in participatory sensing networks, are consistently aligned in ARTSense [18], a reputation-based framework which maintains positive and negative reputation updates by protecting the anonymity of the sender. As maintaining anonymity is a principal pillar in this framework, there is a lack in detecting the source of attacks.

Even though all the systems described above implement security through reputation-based mechanisms, their proposed assumptions may, sometimes, be insufficiently general and unfeasible in a specific environment.

For a reputation-based application to enforce trust in devices, complementary security mechanisms must be defined, in order to ensure the consistency of actions with good intentions that the reputation module requires. Policy enforcement mechanisms can be used to define the rules that will enable a consistent data flow in the system. On this note, authors of [19] have proposed a Model-based Security Toolkit, named SecKit, for the enforcement of security policy rules at the MQ Telemetry Transport (MQTT) protocol layer to support security and privacy requirements. Authorizations and obligations are identified and a dedicated module, called Policy Enforcement Point, acts as a connector that intercepts the messages exchanged by the broker using the publish–subscribe mechanism and generates proper notifications that might lead to the execution of an enforcement action (i.e., allow, deny, modify, and delay). The main drawback of SecKit is the high overhead

introduced when one publisher has many interested subscribers, and a policy needs to be checked for every subscriber.

The majority of IoT systems are comprised, mainly, from three layers, perceptual–transport–application, in which data is collected, transmitted to a certain sink and, finally, intelligently processed and presented in accordance to users' requirements. Given the needs it covers, cloud is a suitable third layer of an IoT system, by providing means for data-driven decision making services. CloudIoT [20] is a novel paradigm where cloud and IoT are merged together. Despite these advantages, the requirement of logically linking each device in the IoT system to a central point that cannot be fully controlled by the IoT system' administrator, using cloud as the single third-layer component may not always be the best direction. To this end, a context-aware multi-source trust evaluation model interleaved with a reputation system evaluates the trustworthiness of a user in a Fog based IoT (FIoT) [21]. A monitor assistance module keeps track of untrusted users' behavior in order to identify malicious users before they start a communication. Although historical experiences help to incorporate indirect trust in FIoT, still a node could provide fabricated trust value for other nodes in order to influence the reputation. Similar to this, the security platform presented in this paper is designated as a gateway-centric platform, enabling the use of cloud services for specific tasks.

RESFIT platform takes advantage of the solutions presented above and offers stability and reliability in terms of latency constraints in the context of tracking and computing reputation scores for IoT devices. For reputation-aware recommendation, we propose a reputation model that calculates reputation scores by weighting trust both from the system state and from the sensor data. Furthermore, our platform enables the use of policy enforcement mechanisms, by implementing a dynamic firewall, that not only enforces the stability of communication flows, but can also adapt to specific requests that may appear in the operating cycle of a device. For instance, when approaching the battery limit, a device may require to prolong its uptime and battery, by processing fewer requests or publishing less data.

3. Platform Design and Implementation

3.1. Overall Architecture

The IoT reputation and security monitoring platform proposed in this paper is a three tier system consisting of a client agent, a gateway software and a cloud application. This platform permits, through the cloud application, the management of an IoT device fleet, by allowing the administrator to control and visualize the state of the IoT system composed of client IoT devices and gateways. The RESFIT platform includes a smartphone management application, that allows segregating the role of system owner and administrator. Using the smartphone application, the system owner can approve the commands received by an IoT device from the cloud layer (managed by the administrator).

RESFIT is a gateway-centric platform, the gateway module having the primary task of aggregating telemetry information from the IoT devices and computing the reputation of each client. The cloud application aggregates the information from both gateways and IoT devices, allowing an administrator to inspect, using a web application, the telemetry information and the reputation level for each IoT client. Furthermore, the web management application allows the administrator to send actuation commands to the IoT devices. RESFIT is not designed to replace a functional IoT platform, its main purpose being to complement it and focus strictly on security functionalities like monitoring and computing reputation. Thus, the platform can be easily integrated with third-party IoT platforms or proprietary applications by using a simple application programming interface (API) on the IoT device level. The client module is a lightweight agent that runs on the IoT device and communicates with a data plane (third-party application) using an API. The data plane application can use the client agent to publish reputation related data and fetch the most trusted information from the gateway, the agent bridging the communication between the data plane application and the gateway. Besides the reputation related tasks, the client

agent extracts various telemetry information from the IoT device and synchronizes it with the gateway in a real-time manner.

The gateway is a component deployed on the IoT network edge, which aggregates information from all the client agents installed in the network. This module is tightly coupled with the data plane network component of the IoT platform by communicating with a publish–subscribe broker, which supports the MQTT protocol. This allows the gateway to implement an application layer firewall for the MQTT protocol, by fetching firewall rules from the client IoT device and deploying them on the edge.

The gateway computes a reputation score for each IoT client device and permits other client agents to query the most trusted device from the network. Given this, if an IoT client device publishes qualitative data to the network and if it has a reliable and trustful behavior, then its reputation score will increase on the gateway side. It is through this mechanism that an administrator can configure the overall platform to monitor and respond to context-related metrics, to a certain degree. This idea is further detailed in Section 3.6, in which we present the inner structure of the reputation computation module.

The cloud application synchronizes data in real-time from all the registered gateways and displays the information to a web application, allowing the administrator to inspect the state of the system using charts, IoT device parameter information and IoT device/gateway connection health status.

The smartphone management application receives the commands to be approved from the gateway module, which buffers the unauthorized commands. Once a command is approved, the gateway sends the information to the client IoT device. For the approval protocol, the smartphone and the gateway establish an end-to-end trust relation and the smartphone cryptographically signs the approved command. Figure 1 depicts the general architecture of the IoT reputation platform.

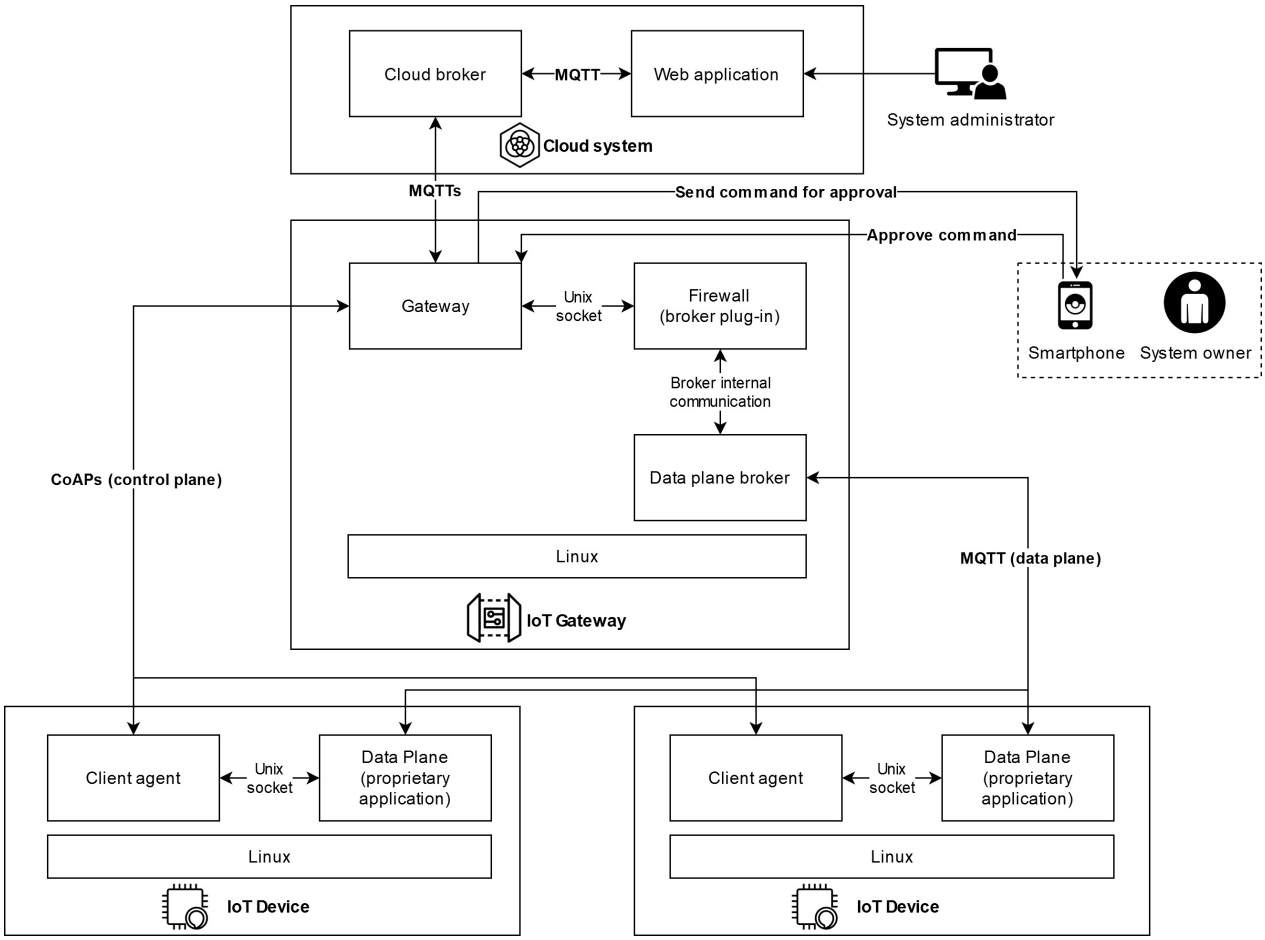

Figure 1. General architecture of the IoT reputation platform.

Data plane applications deployed on the end-point devices are communicating using a MQTT broker which is deployed on the same edge device as the gateway module. This network architecture is a prerequisite of the IoT reputation platform and targets one of the most used IoT communication design patterns: the publish–subscribe paradigm.

3.2. Client Module

The client module runs on the IoT device, and it is tightly coupled with a third-party data plane application. The client is a lightweight Linux agent that has two important tasks: collecting various system telemetry information from the IoT device and reporting it to the gateway, along with exposing a reputation and firewall API to the data plane application. The purpose of collecting system telemetry information is to provide data support for the higher layer modules (gateway, cloud) to detect security anomalies or malfunctioning scenarios (e.g., high memory usage, high CPU usage, high number of processes). Another important capability of the IoT reputation platform is the MQTT application layer firewall. This feature addresses the specific scenario of a resource constrained IoT device, which needs to react to a DoS attack and discard malicious MQTT packets. In order to save CPU cycles and battery consumption, the data plane application delegates this filtering task to the gateway module, using the client agent API. Besides the system telemetry information, the client agent also collects data plane network telemetry information. This mechanism is implemented by inserting software probes in the network ingress part of the data plane application. This reflects in real-time the number of packets per minute and the length of the received packets, being a method to validate the result of enforcing a certain firewall rule on the gateway side or to detect ingress traffic anomalies.

The client agent API exposes a set of functions that allows the data plane application to execute reputation operations: get the most trusted device within a sensor category, provide feedback for a certain IoT device and broadcast a sensing query in the IoT network.

The communication between the client agent and the gateway is a critical part of the IoT reputation platform, given that several conditions must be met: the client agent samples frequently telemetry information and uploads it to the gateway, the communication has to be fast and lightweight in terms of protocol overhead and most importantly both the gateway and the IoT device might become offline. In these conditions, the reporting protocol must handle failures and must permit an asynchronous communication between the client and the gateway, where the gateway must also have the capability to send messages to the client without a prior request. In order to achieve this, a CoAPs based reporting protocol was implemented. By doing this, we ensured a separation between the behavior of data and control plane: while MQTT is widely used for transferring data from nodes to sink-gateways, it is bound to use IP networks, thus limiting the capability of monitoring sensors installed in non-IP networks; CoAP removes this limitation, enabling our platform to connect to IoT nodes that exist both in IP and non-IP networks. Furthermore, the RESFIT platform does not offer a specific dataplane application (only a testing one), but, instead, connects to the one that exists on the node. The client agent communication consists of two protocols: the reporting protocol—with the client agent as CoAP client and the gateway as CoAP server and the commands protocol—with the client agent as CoAP server and the gateway as CoAP client. While the reporting protocol has the purpose of transmitting telemetry data to the gateway, the commands protocol has the purpose of transmitting asynchronous commands from the gateway to the client. Both of these protocols are secured using the DTLS-PSK mechanism, which is using the client agent identifier and pre-shared key (PSK). When first started, the client agent software randomly generates an identifier and a PSK, which are shared with the gateway using an out-of-band mechanism (a generic form of this mechanism will be a procedure in which the administrator manually copies the PSK generated at the client level and inserts it in the gateway database, using the *gateway_cli* tool available with the RESFIT platform). Thus, when a packet is received by the gateway, the DTLS-PSK protocol is executed with the identity presented by the client and the PSK stored on the gateway side. The reporting protocol also has a keep-alive role,

which consists of packets sent by the client periodically. If the client or the gateway fails to respond to the keep-alive protocol, the peer is marked as down and a registration protocol is reinitialized.

The commands protocol is employed by the gateway in the process of controlling the telemetry report cadence and scenarios. The client agent has a modular telemetry implementation with a core module that handles all the base operations and a specialized module for each type of telemetry information. When the client agent starts, it synchronizes the information with the gateway and all the subsequent updates are based on the configuration received from the gateway. The gateway configuration consists of two types of notifications: push and threshold notifications. Push notifications have the purpose of updating periodically certain telemetry information and threshold notifications have the purpose of synchronizing data only if the value passes a given threshold. Threshold notifications are implemented using a fast internal scan rate and the data is updated only if necessary, thus the gateway can monitor changes only for a subset of parameters while limiting the network traffic. In Figure 2, is presented the client agent telemetry and commands protocol architecture. As can be observed, the client agent uses as a telemetry data source, both the data plane application and the IoT host device system (Linux).

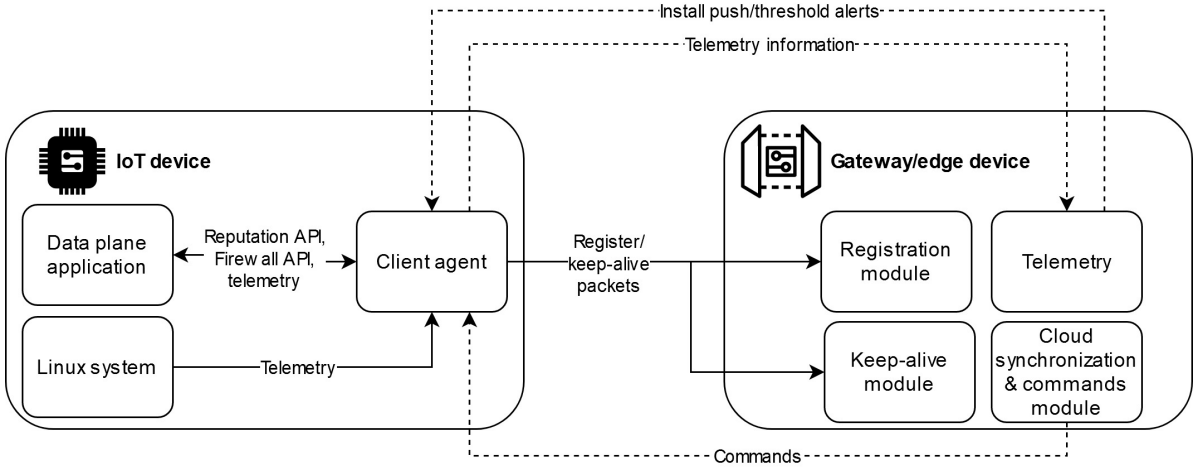

Figure 2. The communication between client and gateway.

The RESFIT client delivers to the data plane application a MQTT client ID which is identical with the client agent's unique identifier. This is required because the firewall rules on the gateway side are shaping the ingress traffic for a given node which is identified by the MQTT client ID. Furthermore, a fraction of the node reputation score is computed using the number of transmitted packets, the MQTT client ID being the element that uniquely identifies the transmitter node.

A security issue in this context is mitigating the possibility of a rogue node to impersonate the MQTT client ID of another node, as this would affect the reputation of the impersonated node. To eliminate this risk, the client agent delivers to the data plane application the device identity (used as MQTT client ID and MQTT username) and a unique temporary password (used as MQTT password). Thus, the data plane application authenticates to the broker using the aforementioned credentials, while the client agent delivers to the gateway the unique password, which in turn is transmitted to the broker authentication plug-in. Still, this mechanism solves only the data plane to broker authentication, but another security concern remains the data plane to data plane communication (sensor data transmission which impacts the reputation score). The MQTT protocol does not have a mechanism to encapsulate the data source identity in the packet, thus the broker security plug-in intercepts each packet and appends a tag with the publisher's identity in the packet payload. On the subscriber side, the outermost identity tag is checked and the publisher's identity is obtained.

The client agent module of RESFIT is implemented in the C programming language, being a single thread application which uses the run-to-completion paradigm. The client agent has an internal userspace scheduler which executes a task on several inputs: when a CoAP packet is received from the gateway, when a control packet is received from the data plane or when an internal timer expires. For the CoAP communication, the client agent employs the libcoap library. The client agent must run on a Linux operating system as it uses several Linux primitives (timerfd structure, Unix socket, etc.). The telemetry system of the client agent has a modular structure which comprises several collectors: static information collectors (e.g., kernel version, hostname), sysinfo information (e.g., number of processes, CPU load) and data plane collectors (e.g., number of received packets per minute, average packet length). Given this structure, the telemetry module can be easily extended with other types of collectors. The internal communication between the data plane application and the client agent is supported by an Unix socket initialized as a SOCK_SEQPACKET, enabling us to use the features present in the Stream Control Transmission Protocol (SCTP), especially the integrated mechanism of delivering messages in accordance with their sending order. The internal connection employs a lightweight Type-Length-Value (TLV) encoding of the messages along with a reconnect capability if one of the components is restarted.

3.3. Gateway Module

The gateway component of the RESFIT platform is a core element of the system, being responsible for aggregating information from all the local IoT devices, computing the reputation score for each device and updating all the information to the cloud layer. Each client agent has to register its credentials (identifier and PSK) to the gateway in order for the registration protocol to be executed successfully. This is achieved on the gateway side, using a gateway command line (CLI) tool. The CLI tool allows the administrator to add a new IoT device to the internal database of the gateway. Until the credentials are added to the gateway side, every attempt of the client agent (IoT device) to complete the registration protocol will halt, because the DTLS-PSK handshake fails. This new client addition procedure represents the out-of-band mechanism mentioned in Section 3.2 and it was designed in this manner to ensure that only administrator validated IoT devices are installed in the system.

Being the middle layer of RESFIT, the gateway implements two communication protocols: CoAPs for the control plane downstream connection with the IoT device and MQTTs for the cloud layer upstream connection. Regarding the downstream connection, as mentioned in the previous section, the gateway acts as a CoAP server when receiving control data from the clients and as a CoAP client when configuring notifications setup on the client. Being a multi-tenant system, the gateway has to multiplex the PSK used to secure the CoAP connections. To achieve this, a mechanism equivalent to the Server Name Indication (SNI) extension of Transport Layer Security (TLS), has been implemented for Datagram Transport Layer Security (DTLS) connections, where the gateway switches a context PSK based on the incoming packet identity. The client PSK along with the identities are stored in a local database, registered using an out-of-band mechanism (the administrator action presented at the beginning of this section) and fetched by the gateway software when a PSK is missing from the local cache. Although simpler, the scenario for the CoAP client request is similar, with the gateway switching the context DTLS-PSK based on each IoT device destination. As presented in the client agent section, the implemented DTLS based protocol is fault tolerant, handling situations where both the client agent and the gateway might become offline. These scenarios are handled using retransmission on the application layer, triggering the registration protocol if multiple packets cannot be transmitted (threshold-based).

The mechanism for registering the gateway module to the cloud application is similar to the client to gateway registration mechanism: when first started, the gateway randomly generates an identifier and PSK, and this information is registered on the cloud side using

an out-of-band mechanism. These credentials are the security anchor used by the gateway to communicate with the cloud.

As mentioned before, the MQTT protocol is used to implement the communication channel between the gateway and the cloud application. This protocol was chosen for various reasons:

- It is a reliable and lightweight protocol;
- It is fast and offers queuing by design (transport level);
- It permits asynchronous cloud to gateway communication, even if the gateway is deployed in a network with network address translation (NAT);
- It easily permits future enhancements like gateway to gateway communication (when both of the gateways are deployed in a NAT network);
- It enables us to lock the creation of the communication channel between the gateway and cloud applications to IP-based networks.

Regarding the upstream MQTT connection, the gateway acts as a MQTT client, which maintains a continuous connection with the cloud application. This communication is secured using a TLS connection, where the cloud application broker is authenticated by validating the server certificate against a local certificate trust store. The gateway is authenticated to the cloud broker using the unique identifier and PSK as MQTT username and password. Being part of a publish–subscribe architecture, the communication between the gateway and the cloud application must use a security layer which, besides confidentiality, provides authentication of parties: the gateway must receive commands only from the cloud and the cloud must increment the statistics for the correct gateway, mitigating spoofing attacks. This security layer was implemented using the gateway PSK value and the MQTT topic. The gateway publishes commands to a MQTT topic equal to the PSK value, which is concatenated with the communication direction ("gateway-to-cloud"). In a similar manner, the cloud publishes information to the same PSK topic, with a different suffix ("cloud-to-gateway"). The direction suffix is used to obtain a full-duplex communication between the gateway and the cloud, while achieving authenticity, given that the PSK is known only by the gateway, the cloud application and the cloud broker (which is part of the cloud application).

Besides the CLI tool, which is a separate binary, the gateway system has another important sub-module which handles the inspection of client data plane MQTT packets and acts as an application layer firewall. The firewall system implements the interface of a Mosquitto plug-in (the MQTT broker implementation used in the IoT platform). Because a Mosquitto plug-in only has MQTT authentication related functions, we modified the Mosquitto solution and added a packet inspection capability. Thus, before routing a packet from a publisher to a subscriber, the Mosquitto broker redirects the packet (along with metadata like source and destination) to the firewall plug-in. Then, based on a return code from the firewall plug-in, the packet is routed or dropped. Internally, the firewall plug-in classifies the packet based on a firewall ruleset received from the gateway. A firewall rule has the following criteria:

- Destination MQTT client ID;
- Number of packets per minute;
- Length of the packet;
- The Quality of service (QoS) value of the packet.

A firewall rule is installed by the data plane IoT application, which transmits it to the client agent module (using the SCTP-based Unix socket) and which in turn transmits it to the gateway (using the CoAPs connection). Afterward, the gateway transmits the firewall rule to the firewall plug-in. Firewall statistics that are returned periodically from the firewall plug-in to the gateway include metrics like number of packets allowed/rejected by a certain rule on the ingress side, along with the number of packets transmitted/rejected on the egress side. These metrics are used for telemetry purposes, being uploaded to the cloud application, and for computing the reputation score.

The gateway module is implemented in the C++ programming language using the Boost. Asio framework [22]. The gateway software is also a single thread application that uses the run-to-completion paradigm and relies on the Boost.Asio scheduler. The gateway scheduler executes a task when an internal event is triggered (e.g., timer) or when it receives a packet from the following sources: CoAP packet from the client agent, MQTT packet from the cloud application, statistics packets from the firewall module. Similarly to the client agent, the gateway also uses libcoap to implement the CoAPs communication and the C++ Eclipse Paho MQTT library for the MQTT communication. For the internal communication between the gateway core and the firewall module, a stream Unix socket is used with a binary TLV encoding protocol. The communication protocol between the gateway core and the firewall module is fault tolerant, with the firewall rules being synchronized from the gateway every time the gateway restarts (if this connection is interrupted every module continuously tries to re-establish the link with the other module). The firewall module of the gateway software is implemented as a plug-in of the MQTT Mosquitto solution, this being a Linux shared object library that is loaded at run-time by Mosquitto (indicated in the Mosquitto configuration file).

3.4. Cloud Module

The cloud application is a system aggregation module that synchronizes data from multiple gateways and IoT clients. The application consists of a back-end module that communicates with the gateways via the MQTT protocol and a web front-end, which is the user interface. The back-end module acts as a MQTT client that subscribes and publishes data to a different MQTT topic given by the gateway PSK. The back-end module also provides persistence for the gateway published data using a NoSQL database for storing the received information. The NoSQL solution was chosen because of its schemaless property, being adequate for a dynamic IoT environment where the telemetry data extracted from the client are subject to change and does not have a fixed format. The synchronization mechanism between the gateway and the cloud back-end module consists of a lightweight protocol, where only the updates (new events) of a client are transmitted to the cloud in a JSON format. Thus, the gateway maintains an internal representation of a client IoT device and sends an update to the cloud only when a field of that internal representation changes. This protocol is fault tolerant and takes into consideration scenarios where the cloud or the gateway might become offline, thus a full synchronization is executed every time an entity starts a new connection. Furthermore, the cloud module has the capability of sending commands to the gateways, like full synchronization commands for which the gateway has to send all the IoT device information. These commands are asynchronous, a capability that comes along with using the MQTT protocol. The communication between the gateways and the cloud application is implemented using a cloud MQTT broker, which is a component of the cloud layer. While the communication between the gateways and the cloud broker is encrypted and authenticated (using the identity and PSK), the communication between the cloud application and the cloud broker is not encrypted, being an internal one (same trust domain and different MQTT port, which listens only on the loopback interface). The cloud application is also the control plane module which allows the gateways to login to the MQTT cloud broker using the identity and PSK information as credentials. Thus, after an administrator registers a gateway in a web interface, the cloud back-end refreshes a credentials file and notifies the MQTT broker about the change. A similar action is executed when a gateway is removed from the web interface and the credentials file is modified accordingly, the active connection is closed.

Besides storing a real-time shadow copy of the IoT clients, the cloud application also holds historical data for the firewall and reputation score samples. The historical values are displayed in a chart, which helps the administrator to inspect the evolution of an IoT client. The web application is synchronized with the back-end in real-time, having implemented an asynchronous pooling mechanism that helps in displaying in the user

interface the telemetry information with minimum delay. The high-level structure of the cloud application is depicted in Figure 3.

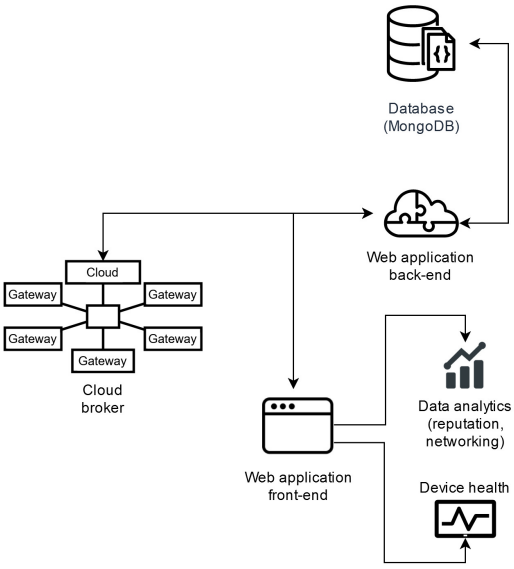

Figure 3. High-level architecture of the cloud application.

The cloud application backend module is developed using Java Spring Boot framework. The cloud back-end module employs a MQTT client for communicating with the gateways, for this purpose the MQTT Eclipse Paho Java library is being used. The front-end module of the cloud application is implemented using the AngularJS framework, the web interface being synchronized in real-time with the back-end using an AJAX pooling mechanism. The access to the front-end module is restricted to the administrator users only, the authentication and authorization being based on a digital certificate (TLS connection with mutual authentication).

The cloud layer also includes a Mosquitto MQTT broker for handling the communication with the gateways. The communication between the cloud application and the cloud broker uses an internal connection.

3.5. Smartphone Command Authorization Module

The RESFIT platform also implements a smartphone application that allows an administrator to authorize sensitive commands received by the IoT device and to receive real-time notifications regarding telemetry anomalies. The smartphone application addresses the scenario where RESFIT is considered a partially trusted entity and where the IoT network owner wants to control the sensitive commands received from the cloud application. In this context, the cloud application management and monitoring are outsourced to another organization, but the IoT network owner wants to enforce a security control layer. For instance, if the cloud application administrator wants to execute a disruptive action (e.g., a reboot command), the owner might want to be notified about this event and approve it. The disruptive action approval mechanism relies on an end-to-end security relationship between the owner's smartphone and the IoT gateway. Thus, each approval action is cryptographically signed by the owner and verified by the IoT gateway. RESFIT acts only as a transport layer in the command approval scenario, being responsible for notifying the owner about the event and then exchanging the messages between the owner and the gateway.

The trust anchor for the command approval feature is a key exchange that takes place between the owner's smartphone and the gateway, the communication channel for this key exchange process being the internal IoT network, without involving the other components of the IoT reputation platform. The key exchange is a gateway claiming process, where the

smartphone claims the gateway by pushing a pre-shared key into the unclaimed device. After the gateway is claimed by an owner, the device will not accept any secret keys. This secret key is used to authenticate every command approval request and response. The claiming process consists of the following steps:

1. The gateway module of the IoT reputation platform generates a short code at a fixed interval and displays the code to the console. Thus, the owner, which has physical access to the gateway, will be able to visualize the short code.
2. The short code along with the IP address of the gateway is entered by the owner into the smartphone application.
3. When claiming a device, the smartphone application randomly generates a claiming key and sends it to the gateway, along with the owner's identity, using the HTTPS protocol.
4. When receiving the message from step (3), the gateway embedded HTTP server verifies the short code from step (1). If the code delivered by the smartphone application is correct, the gateway saves in a local file the claiming key along with the owner's identity. After saving the owner's identity, the gateway transitions into a claimed state and discards subsequent claim requests.
5. On the smartphone application, the administrator can inspect the IoT devices that are registered to the gateway and choose the devices that require a command approval process.

When the RESFIT administrator sends a command to an IoT device, the message is first transmitted to the associated gateway, which in turn relays it to the IoT device. The message is first transported using the MQTT protocol from the cloud to the gateway and then it is transported in a CoAPs request from the gateway to the IoT device. If the client agent receives a sensitive command, an approval protocol is executed with the owner's smartphone. The approval process is triggered on the gateway side when an IoT device which requires approval receives a sensitive command from the cloud application. Thus, before transmitting the command to the client agent, the gateway will execute the approval protocol.

The owner identifier and a timestamp are appended to the command and the new structure is sealed using an HMAC which employs the claim key shared between the gateway and the owner's smartphone. This approval request is transmitted back to the cloud application which relays the request to the owner using a Firebase push notification mechanism. The smartphone compares the timestamp of the last approval request against the current approval request in order to mitigate replay attacks and after verifying the HMAC seal, the owner can approve the command. The approval response is also sealed with the HMAC, which employs the claim key and transmitted back to the cloud application in order to be relayed to the gateway. In the final step, if the gateway validates the origin of the approval response, it relays the command to the IoT device. In Figure 4, is presented the structure of the smartphone application along with the claiming and approval processes.

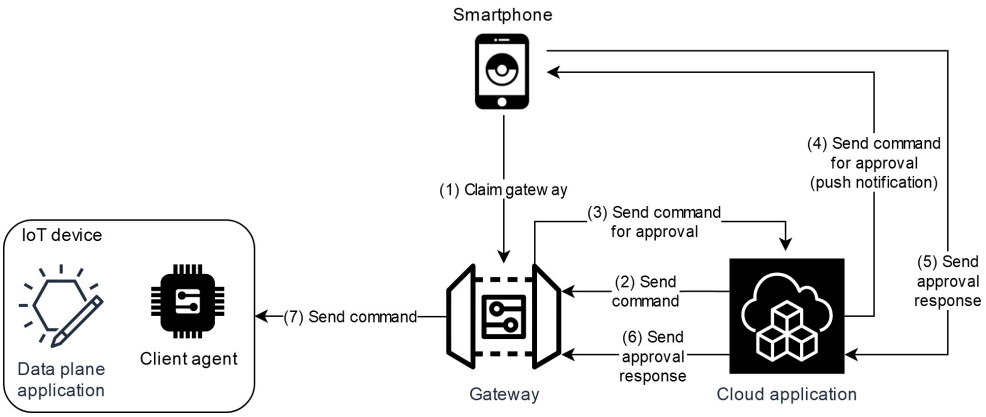

Figure 4. Smartphone claim and approval command protocol.

The smartphone application is implemented in the Java programming language using the Android SDK. The application communicates with the gateway module (claiming process) and with the cloud module (approval process) using HTTP protocol (representational state transfer (REST) API). The approval command requests are transmitted from the cloud module to the smartphone application using a push notification, implemented using the Firebase system. Regarding the claiming process, the gateway module runs an embedded HTTP/2 server implemented using the nghttp2 C++ library, which allows processing the claim request.

3.6. Reputation Mechanism

The core functionality of RESFIT is the reputation mechanism, which allows obtaining the most trusted device from the IoT network within a given sensor category along with observing malfunctioning devices that need to be removed from the network. The reputation mechanism is implemented using a Naive Bayes Network algorithm, where each independent feature is either a sensor or system feature, as it will be presented in the next paragraphs. Implementing a Naive Bayes to evaluate the reputation of an IoT device has several advantages, among which: a low computational/consumption and fast algorithm along with a simple solution design. These factors have to be taken into consideration for the IoT gateway, which is a resource constrained device and possibly battery powered. Furthermore, the gateway may host another set of IoT applications, thus not being fully dedicated in running a reputation or security algorithm. In [5], we presented an implementation of the Naive Bayes algorithm on an IoT platform. The core of that Naive Bayes implementation was re-used in RESFIT, with several structural modifications.

One of the main drawbacks from [5] was the assumption of a homogeneous IoT network from the sensor capabilities standpoint. This assumption consists of having IoT devices equipped with the same set of sensors and compute a reputation score per IoT device and not per sensor. This requires that all the IoT devices have the same capabilities in order to compare reputation scores computed on the same metrics. For the implementation of the RESFIT platform, we wanted to overcome this limitation and allow heterogeneous sensors in the network, while measuring the reputation score for each sensor feature. Thus, a device equipped with several sensors will have an individual reputation score for each one of those sensors. For instance, a device may have a high reputation score for a sensor and a low reputation score for the other sensors. For this to be achieved, the Naive Bayes network for a sensor consists of the following independent features: the correctness of the sensor output value and the response time from an IoT device when the respective sensor is queried. As we presented in [4], the data reputation score of a device (p) is based on the composition of these individual sensor reputation scores, each one updated only when the weighted satisfaction score (s) passes a defined threshold. Equation (1) presents the computation done for establishing the data reputation score of a device, while Equation (2) shows how the weighted satisfaction score is being calculated.

$$p(T = 1, F_1, F_2, ..., F_n) = \frac{\#\ of\ successful\ transactions}{\#\ of\ total\ transactions} \times PROD(\frac{\#\ of\ successful\ transactions\ with\ F_i}{\#\ of\ successful\ transactions}) \quad (1)$$

$$s = W_{F_1} \times S_{F_1} + W_{F_2} \times S_{F_2} + \cdots + W_{F_n} \times S_{F_n},\ with\ W_{F_1} + W_{F_2} + \cdots + W_{F_n} = 1 \quad (2)$$

where W_{F_i} represents the weight of a certain feature and S_{F_i} the satisfaction score of that feature.

This reputation sensor score is used in conjunction with a system reputation score which measures the overall behavior of the IoT node in the network. The system reputation score has the following independent features:

- The time when the device is registered;
- The number of keep-alive packets transmitted in a time window;
- The number of data plane MQTT packets successfully transmitted in the network (packets which are accepted by other devices).

The system reputation score measures the device uptime, its connection health and the quality of data injected into the network, thus giving the probability of a device to respond with quality data in a timely manner when queried by other devices. Having a Naive Bayes network as the basis, the computations done for obtaining the system reputation score of a device are similar to those presented in Equations (1) and (2). Yet, because in this case we do not receive feedback from different external entities, a similar metric must be provided so that the Naive Bayes network can function properly. Thus, the gateway computes internal pseudo-feedbacks for each device, based on their behavior in the system, using the three metrics mentioned above. Each one of them is computed as shown in Equations (3)–(5).

$$S_{registration\_time} = \frac{\#\,of\ registered\ seconds}{\#\,of\ registered\ seconds\ expected\ for\ the\ sampling\ period} \tag{3}$$

$$S_{data\_packets} = \frac{\#\,of\ txPassedPackets}{\#\,of\ txPassedPackets\ +\ \#\,of\ txDroppedPackets} \tag{4}$$

$$S_{keepalive\_packets} = \frac{\#\,of\ keepalive\ packets}{\#\,of\ keepalive\ packets\ expected\ for\ the\ samplingperiod} \tag{5}$$

After both the data and system reputation score are computed for each device, they can be then used to obtain the total reputation score of a device, computed as shown in Equation (6), that will then offer a specific trust level for that device.

$$R_{device} = W_{data} \times p_{data} + W_{system} \times p_{system} \tag{6}$$

where W_{data} and W_{system} representing the data and system reputation score weights, and p_{data} and p_{system} the data and system reputation scores previously computed.

By implementing this separation between sensor and system reputation, for each device, the RESFIT platform becomes context-aware and enables administrators to better decide and plan any automated actions. As presented in Figure 5, there are two trust weights assigned to each reputation score (system or data), when used for computing the overall reputation score of a device. These trust weights values, included in equation, can be customized by the administrator, thus obtaining one of the following three possible working scenarios:

- Data-aware security (high weight for sensor trust and low weight for system trust);
- Context-aware security (low weight for sensor trust and high weight for system trust);
- A mix between the previous two (trust weights values are identical).

The testing scenarios presented in this paper are based on default trust weights values of 0.7 for system reputation and 0.3 for data reputation.

As it can be observed in Figure 5, the reputation system consists of two Naive Bayes networks (sensor and system), whose output is used in a weighted sum in order to compute the overall trust score. Thus, the system is flexible in terms of Naive Bayes score importance: one Naive Bayes score can have a higher weight than the other (e.g., for computing the overall trust score an application may attach a greater importance to the sensor trust value).

The system reputation score is computed on the gateway side, without requiring any feedback from the clients, the gateway being the module that handles registration/keep-alive and has access to the firewall statistics. Regarding the sensor reputation score, the gateway collects feedback from the clients, the method for computing the feedback being application specific and out of the scope of the RESFIT platform. In the data plane sample application that runs on the IoT device, we propose two simple methods for providing the sensor feedback. The first one consists of computing the reputation score based on a deviation of the current sample from a fixed value. This method is, however, suitable

only for the testing phase of the system. The second method consists of computing the reputation score based on a deviation of the current sample from an average value.

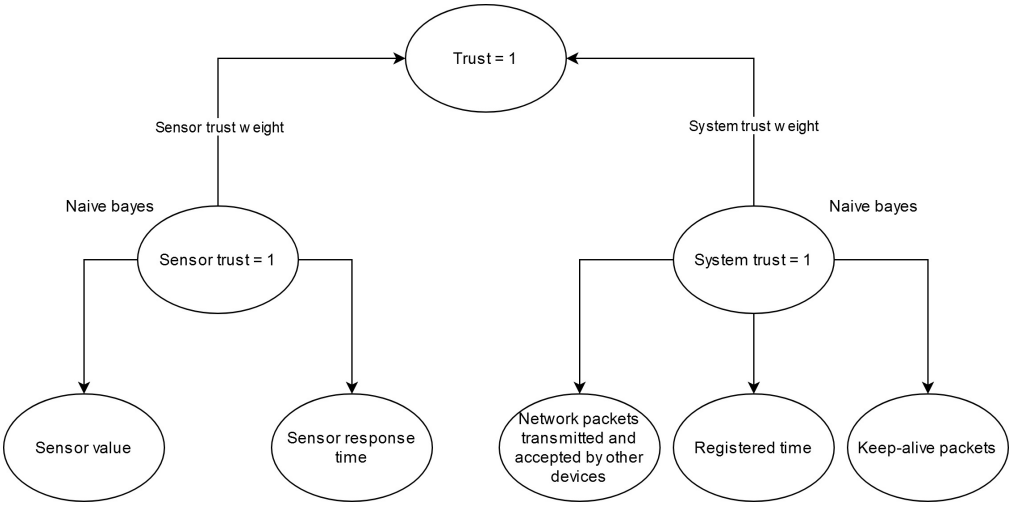

Figure 5. Naive Bayes reputation schema.

The average value is established by averaging all the sensor values received within a time frame. Thus, the closer the current value is to the average value, the higher the reputation score will be. In Figure 6, are presented the steps in computing the reputation score for an IoT device sensor:

1. The data plane application queries the gateway using the client API (via the client agent application) for the most trusted device within a sensing category (e.g., temperature sensors).
2. The gateway delivers the identity of the most trusted device.
3. The data plane application sends a broadcast request in the system in order to query sensor values for a category (e.g., temperature). This message is transmitted using the MQTT protocol (data plane) by sending a message to a predefined request topic like "data/<sensor type>", each data plane application being subscribed to a request topic for each available sensor. After sending the request message, the data plane application starts a timer with a fixed value (time window).
4. The devices equipped with the required sensor responds with a sensor sample information.
5. The requester data plane application collects all the messages received within the time window (timer configured at step 3) and assigns a response time score to each sample. Thus, a sample will have a high response time score if the message is received in the beginning of the time window, otherwise the score will decrease. After the time window ends, the requester data plane application generates the average for all the sensor values and then computes the reputation score for each sensor value, using the difference of the current sample from the average value. The data plane application consumes only the value from the most trusted device (the identity received from the gateway at step 2).
6. The requester data plane application updates the response time score and the sensor feedback score for all the devices that contributed with a sample. Executing this step allows other devices to increase their reputation score and possibly decrease the reputation score of the most trusted devices, if it provides inaccurate data.
7. After receiving the feedback score, the gateway module executes another iteration of the reputation algorithm and it updates the reputation repository accordingly.

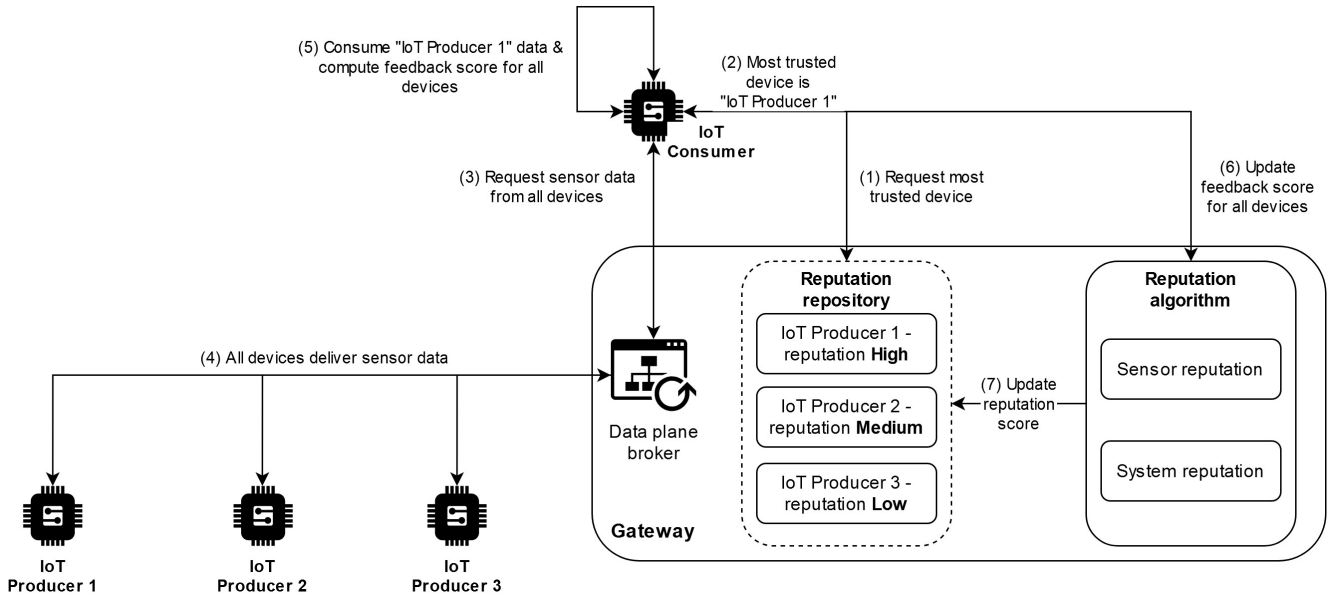

Figure 6. Reputation query and update system.

4. Testing and Analysis

For testing the RESFIT platform, the following setup was used: the client agent and the data plane application were deployed on several Raspberry Pi board (*Raspberry Pi 3 Model B Plus Rev 1.3 with SoC BCM2835 and 1 GB RAM*), the gateway was deployed also on multiple Odroid boards (*Odroid-XU4Q with Samsung Exynos5422 Cortex-A15 2Ghz și Cortex-A7 Octa core and 2GB LPDDR3 RAM*). The cloud module was deployed on a virtual machine running Ubuntu operating system with the web application running on an Apache Tomcat server. Focusing on the customizable components of our proposed security platform, we chose to create our own simulated sensors, instead of using either an existing dataset or a traffic generator. In this manner, we were able, through a single application, not only to create traffic shaped for different sensing categories and data values, but also to have control how feedback for consumed data was being generated. Existing datasets, some of them also artificially generated, would have provided data values tailored for specific attacks, but no information regarding the feedback mechanism (scoring methods, threshold values etc.). The same limitations apply to existing traffic generators: existing permission to alter the traffic generator source code in order to connect with the Client module or offering a functionality through which feedback can be given.

For testing purposes, the data plane application supports the generation of simulated data with target and deviation values (the higher the deviation value, the lower the data quality). The actual sensing value is not important for RESFIT, because each value receives a score (between 0 and 1) which is further processed by the system. In order to simulate external consumers that can offer feedback for the data they receive, we implemented, in the data plane application, two methods for generating computing the a feedback (data quality score):

- Distance from the target value—given that we have a simulated dataset, we can define the target value that is specific to data with highest quality;
- Distance from the average value—computed within a time window, by first establishing the average of values transferred in the specific time interval and then assessing each value produces by a device against the previously obtained average value.

This behavior of the data plane enables the creation of various test scenarios, thus allowing a complex analysis of how the reputation-based security platform reacts to different stimuli. A first test takes into consideration a normal system, in which nodes deliver data with different levels of accuracy (accuracy can be manipulated using the deviation

parameter used for configuring each data plane instance). A minimal test system has been constructed by deploying four clients that, for emphasizing the security characteristics, are configured as follows:

- Three of them act as producers, publishing data to a specific (e.g., temperature) feature at regular intervals (in this case 1 s) and with specific deviations from the same standard value. In this case, the target value is 30 and the deviations are: 1 for the first client, 2 for the second client and 3 for the third client—this will result in an incremental ranking of their data reputation;
- The last one acts as a consumer for that specific feature, for which producers send data, and gives feedback for clients that publish values in a defined time period. In this case, every 60 s there is a 10 s time frame in which other clients can send feature-specific values. The feedback is computed in relation to the target value (set to 30, as mentioned above).

This test includes also a simulated device malfunction (e.g., power outage in a certain sector, device overheated and shut down or encountered a software error and is no longer responding) occurring for Device 3 at half way during the test. As it can be seen in Figure 7, Device 1 is the "consumer" node, as it does not have any reputation value calculated since it does not receive any feedback from other nodes, but instead offers feedback to the other 3 nodes, while Devices 2 to 4 are the "producer" nodes and each has an associated reputation value, depending on the accuracy of its published data (generated based on the deviations mentioned above). As it can be seen from Figure 7 the system stabilizes quickly and the ranking of nodes is consistent with how data are being generated. Furthermore, in Figure 7, when Device 3 becomes unresponsive, it can be seen that its data reputation value remains constant, while its system reputation value drops and continues this trend until the end of the simulation (Figure 8). Thus, by creating this separation between its data accuracy and hardware stability, RESFIT does not hinder nodes that were offline for a period of time and, when coming back online, continue to produce trustworthy data. The system presents a high sensibility to any variation of reputation, related to either data feedback or system behavior, and each shift of a node from the normalized behavior is noticed in the reputation score. Still, even though the system reputation is computed internally by the gateway, as mentioned in Section 3.6, there is a limitation regarding the data reputation that needs valuable external feedback from the consumers. Being only for testing purposes, the feedback algorithm included in the data plane application of the RESFIT platform is not adapted for real IoT systems, thus an administrator will have to deploy their own feedback mechanism.

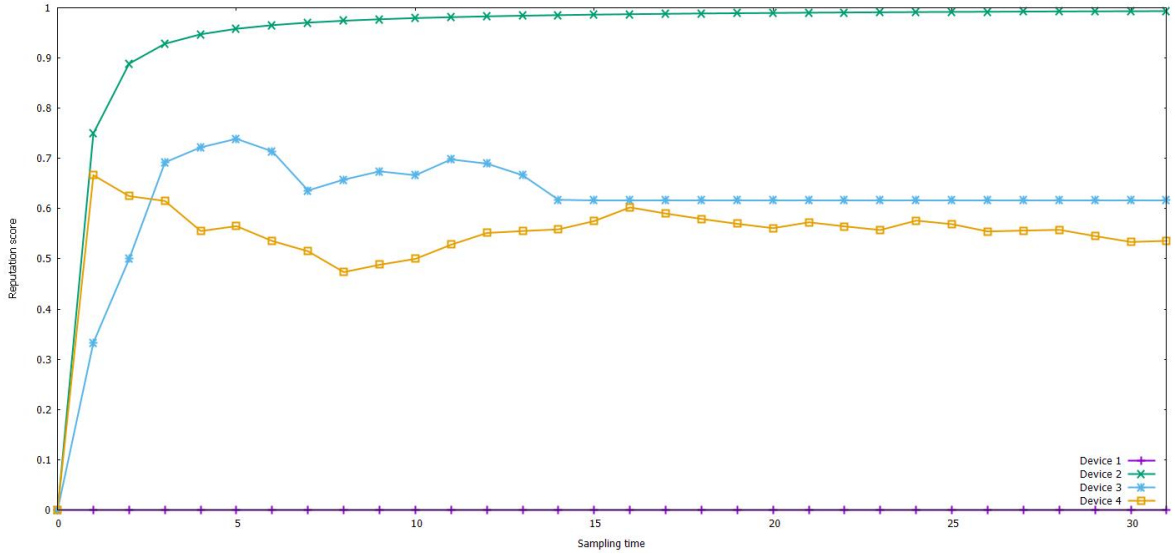

Figure 7. Sensor value reputation of each device, obtained in test 1.

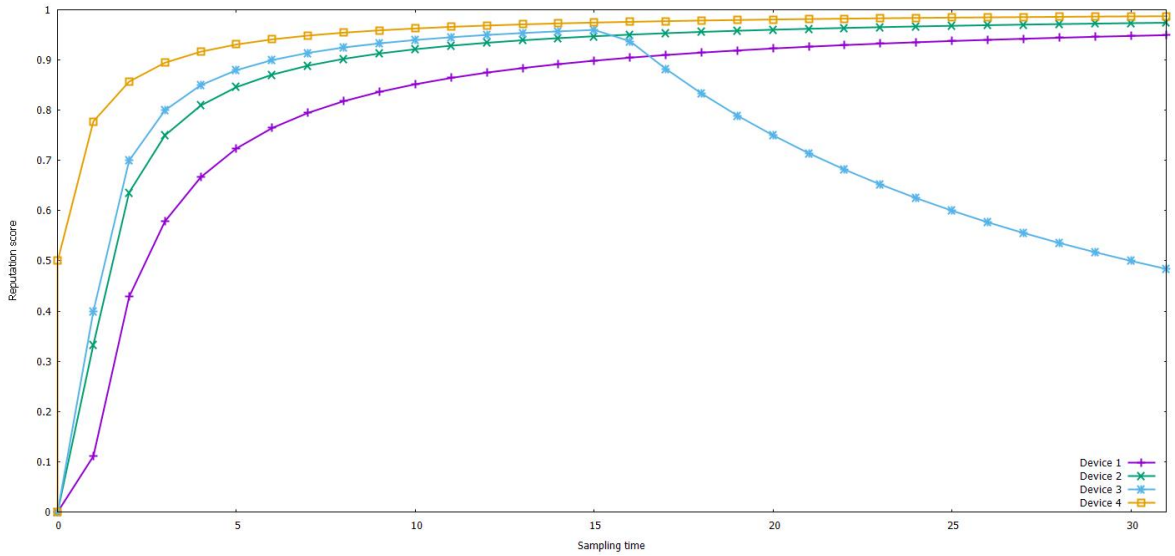

Figure 8. System reputation of each device, obtained in test 1.

The second test scenario starts from the normal functioning of the IoT system, also with three nodes configured as producers of data for the same specific feature and one node configured as a consumer of that data (it will also give feedback for each producer). After a period, a simulated Denial-of-Service (DoS) attack is successfully executed on the entire IoT system, thus all communications are halted. This can be seen in Figure 9, as the system reputation for all the devices drops for a period of time, until the DoS is successfully mitigated and communications are restored.

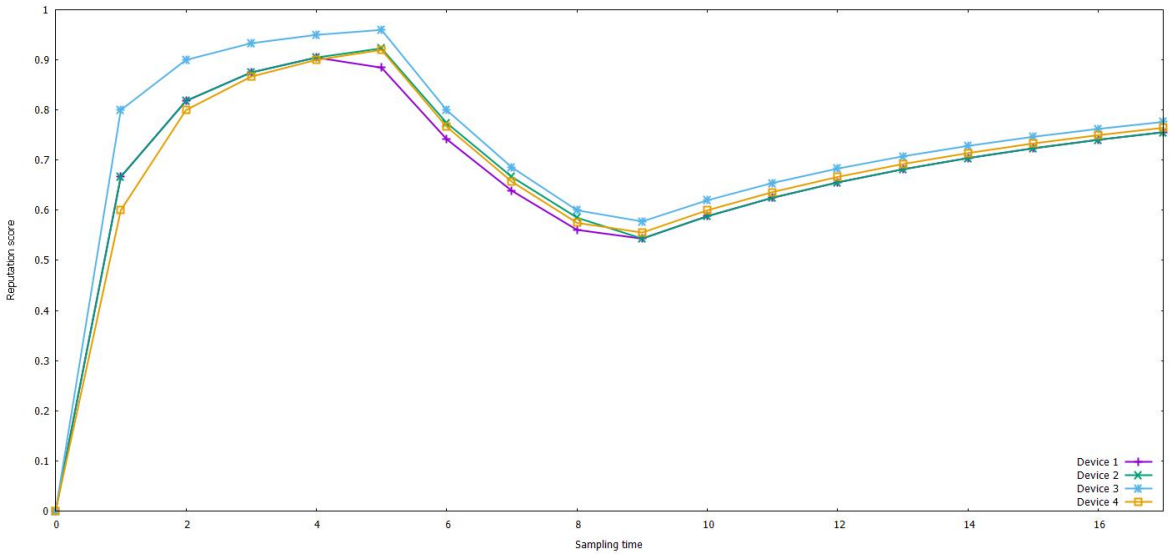

Figure 9. System reputation of each device, obtained in test 2.

In this scenario, it is considered that the DoS attack has been just a forefront from the real intention of the attacker, namely to infiltrate the network through a vulnerable node. This is simulated as a success for the attacker and Device 2, the one with the most accurate data, becomes rogue and now produces low-quality data. The other two producer-nodes maintain their data quality.

The RESFIT platform detects the malicious data that is now being inserted in the system and has a rapid response, by decreasing data reputation of Device 2. This can be seen in Figure 10, in the second part of the graph, where the data reputation of Device 2 drops continuously, while the other two nodes have their data reputation stable or even

slightly increasing (this is situation for Device 4, since its data is now close to the values provided by Device 3, that produced also medium-quality data and Device 2 is being slowly eliminated from the system-passing under a certain threshold for data reputation means that the node is no longer trustworthy). The node elimination threshold can be programmatically configured and updated by the administrator depending on the trend in data quality observed for that specific system. In case of multiple nodes being hijacked by attackers and used to influence data quality in a system, the output RESFIT will produce depends mainly on how feedback is generated for this malicious data that floods the systems. The number of rogue IoT nodes is also important, but does not affect as much the viability of the detection method as how the feedback does. As it can be seen from both Figures 7 and 10, reputation is easy to lose and hard to recover.

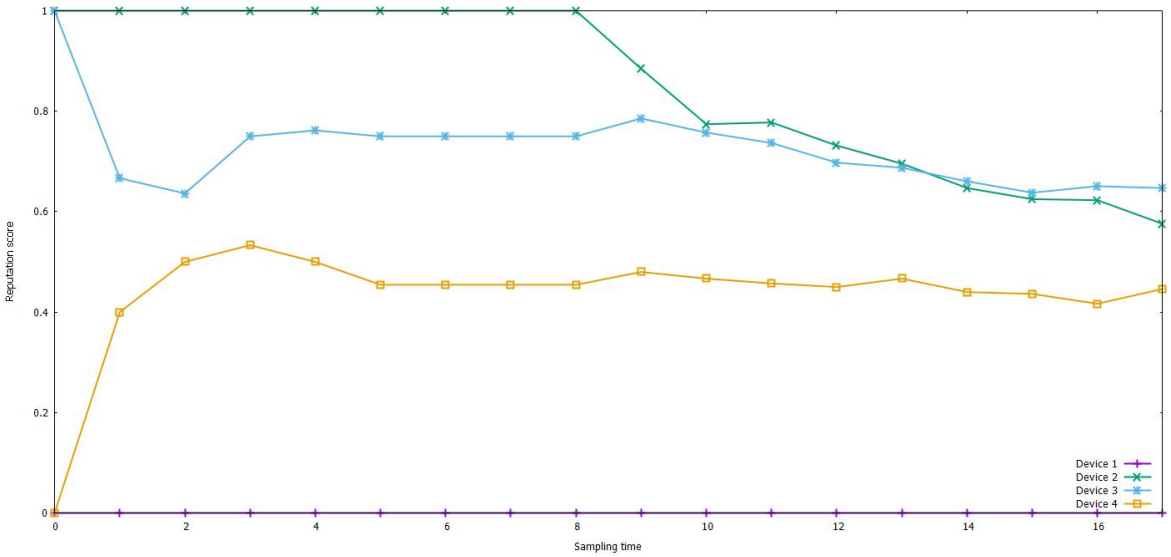

Figure 10. Sensor value reputation of each device, obtained in test 2.

The IoT devices communicate with the gateway using the CoAP protocol, transmitting register, keep-alive and telemetry packets. A gateway can manage a large number of IoT devices, thus from the performance perspective important metrics are how many CoAP requests per second can the gateway handle and what is the average response time under high load. To evaluate these metrics, a CoAP stress test was executed to observe the performance capabilities of the gateway module. In this test scenario, the gateway software runs on an Odroid board and two client agents are started on a virtual machine, each client agent transmitting several keep-alive packets at a high rate. Every client agent measures the delay of the CoAP keep-alive response and computes the average delay for all the transmitted packets. By having client agents which are transmitting several requests within a second time interval, the gateway is forced to switch the CoAP DTLS-PSK to serve requests originating from different CoAP identities. The results of the experiment are presented in Figure 11. As it can be observed, the average response time for a high number of requests per second respects the near real-time constraints of the client agent (without causing retransmissions or dropping packets). To conduct the experiment, the *libcoap* library configuration was modified to support a large number of CoAP concurrent sessions (default number is 100).

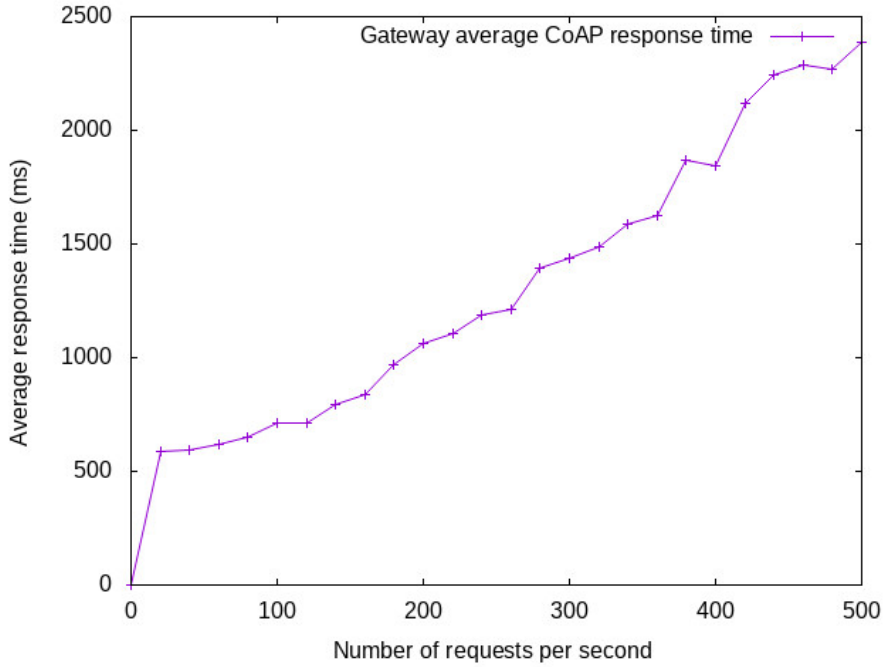

Figure 11. Gateway average response time for concurrent CoAP requests.

5. Conclusions and Future Work

In this paper, we propose RESFIT, a reputation-based security monitoring platform that makes use of Naive Bayes networks to evaluate the reputation of each device connected in the network. A gateway-centric architecture was chosen when designing the security platform and this could be seen across the paper, as the gateway has the primary logic in both computing the reputation and also enforcing a packet filtering policy. From the management point of view, an administrator has an integrated overview of the IoT system through the cloud component, while the mobile application enables a user to rapidly be aware of any suspicious state changes that a device could be requested to execute. Based on the previous results, the platform shows a good response time in detecting malicious actions and, by further implementing a blacklisting logic inside the IoT system if a node goes beneath a certain reputation threshold, compromised devices will be automatically eliminated to ensure the stability of the IoT system.

Ownership reinforcement of an IoT network through additional security control layer and a gateway-centric reputation solution that can counter incipient faulty misbehavior of nodes are two main contributions of this article.

The reputation system provides a flexible mechanism for evaluating the IoT nodes by means of the two Naive Bayes networks that are evaluating both the system and the sensor reputation. Moreover, each Naive Bayes network allows adding multiple features with configurable threshold values, making the solution suitable for a tailored IoT reputation scenario.

Even though the Naive Bayes method for computing the reputation score is a feasible solution, from the performance and low resource requirement perspective, by providing a fast mechanism to train the network about historical data and obtain a binary classification outcome (if a device is trusted or not), a future research direction in this area is implementing other machine learning classification methods like k-nearest neighbors (k-NN), support vector machines (SVM) and decision tree classifiers and comparing the results with the ones obtained using the Naive Bayes solution in terms of prediction accuracy and performance requirements. Another aspect of the proposed solution that can be improved is the way the registering of IoT devices to the gateway is executed. RESFIT requires an out-of-band channel for transferring the IoT device identity and PSK to the gateway, which in our implementation assumes a manual intervention of the owner. This operation can be

cumbersome for an IoT network with a large number of devices, thus, implementing an automatic IoT device registration protocol is a future direction to focus on.

Another research direction is extending the capabilities of RESFIT in terms of automation and integration with third-party applications. This can be achieved by exposing all the functionalities which are accessible to an administrator through REST APIs. Thus, an administrator could use scripts to interact with the platform in an automatic manner and data aggregated by the cloud application could be exported to a third-party application for further analysis. A REST API can be developed on the gateway layer, given that this module already runs an embedded HTTP/2 server for interacting with the smartphone application. By implementing a REST API on the gateway, an administrator can query and configure information without employing the cloud module. Furthermore, a REST API would allow RESFIT to become a building block for other IoT applications.

Author Contributions: Conceptualization, Ș.-C.A., B.-C.C. and I.B.; methodology, Ș.-C.A., B.-C.C., M.C., I.B. and I.M.; software Ș.-C.A., B.-C.C., M.C., M.M. and I.M.; validation, Ș.-C.A., B.-C.C., M.C. and M.M.; writing—original draft preparation, Ș.-C.A., B.-C.C., M.C, I.B.; writing—review and editing, Ș.-C.A., B.-C.C., M.C., M.M., I.B. and I.M.; supervision and funding acquisition, I.B. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by a grant of the Romanian Ministry of Research and Innovation, CCCDI—UEFISCDI, project number PN-III-P1-1.2-PCCDI-2017-0272/Avant-garde Technology Hub for Advanced Security (ATLAS), within PNCDI III.

Data Availability Statement: Data used for testing the platform has been generated using the *dataplane* component. This component and the entire RESFIT platform can be found at https://github.com/atlas-iot/resfit (accessed on 19 November 2020).

Acknowledgments: The authors would like to thank the reviewers for all useful and helpful comments on our manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Patrono, L.; Atzori, L.; Šolić, P.; Mongiello, M.; Almeida, A. Challenges to be addressed to realize Internet of Things solutions for smart environments. *Future Gener. Comput. Syst.* **2020**, *111*, 873–878. [CrossRef]
2. Sicari, S.; Rizzardi, A.; Grieco, L.A.; Coen-Porisini, A. Security, privacy and trust in Internet of Things: The road ahead. *Comput. Netw.* **2015**, *76*, 146–164. [CrossRef]
3. Aberer, K.; Despotovic, Z. Managing Trust in a Peer-2-Peer Information System. In *CIKM '01, Proceedings of the Tenth International Conference on Information and Knowledge Management, Atlanta, GA, USA, 5–10 October 2001*; Association for Computing Machinery: New York, NY, USA, 2001; pp. 310–317. [CrossRef]
4. Bica, I.; Chifor, B.; Arseni, S.; Matei, I. Multi-Layer IoT Security Framework for Ambient Intelligence Environments. *Sensors* **2019**, *19*, 4038. [CrossRef] [PubMed]
5. Bica, I.; Chifor, B.; Arseni, S.; Matei, I. Reputation-Based Security Framework for Internet of Things. In *Innovative Security Solutions for Information Technology and Communications*; Simion, E., Géraud-Stewart, R., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 213–226. [CrossRef]
6. Chifor, B.; Arseni, S.; Matei, I.; Bica, I. Security-Oriented Framework for Internet of Things Smart-Home Applications. In Proceedings of the 2019 22nd International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania, 28–30 May 2019; pp. 146–153. [CrossRef]
7. Zhang, P.Y.; Zhou, M.C.; Fortino, G. Security and trust issues in Fog computing: A survey. *Future Gener. Comput. Syst.* **2018**, *88*, 16–27. [CrossRef]
8. Tormo, G.D.; Mármol, F.G.; Pérez, G.M. Dynamic and flexible selection of a reputation mechanism for heterogeneous environments. *Future Gener. Comput. Syst.* **2015**, *49*, 113–124. [CrossRef]
9. Ganeriwal, S.; Balzano, L.; Srivastava, M.B. Reputation-Based Framework for High Integrity Sensor Networks. *ACM Trans. Sen. Netw.* **2008**, *4*, 1–37. [CrossRef]
10. Fang, W.; Zhang, C.; Shi, Z.; Zhao, Q.; Shan, L. BTRES: Beta-based Trust and Reputation Evaluation System for wireless sensor networks. *J. Netw. Comput. Appl.* **2016**, *59*, 88–94. [CrossRef]
11. Wu, X.; Huang, J.; Ling, J.; Shu, L. BLTM: Beta and LQI Based Trust Model for Wireless Sensor Networks. *IEEE Access* **2019**, *7*, 43679–43690. [CrossRef]
12. Fang, W.; Cui, N.; Chen, W.; Zhang, W.; Chen, Y. A Trust-Based Security System for Data Collection in Smart City. *IEEE Trans. Ind. Inform.* **2021**, *17*, 4131–4140. [CrossRef]

13. Fang, W.; Zhang, W.; Shan, L.; Ji, X.; Jia, G. DDTMS: Dirichlet-Distribution-Based Trust Management Scheme in Internet of Things. *Electronics* **2019**, *8*, 744. [CrossRef]
14. Son, H.; Kang, N.; Gwak, B.; Lee, D. An adaptive IoT trust estimation scheme combining interaction history and stereotypical reputation. In Proceedings of the 2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC), Las Vegas, NV, USA, 8–11 January 2017; pp. 349–352. [CrossRef]
15. Chen, J.; Tian, Z.; Cui, X.; Yin, L.; Wang, X. Trust architecture and reputation evaluation for internet of things. *J. Ambient Intell. Humaniz. Comput.* **2019**, *10*, 3099–3107. [CrossRef]
16. Alswailim, M.A.; Hassanein, H.S.; Zulkernine, M. A Reputation System to Evaluate Participants for Participatory Sensing. In Proceedings of the 2016 IEEE Global Communications Conference (GLOBECOM), Washington, DC, USA, 4–8 December 2016; pp. 1–6. [CrossRef]
17. Restuccia, F.; Das, S.K. FIDES: A trust-based framework for secure user incentivization in participatory sensing. In Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014, Sydney, Australia, 19 June 2014; pp. 1–10. [CrossRef]
18. Wang, X.O.; Cheng, W.; Mohapatra, P.; Abdelzaher, T.F. ARTSense: Anonymous reputation and trust in participatory sensing. In Proceedings of the 2013 IEEE INFOCOM, Turin, Italy, 14–19 April 2013; pp. 2517–2525. [CrossRef]
19. Neisse, R.; Steri, G.; Baldini, G. Enforcement of security policy rules for the Internet of Things. In Proceedings of the 2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), Larnaca, Cyprus, 8–10 October 2014; IEEE Computer Society: Washington, DC, USA, 2014; pp. 165–172. [CrossRef]
20. Botta, A.; de Donato, W.; Persico, V.; Pescapé, A. Integration of Cloud computing and Internet of Things: A survey. *Future Gener. Comput. Syst.* **2016**, *56*, 684–700. [CrossRef]
21. Hussain, Y.; Akbar, M.A.; Alsanad, A.; Alsanad, A.A.; Nawaz, A.; Khan, I.A.; Khan, Z.U. Context-Aware Trust and Reputation Model for Fog-Based IoT. *IEEE Access* **2020**, *8*, 31622–31632. [CrossRef]
22. Kohlhoff, C. Boost.Asio. 2021 Available online: https://www.boost.org/doc/libs/1_76_0/doc/html/boost_asio.html (accessed on 23 July 2021).

Short Biography of Authors

Ștefan-Ciprian Arseni leads the Software Applications Security Laboratory in the Advanced Cyber Security Technologies Center of Excellence, "Ferdinand I" Military Technical Academy of Bucharest. He obtained their B.S degree in Computer Science from the "Ferdinand I" Military Technical Academy of Bucharest in 2011, the M.S degree in Electronics and Telecommunications from the University "Politehnica" of Bucharest in 2014 and is currently working towards a Ph.D degree in Electronics and Telecommunications from the University "Politehnica" of Bucharest with a thesis topic focusing on Security in Internet of Things. His research interests include, among others, Internet of Things, Cryptography and Cybersecurity. He led, as project manager, and participated, as a researcher, in a number of national funded projects in the area of Security, Biometrics, Cloud and Internet of Things.

Bogdan-Cosmin Chifor is a software architect with expertise in security, operating systems, networking and embedded systems. He has obtained their Ph.D from the "Ferdinand I" Military Technical Academy of Bucharest in the field of IoT security. His field of interest also includes electronic signature schemes, communication protocols, applied cryptography and developing security solutions. Mr. Chifor has 8 years of experience in development and research. He published 14 security related papers in international conferences and journals and is the co-author of 7 pending US patents related to security and network protocols. During the last 8 years he implemented and developed many security software and hardware solutions related to data encryption, digital signatures, PKI enabled applications, FPGA security solutions, cryptographic IPSec appliances, embedded development for network equipment and Android VoIP security solutions. He was involved in several research projects related to FPGA security development, Android security solutions and IoT security frameworks.

Mihai Coca leads the Computer Science and Cyber Security Laboratory at "Ferdinand I" Military Technical Academy, Bucharest, Romania. He received the B.S. and M.S. degrees in computer science and information security from the "Ferdinand I" Military Technical Academy, in 2015, respectively, 2017. He is currently working towards a Ph.D. degree in electronics, telecommunications and information technology with University Politehnica of Bucharest (UPB), Bucharest, Romania. His current research interests include earth observation multispectral and SAR processing based on temporal analysis in image time series, change detection, and anomaly detection. Supplementary research interests include decentralized and collaborative AI based on blockchain technologies and machine learning algorithms for embedded systems.In 2018, he performed a three-month internship with the German Aerospace Center (DLR), Oberpfaffenhofen, Germany as visiting Ph.D. student with the Remote Sensing Technology Institute (IMF). In the same year, he participated on BigSkyEarth Training School, which took place at the Vicomtech Research Center, San Sebastian, Spain.

Mirabela Medvei works in the Cyber Security Laboratory at "Ferdinand I" Military Technical Academy of Bucharest. She obtained her B.S. degree in Computer Science and Information Security from the "Ferdinand I" Military Technical Academy of Bucharest, in 2018 and the M.S. degree in Computer Science and Information Technology from University "Politehnica" of Bucharest in 2020. Her field of interest includes FPGA based accelerators for compute-intensive applications, FPGA based solutions to accelerate network security management, SDN, embedded systems and Parallel Processing Models.

Ion Bica is Professor at "Ferdinand I" Military Technical Academy of Bucharest, where he leads the Faculty of Information Systems and Cyber Security. He received their Ph.D. in Computer Science in 2004 with a thesis on electronic signatures and trusted third parties. He is teaching courses about computer networks, cryptographic algorithms and protocols, and security evaluation of information systems. His current research interests include design and implementation of efficient security solutions for cloud and IoT. He published 6 books and more than 80 papers in journals and conference proceedings, co-edited 2 conference volumes published by Springer and delivered invited talks at many universities and international conferences. He is an active member in several COST Actions and NATO Working Groups on cyber security.

Ioana Matei has obtained her B.S. degree in Computer Science and Information Security from the "Ferdinand I" Military Technical Academy of Bucharest in 2016 and her MS degree in Computer Science and Information Technology from University "Politehnica" of Bucharest in 2018. She is currently enrolled as a Ph.D student in the "Ferdinand I" Military Technical Academy of Bucharest, with a research topic on IoT security. She is Co-Founder and CTO of Ridesafe, an IoT startup that is developing automotive devices. Over the time, she was involved in different pilot projects around the IoT area, like Smart Houses, Pulse Detection Device or Smart Pill Dispenser, projects that aimed to improve people's lives by using the IoT technology.