Check for updates

Research Article

International Journal of Distributed Sensor Networks

International Journal of Distributed Sensor Networks
2020, Vol. 16(2)
© The Author(s) 2020
DOI: 10.1177/1550147720908199
journals.sagepub.com/home/dsn
SAGE

Trade-offs involved in the choice of cloud service configurations when building secure, scalable, and efficient Internet-of-Things networks

Amitabh Mishra, Thomas Reichherzer, Ezhil Kalaimannan, Norman Wilde and Ruben Ramirez

Abstract
This article focuses on results obtained from two cloud-based models that examine trade-offs between security, scalability, and efficiency of data collection for Internet-of-Things sensor networks. This work can provide insight for Internet-of-Things systems designers in choosing security controls and scalability features when working with cloud services. The results were obtained from a smart home Internet-of-Things prototype system in which data records from in-home sensors are transmitted wirelessly to an in-home hub, which forwards them to a cloud web service for storage and analysis. We consider different configurations and security controls on the wireless (in-home) and on the wired (home-to-web) sides. The configuration on the wireless side includes encrypted or plain-text transmission from the wireless sensors to the in-home hub for probing if software encryption of sensor data adds appreciable delay to the transmission time. The configuration on the wired side includes encryption or plain-text transmission, with or without authentication, with or without scalable cloud services. For each configuration, we measure end-to-end latency, transmission latency, and processing latency at the web service. Results of the experiments on the wired side showed much greater latencies and variability of latencies when using scalable cloud services.

Keywords
Internet of things, cloud computing, security, scalable services, encryption

Date received: 2 October 2019; accepted: 22 January 2020

Handling Editor: Yanjiao Chen

Introduction to smart homes and the Internet-of-Things

Smart devices and the Internet-of-Things (IoT) are increasingly pervading our daily lives shaping and transforming our experiences with the physical world. In the home, smart devices add conveniences, comfort, or safety. They regulate environmental conditions in the home by tracking our activities to infer our needs for cooling and heating while optimizing energy consumption. They keep track of our fitness and health conditions[1] or provide new services for controlling and maintaining devices in the home.[2] Market studies forecast an exponential growth of IoT consumer devices with more than 13 billion new devices connected to our networks by 2020.[3] Those devices may leverage wireless networks available in a home to

Department of Computer Science, University of West Florida, Pensacola, FL, USA

Corresponding author:
Amitabh Mishra, Department of Computer Science, University of West Florida, 11000 University Parkway, Building 4/232, Pensacola, FL 32514, USA.
Email: amishra@uwf.edu

Creative Commons CC BY: This article is distributed under the terms of the Creative Commons Attribution 4.0 License (https://creativecommons.org/licenses/by/4.0/) which permits any use, reproduction and distribution of the work without further permission provided the original work is attributed as specified on the SAGE and Open Access pages (https://us.sagepub.com/en-us/nam/open-access-at-sage).

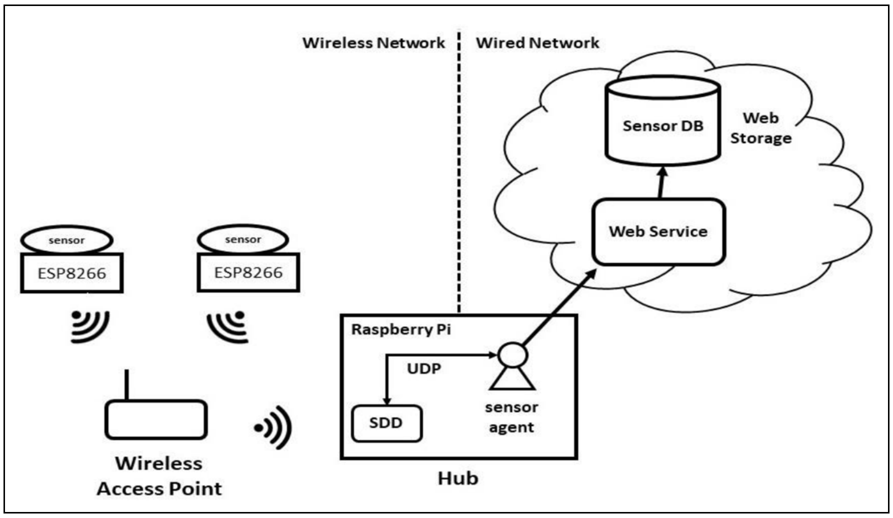

Figure 1. IoT/SMILE home network overview.

connect to the Internet and integrate with cloud services. This integration enables smart devices to access software and firmware updates as they become available, tap into vast cloud computing and data storage services, or offer automation and remote monitoring and control for residents of a home.

For smart homes to adapt to user needs and provide conveniences, comfort, and safety in the home, data from the smart devices must be shared and integrated to build a complex model of a home and its residents. These models can be used to track activities and assess the resident's needs and make recommendations or take actions in situations where an emergency arises. For example, a smart house that detects a water leak in the basement while the resident is at work can instantly notify the resident through text messages while also shutting off the main water valve of the house to prevent flooding in the home.

With limited data storage and processing capabilities, smart devices normally rely on cloud services to handle their data. Those services provide the scalability and reliability to ensure that the smart home is responsive to the resident's needs. Smart devices either directly connect to cloud services to upload their data or use a hub in the home that collects the data from multiple smart home devices and uploads them to the cloud (see Figure 1). As devices vary in their needs to communicate with cloud services to send data back and forth, we can expect a wide range of communication requirements for bandwidth and latency.

Studies have shown that smart IoT devices offer minimal to no protection from cyberattacks that may compromise the device or the data that it stores or transfers into the cloud.[4] Security-related standards and regulations are not well established yet, resulting in numerous legacy communication protocols which may not provide ample protection to data privacy. Moreover, the overhead of such legacy protection for data storage and transmission is not well understood.

In response to these challenges, as a continuation of our previous work in Reichherzer et al.,[5] we report on two cloud-based models that examine design alternatives that provide trade-offs between security, scalability, and efficiency for IoT networks. The study was performed on a prototype smart home system using multiple sensors in a network developed at the University of West Florida. This sensor network is also utilized in another research project intended at developing a Smart Independent Living for Elders (SMILE) home.[6]

We used this experimental platform for our two cloud-based models. The first considered different security controls for the wireless part of the network on the left side of Figure 1. The second considered different controls and designs for the wired part on the right side of the figure. In both cases, for simplicity of measurements, we consider unidirectional data flow from the sensor, through the sensor agent and the hub, to the Web Store which consists of a set of several web services with a backend data storage. As our indicator of efficiency, we use latency, defined as the overall elapsed time between message generation and receipt. We used the Network Time Protocol (NTP) to guarantee synchronization among clocks between the different system components.

In this article, section "Smart home/IoT security and scalability context" provides background on the cloud-based model, specifically the security issues that are being studied and those not studied. Section "The SMILE home system architecture" describes the architecture of the prototype SMILE home system including the wired and the wireless components in the system and the web services and data storage facilities on which it relies. Section "Related works on IoT network security" describes the related work on IoT security, while sections "Wireless network experiment" and "Wired network experiments" describe the cloud-based model for the wireless and wired sections of the system. Finally, section "Discussion and conclusion" discusses the experimental results and conclusions of the cloud-based model.

Smart home/IoT security and scalability context

The adoption of IoT networks in general, and of smart home networks, introduces significant security risks. In this article, we consider design alternatives that address an important subset of these risks. In the terminology used by NIST,[7] we consider an outsider adversary with high to very-high capabilities such as an experienced and well-financed hacker group. Such an adversary may be seeking to capture valuable data about smart home occupants or to compromise network components for use in denial-of-service (DoS) attacks. The adversary will have the capability to intercept and interfere with wireless and possibly wired communications (see Figure 1). However, we exclude from consideration insider attacks on the producers of IoT components since they are not addressable by network design but rather by other means such as a careful selection and monitoring of personnel. Similarly, we will not consider physical attacks on the sensors or other devices within the smart home since these are best addressed by physical security and by introducing tamper-proof components into each device. Finally, we largely exclude attacks on vulnerabilities in the web service itself or in the web storage facility since these classes of attack are not specific to IoT applications. Thus, our focus is on evaluating design approaches that provide secure communications, both wireless and wired within a home.

We organize our analysis of the different design approaches around the traditional CIA Information Security Triad of confidentiality, integrity, and availability attributes.[8] Related to availability is a fourth attribute of scalability; in many cases, IoT networks will need to sustain availability as data volume grows, either gradually as more nodes are added or at extreme events such as a flood or a strike.

Confidentiality is an essential attribute in almost all IoT implementations. In a smart home for example, sensors measuring weight, illumination, or temperature can divulge living habits of the occupants, affecting their privacy. Availability and integrity become prominent considerations contingent on the use case. Failures in monitoring the refrigerator to reorder milk would be more a nuisance than a disaster. However, if a fall detector on a resident in an assisted living home is programmed to call for an emergency ambulance service, system availability must permit continued operation even in DoS attack scenarios while data integrity ensures defense against spoofing.

The design approaches we used also impact the scalability of the IoT network, but the scalability obligations depend significantly on the IoT application. If the smart homes are located within a single gated community, the associated volume of data can be anticipated. However, smart-city-wide IoT applications must be able to handle huge and flexible data volumes with great reliability. Such challenges can be better addressed through cloud infrastructures.

The SMILE home system architecture

The design alternatives we evaluate in this article were implemented within the general IoT network architecture called "Smart Independent Living for Elders" that is shown in overview in Figure 1. An important design criterion for this architecture is the ability to update after deployment. A smart home must be able to incorporate new sensors and technology without disruption. Changes to the sensor hardware that enhance system performance and reliability must disrupt the system's response, data collection, or processing. Similarly, for the software, bug or security fixes or needed upgrades must be installable without disruption.

Another important design criterion is that significant computation capacity may be needed to correlate sensor data and make any needed near-real-time decisions. However, the devices in the home are unlikely to have appreciable storage and processing power, so most processing must be moved to the cloud.

To meet these goals, the SMILE architecture is created using distributed processes modeled through software agents that either run as part of the web service in the cloud or execute in the home on the home's central computing device, which acts as a hub to collect sensor data and or interact with residents in the home. All SMILE devices like sensors, widgets for browser visualization, or actuators are operated using software agents. Each agent wraps the technology and offers an interface for data collection and device control to generate a smart home response. To handle updates, new or updated agents would be signed to establish authenticity and pushed to the hub or the cloud.

The sensor boxes in our experiments are created using commonly available sensors and ICs such as single board computer nodes, temperature, light, pressure, and infrared sensors. Separate sensor boxes are assembled using electronic components for creating the sensors. The sensors gauge environmental data and the ESP8266 IoT node in each box collects and sends the data using the Message Queuing Telemetry Transport

(MQTT) application layer protocol for reliable data delivery. All wireless devices transport data in the home through a wireless access point to the Raspberry Pi, which acts as the sensor hub.

The data packets sent by the mesh network are received by a Raspberry Pi platform running the Sensor Data Distribution (SDD) software on a continuous basis. The packets are then forwarded by the SDD over an internal User Datagram Protocol (UDP) channel to sensor agents which are sensor processing units running on the Pi platform. The sensor agents can be restarted remotely, replaced or shut down as required in the process of smart home data collection.

The sensor agents running on the Pi hub have wired Internet communication to a web service in the cloud. For the experiments reported on in this article, the web service simply stored the sensor values in a sensor database. In a complete SMILE system, the web service would process the accumulated smart home data utilizing machine-learning algorithms to deduce activities in the home in order to provide smart assistance for assisted living as required. The web service and the sensor agents are implemented using Java EE and Java, respectively.

Related works on IoT network security

There seems little reason to doubt that the scale and reach of IoT systems will continue to expand rapidly. Many modular, application-specific IoT devices are in the market today, incorporating dedicated hardware combined with network connectivity to the cloud for data collection and processing. Bain and Co. forecast that the IoT market will grow to US$520 billion in 2021,[3] while IDC estimates the spending on IoT to reach 1.2 trillion by 2022.

There are, however, multiple technological and social hurdles that need to be addressed.[9] Prominent problems for IoT acceptability include making devices smarter by facilitating their adjustment and autonomous performance; ensuring security, privacy, and trust; and empowerment of total interoperability of interconnected IoT gadgets. A basic consideration is the security of personal data. This issue is gaining prominence with the growing awareness of the possible impacts of loss of privacy.[10] While some possible standards have started to emerge on the horizon, there still seems to be no consensus among IoT product manufacturers regarding the implementation of security in IoT devices.[11] The US National Intelligence Council lists IoT as one of the six civil technologies that are potentially disruptive and could impact US national power.[12]

Several serious security incidents involving IoT have already occurred. Millions of healthcare records have been leaked,[13] the signaling system for the CSX Train network was attacked,[14] a variety of new vehicle models were hacked forcing recalls,[15] the readings of smart energy meters were altered,[16] and a major portion of the Internet was brought down through distributed denial-of-service (DDoS) attacks from baby monitors.[17]

Such incidents imply that serious security considerations need to be incorporated into the process of IoT system design. The design must contemplate both the security of the IoT devices and the issues raised when such devices are connected to a cloud platform. Cloud services have considerable advantages due to the lack of restrictions on fixed infrastructure and features such as location transparency, service abstraction, and dynamic scalability. However, security considerations become crucially important on the cloud due to enormous storage requirements combined with the need for fast access and information processing.[18]

A detailed survey on the considerations and issues in IoT security is offered by Jing et al.[19] The article also investigates and suggests ways to heterogeneously incorporate IoT products and services across layers. Security becomes a vital design objective for data-in-transit security, where data are encrypted and hashed before transmission over the Internet. This processing has a considerable influence on performance which has been overlooked by researchers so far.[20] Müller et al.[21] describe research to ascertain the performance impact of security strategy decisions in arbitrary cloud database systems and the impact of using certain combinations of encryption and hashing algorithms in Cassandra and DynamoDB.

Motivated by the innovations available through cloud computing, Kardas et al.[22] proposed a security and privacy model for radio-frequency identification (RFID) systems merged with the cloud computing model to enhance scalability and performance while preserving security and privacy of the system. The work reported by Sun et al.[23] primarily aims to emphasize the major security, privacy, and trust concerns in current cloud computing systems and assist users in recognizing the tangible and intangible threats they present.

Few works in the literature studied the various trade-offs in IoT environments. Zhang et al.[24] use an unreliable link model to investigate the trade-off between the energy consumption and wireless communication in a multihop network. The framework in Tasiopoulos et al.[25] surveys delay-tolerant networks (DTNs) for trade-offs between the packet delivery delay and types of packet transportation costs in wired network and mobile wireless network environments.

The research presented in Mukherjee et al.[26] discusses the wide-stretching application security demands for IoT and varied network-edge resource limitations for end-to-end cloud-fog communication. The authors present a secure end-to-end protocol for resource-limited devices by adjusting security functionality used in unconstrained devices. Similar work was reported in Tewari and Gupta,[27] which analyzes the security problems of each layer in the Transmission Control Protocol/Internet Protocol (TCP/IP) model and the issues related to cross-layer heterogeneous integration and security.

Wireless network experiment

The instrumentation setup starts with the wireless sensors that sense ambient physical parameters and relay the data to the next stage in the sequence, the sensor agent over a wireless link. The following section explains the details of the wireless network configurations.

Experimental setup

A smart home involves a limited, manageable number of IoT sensors and actuators making scalability in the wireless network a non-issue but the same cannot be asserted about the integrity and confidentiality aspects of the application data. The IoT network is prone to spoofing and sniffer attacks for collecting unauthorized data and disruption of applications and services. Resorting to encryption of IoT sensor and actuator data using a shared key working, the sensor and the sensor agent could take care of integrity and confidentiality.

Sensor agents are deployed to receive transmitted sensor data relating to physical quantities on wireless links from sensor nodes in a smart home. By default, these transmissions are plain text and are vulnerable to tapping by a tuned receiver, making listening and man-in-the-middle attacks a potential threat. To add application-level security and reduce the risks of such threats to this data, we encrypted it prior to transmission. Due to high frequency and hence high amount of generated data, the time taken at the smart sensor nodes to encrypt sensor data could add additional encoding latency in addition to local processing delay. In our experiment, we measured the mentioned delay in communication post encryption and contrasted the throughputs with and without encryption.

Managing the energy consumption in wireless IoT sensors and actuators is an important consideration for smart home designers because of battery-based power of such sensors and actuators. The limitation of powering remote IoT sensors and actuators makes it essential for the designers to also keep battery life and the number of replacements in perspective. The power consumption in sensors involves uses in transduction, analog-to-digital conversion, and further processing followed by transmission and reception. Encrypting sensor data would mean more power required for processing, adding to the power consumption. This additional energy consumption due to encryption can be found out by checking the battery drain. This battery consumption can be optimized.[28]

In the experiments for the wireless network configuration, we used ESP8266-based sensor boards. The ESP8266 is a low-cost, 32-bit, Wi-Fi-enabled microcontroller chip running full, integrated TCP/IP protocol stack. It requires nominal external circuitry and incorporates the microcontroller, standard digital peripheral interfaces, filters, amplifiers, RF balancing unit, antenna switches, and power management units in the same package. It can be used to provide cost-effective, low-power, highly integrated Wi-Fi solutions.

The ESP8266 system-on-chip (SoC) microcontroller board comes with a 2.4-GHz Wi-Fi (802.11 b/g/n, supporting WPA/WPA2) built into it. It is apt for building applications for wireless sensor networks and WiFi-enabled IoT systems. Our experimental setup used a specific model of ESP8266, the ESP-12E, to study the performance of an IoT node in encrypting and sending a stream of data over the TCP/IP protocol stack.

Our IoT network utilizes the MQTT protocol as the application layer protocol for reliable data delivery.[29] MQTT is a widely used protocol best suited for message delivery to remote locations that have low network bandwidth and/or low processing capabilities. It is a Publish–Subscribe protocol which means that any nodes subscribed to a topic on the MQTT broker receive all the messages published to that topic by any other node that publishes to it. This project uses the Mosquitto as the MQTT broker and PubSubClient as the client.

The reason behind the choice of MQTT as against a more secure MQTT-SN (MQTT for Sensor Networks) was the end-to-end travel of data from the sensors to the cloud which could not be done completely through datagrams for reasons of error and inefficiency. While MQTT-SN would be a good choice for a secure protocol at the local network level, using it in the links beyond the local network to the broker would not be a wise choice.

Moreover, not all brokers offer a robust support for Transport Layer Security (TLS) now. The authors tried to keep the scenarios general, and not specific. In subsequent research work, the scenarios with TLS can be implemented and compared for the added TLS overhead. If the scenarios were limited to the local network, MQTT-SN would certainly be an added level of security, but the scenarios modeled go up to the cloud. Hence, the protocol chosen had to be one that works locally, as well as up to the broker.

The network was setup in two different configurations to measure the latency, reliability, and consistency of the data transmitted over the network. Each of these two networks consists of a sensor node running the PubSubClient, a WiFi router, and a receiving node that hosts the MQTT broker. The two configurations were

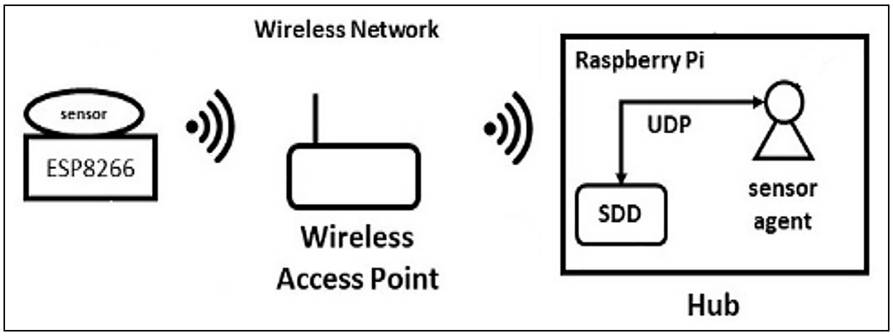

Figure 2. IoT wireless sensor network configuration 1.

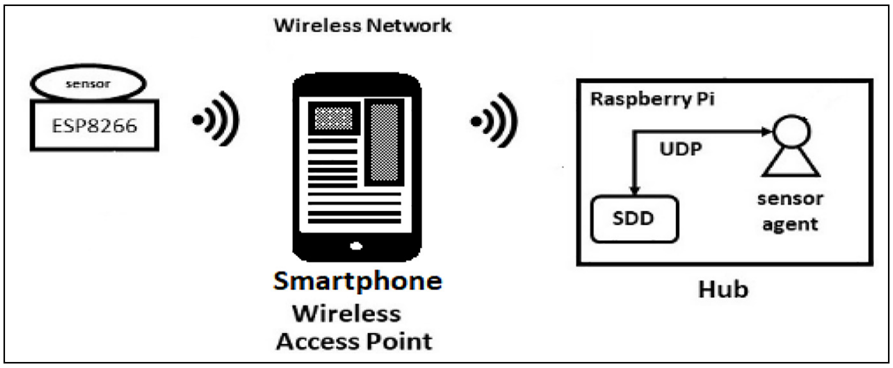

Figure 3. IoT wireless sensor network configuration 2.

chosen because one of the configurations uses a local router working using WiFi connectivity, while the cellphone hotspot implements the scenario for 4G LTE connectivity providing us with two sets of data transmission parameters. We have not tried to compare the transmissions, because the performance of the two transmission mechanisms would depend on other factors such as bandwidths of the connections, infrastructure support, LTE coverage, interference, and mobility. The two configurations are as follows:

1. ESP8266 (sensor node)—Netgear N450 (WiFi router)—Raspberry Pi 3 (receiving node): the Raspberry Pi 3 was installed with Mosquitto broker and configured to listen at 1883 for MQTT connections, as shown in Figure 2.
2. ESP8266 (sensor node)—Android smartphone mobile hotspot (WiFi router)—Raspberry Pi (receiving node): an Android smartphone was configured to serve as a WiFi hotspot at a unique SSID to receive connections from only known hosts. This setup was used to simulate an isolated network for getting unaffected readings for the latency of the data transfer, as shown in Figure 3.

In both the network configurations, the ESP8266 was flashed with the same code involving the transfer of a dataset comprising 10,000 24-byte string values. In each case, after establishing a WiFi connection with the router, ESP-12E would start sending the assigned number of 24-byte strings over MQTT after one or two rounds of Advanced Encryption Standard–Cipher Block Chaining (AES-CBC) encryption on every string of data. An AES library was used for encrypting the 24-byte strings. The AES block size used was 128 bits for both types of encryptions. Although the authors have not used too many sensors in their setup, in a real smart home scenario, there could be ten to hundreds of sensors in a home or neighborhood. A block cipher was used by us because key management could become a possible issue with stream ciphers due to high number of key requirements while scaling the network up to more number of sensors. For both the configurations, the same size datasets were used to obtain the latency values for the data transfer. Multiple iterations were performed on both the configurations to obtain statistical consistency in the readings. Several observations were made regarding the behavior of the IoT network depending on the network activities of the home-based network, considering it a traditional smart-home system network.

Table 1. Time taken (in milliseconds) for the data transmission per message.

Trial	Medium: home router (delay per message in ms)			Medium: cellphone hotspot (delay per message in ms)		
	No encryption	AES-CBC single encryption	AES-CBC double encryption	No encryption	AES-CBC single encryption	AES-CBC double encryption
1	0.169	0.2	0.336	0.231	0.253	0.363
2	0.183	0.208	0.314	0.238	0.265	0.361
3	0.169	0.202	0.331	0.252	0.277	0.357
4	0.168	0.189	0.317	0.238	0.281	0.366
5	0.18	0.223	0.31	0.242	0.269	0.368

AES: Advanced Encryption Standard; CBC: Cipher Block Chaining.

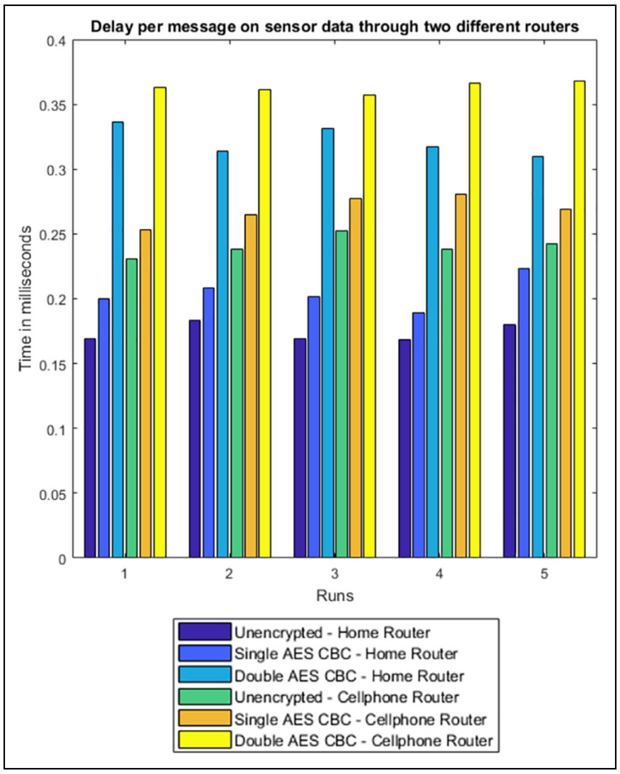

Figure 4. Comparison of latencies for wireless configurations.

Experimental results

Like several other off-the-shelf sensors commonly available, the sensors that we used for building the IoT smart home did not come with in-built hardware security. Security was an add-on feature to these sensors. To get a base time for raw, unencrypted data were sent and the transmission times were noted for the sensor values across 10,000 messages to get a good average. This was done for two different media using a home router for connection in one setup and the cellphone hotspot in the other. Five different runs of the experiment were performed to ascertain a range on variability of transmission time. Transmission times were also noted for the same number of messages in five different runs for encrypting and then sending the data. The data were then encrypted for the second time to improve the security and times were noted as before. Table 1 and Figure 4 show the findings of the experiments for the two connectivity media arrangements. As expected, a delay in the transmission time was found in case of single encryption, which increased for double encryption for both the media. However, it was observed that the overhead for encryption through software was not too high and was negligible for most real-time purposes, even with the cellphone being used as a router.

Wired network experiments

The wired portion of our IoT sensor network consists of the components on the right side of Figure 1, that is, the Raspberry Pi with its sensor agent, the cloud-hosted web service that receives sensor data, and the cloud-hosted sensor database that stores the sensor data. We ran experiments on four different configurations having different security and scalability attributes.

Configuration 1: base system and minimal security, with no scalability

In this configuration, an Amazon Web Services (AWS) EC2 running Ubuntu server is used for hosting the MySQL sensor database and the web service running on a Glassfish server, as proposed in Reichherzer et al.[5] The sensor agent communicates with the web service through unsecured Hypertext Transfer Protocol (HTTP) messages.

We judge confidentiality, integrity, availability, and scalability to be poor in this configuration (Table 2). There is no confidentiality or data integrity because the HTTP connection is unsecured and could be monitored or spoofed. Availability is limited because the single AWS instance and its Internet endpoint provide a single point of failure. Some awkward limited vertical scaling is possible since the instance can be shut down and replaced by one with more compute power. But the service and database cannot be smoothly horizontally scaled by adding more instances.

Table 2. Summary of information security attributes.

Configuration	Confidentiality	Integrity	Availability	Scalability
1	Poor	Poor	Poor	Poor
2	Moderate	Moderate	Poor	Poor
3	Good	Good	Poor	Poor
4	Good	Good	Good	Good

Configuration 2: message security, with no scalability

In this configuration, we host the web service and the sensor database as before, on a single AWS instance using the same operating system, application server, and database server. However, the Glassfish application server is now configured for HTTPS (Secure Sockets Layer—SSL) messaging utilizing a certificate signed by a well-known certification authority.

In this configuration, moderate confidentiality and data integrity are provided by the encrypted SSL messaging. The signed certificate guarantees that messages are being sent to the correct endpoint. However, there is still no authentication of the sensor agent, so the web service has no such guarantee as to the source of the data it is receiving. Availability and scalability continue to be poor, for the same reasons given for configuration 1.

Configuration 3: message security and agent authentication, with no scalability

This configuration builds on the previous one by assigning credentials to the sensor agent consisting of a username and password. These are sent as part of each message and are used by the web service to authenticate the source of incoming messages. Authentication is done using the same MySQL database server which is given an additional table to hold usernames and hashed passwords.

We would rate the confidentiality and data integrity characteristics of this configuration as good (Table 2). It shares with configuration 2 the encrypted SSL messaging. In addition, authentication is now bidirectional, with both certificate authentication of web service to sensor agent and password authentication of sensor agent to web service. Thus, the web service now has some assurance of the integrity of the data it receives. If, for example, a sensor agent were compromised, its credentials could be revoked to limit the damage it could do. The availability and scalability characteristics of this configuration continue to be poor due to the continued reliance on a single hardware instance.

Configuration 4: commercial security and scalability

Several companies now provide computing infrastructure that allows developers to create software applications that are both secure and scalable. Our fourth configuration relies on such an infrastructure, the collection of AWS.

In this configuration, the sensor agent, instead of transmitting to the web service directly, sends its messages to an HTTPS endpoint for AWS Kinesis Streams. This is a service for real-time handling of large streams of data records.[30] Capacity can be scaled up or down on-the-fly by adding or removing "shards," each providing a certain volume of message processing capacity.[31]

The web service is replaced by a Java function, compiled into a jar file and deployed to the AWS Lambda service.[32] AWS Lambda is an autoscaling serverless compute service that runs event-based code fragments. Lambda polls our Kinesis stream and triggers a call to the Java function when a batch of records is available for processing.

When triggered, the Java function reads input records from the Kinesis stream and writes them to a table in AWS DynamoDB, a managed NoSQL database service which acts as the sensor database. DynamoDB provides availability by automatically distributing traffic across multiple servers and by automatically replicating data across independent "availability zones" thus avoiding a single point of failure.[33] As with Kinesis, read/write capacity of a DynamoDB table can be scaled up or down on-the-fly.

Underlying Kinesis, Lambda, and DynamoDB is the AWS Identity and Access Management (IAM) service. This is a service that controls access to AWS resources by providing authentication and authorization functions.[34] The sensor agent authenticates to Kinesis using IAM-issued credentials. Similarly, the Lambda function uses an IAM-issued role to read from our Kinesis stream and to write to our DynamoDB table.

We rate the confidentiality and data integrity of configuration 4 as good since, combined with the use of HTTPS for messaging, the IAM-based authentications guarantee data confidentiality and integrity all the way from the sensor agent to the sensor database. Similarly, we rate availability as good, since all the AWS services claim high availability based on redundancy in their underlying infrastructure. (For highest availability, AWS encourages users to distribute their applications across multiple AWS availability zones, whose data centers are independent of each other.[35] We did not explore this kind of redundancy in our experiments.) Scalability is also good, since each service incorporates automatic or semi-automatic on-the-fly scaling.

Experimental procedure

For the wired network experiments, simulated sensor data messages were generated at the sensor agent on the Raspberry Pi hub and transmitted to the web service. All four configurations were tested. Each experimental

Table 3. Average end-to-end latency (ms).

Data rate	Low rate		Max. rate	
Message size (bytes)	24	1024	24	1024
Configuration 1: base, minimal security, no scalability	20.8	20.6	18.8	19.2
Configuration 2: message security, no scalability	20.8	23	18.4	23.3
Configuration 3: message security, agent authentication, no scalability	21.8	20.2	25.6	29.9
Configuration 4: commercial security and scalability	2101	1696	12,425.6	5899

Table 4. Average processing latency (ms).

Data rate	Low rate		Max. rate	
Message size (bytes)	24	1024	24	1024
Configuration 1: base, minimal security, no scalability	0.11	0.09	0.06	0.05
Configuration 2: message security, no scalability	0.06	0.07	0.04	0.06
Configuration 3: message security, agent authentication, no scalability	0.23	0.63	0.16	0.25
Configuration 4: commercial security and scalability	295	130	2118	1357

run lasted approximately 300 s. The following parameters were varied giving four runs for each configuration, that is, 16 runs in total:

- Data rate: *low rate*, five messages per second or *max rate*, as fast as the Raspberry Pi could send.
- Packet size: 24 bytes or 1024 bytes.

The clocks on the Raspberry Pi and the servers involved in the experiments were synchronized using NTP services, which allows for measured time differences to be attributable to processing and transmission latencies and not clock differences. The following times in milliseconds were collected for each message:

t1—message sent from hub;
t2—message received at the web service;
t3—write to data store completed.

The primary measure for evaluating each configuration is end-to-end latency, defined as t3 – t1. This measure would probably be the chief concern of an IoT system designer since it is the time between data generation and availability for processing. Secondary measures include transmission latency, defined as t2 – t1, and processing latency, defined as t3 – t2. These allow us to identify the relative importance of delays incurred in transmitting to the cloud as compared to delays occurring during processing within the web service and data store.

Experimental results

Tables 3 and 4 show the results of the 16 experimental runs. Comparing configurations 1 and 2, we can see that the average end-to-end latency penalty for using SSL message security is quite modest. For example, using the maximum data rate and the 1024-byte messages, the average time goes from 19.2 to 23.3 ms. Similarly, comparing configuration 2 and 3, the penalty for agent authentication remains small. Using the same example of maximum data rate and 1024-byte records, we go from 23.3 to 29.9 ms.

The situation changes radically when we compare configuration 4, with the commercial security and stability, to the earlier configurations. The end-to-end latency goes from 29.9 ms to values of over 5 s. To identify the source of the difference, we can look at the two components of end-to-end latency, that is, the transmission latency and the processing latency. Table 4 shows the average processing latency on the web service. Except for configuration 4, processing latency is almost zero, indicating that processing time is essentially trivial. This shows that the end-to-end delays come almost entirely in transmitting data from the sensor agent to the web service.

For configuration 4, while it is still true that most of the end-to-end latency is transmission latency, the processing latency is more significant, averaging between 130 ms and more than 2 s depending on the data rate and message size. For many IoT applications, the variability of latency may be just as important as the average latency. Variability is most easily seen graphically. Figure 5 plots the end-to-end latency in the maximum data rate runs for the three configurations. The first sample which was a peak with considerably more amplitude than the rest of the data and could be considered an impediment for showcasing the real nature of latency variations has been excluded from the graphs.

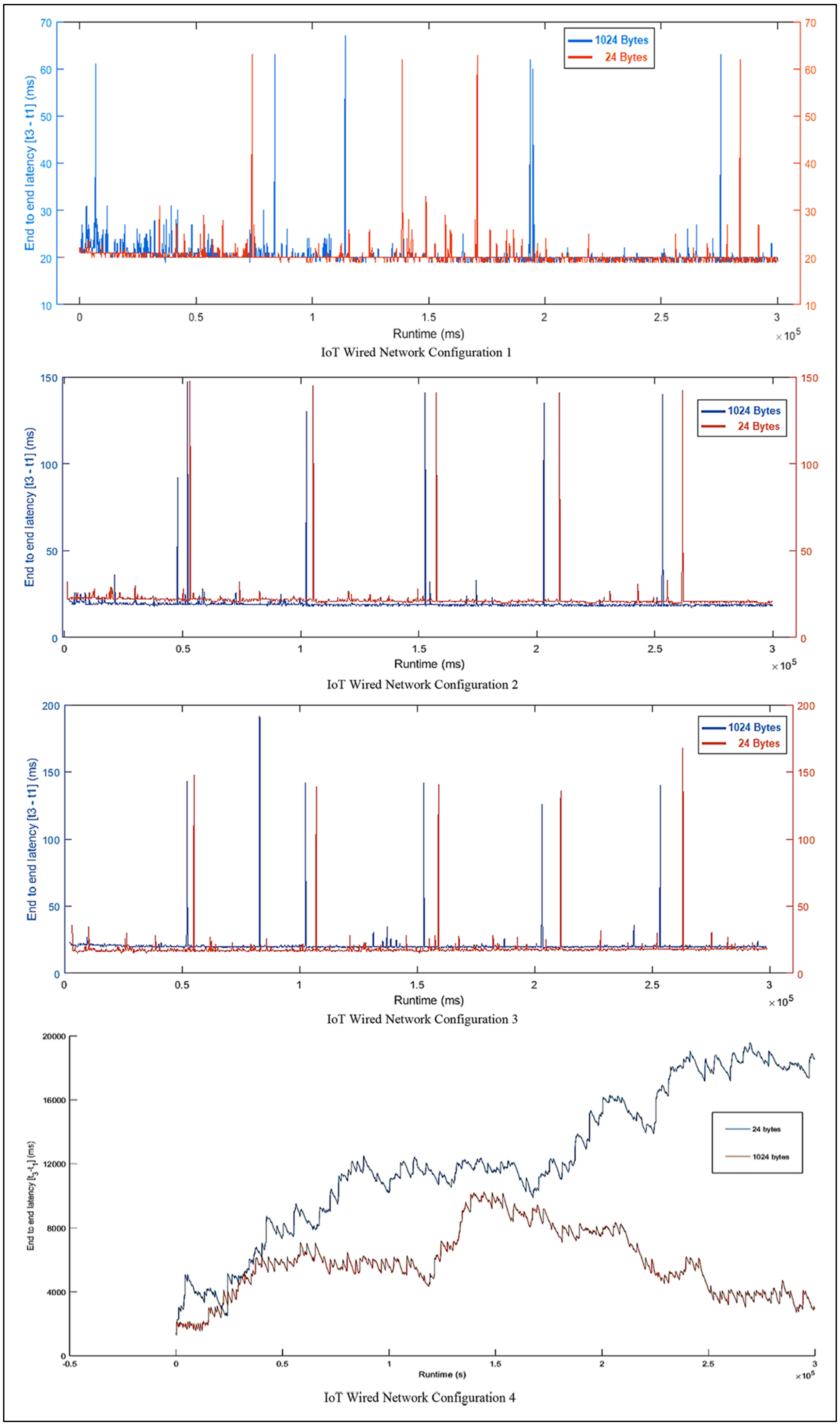

Figure 5. End-to-end latency plots—maximum data rate.
Plots have different vertical scales.

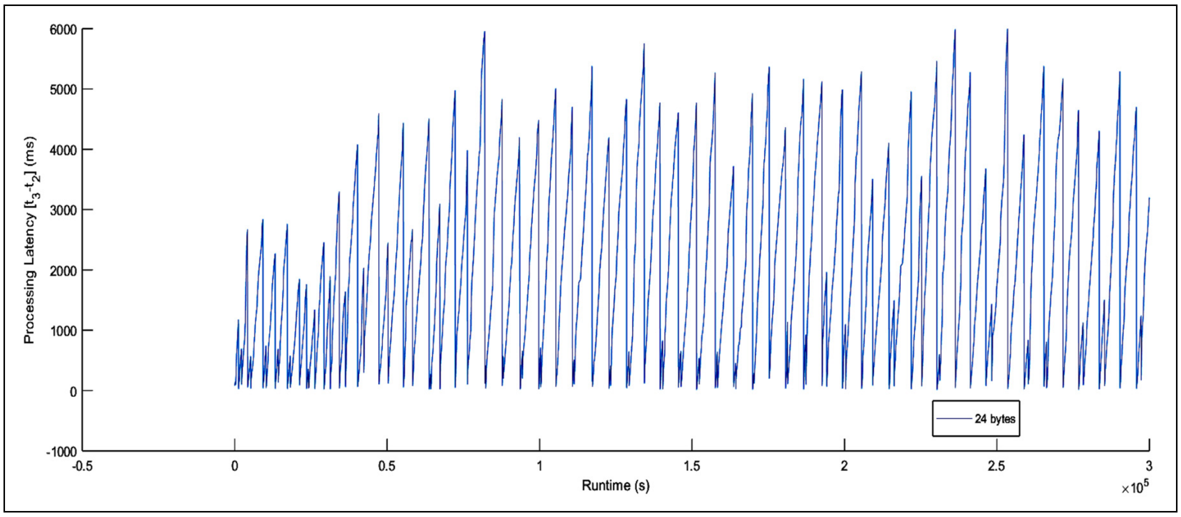

Figure 6. Configuration 4—processing time for 24-byte records.

The picture is broadly the same across configurations 1 to 3. The first message in each run has a large latency, presumably due to the cost of establishing the network connection. This initial latency is higher when HTTPS is used in configurations 2 and 3, which is expected because the session encryption key must be negotiated before transmission begins.

After the initial peak, each plot shows a roughly constant latency but with spikes from time to time. These presumably represent moments when resources are deallocated and re-allocated in the underlying computing infrastructure as connections are being re-established.

For configuration 4, with the commercial infrastructure, the picture is very different. The plot shows broad increases and decreases over the 300-s period of the run. Also, we see a spiky, almost sawtooth-like, behavior. Finally, there is substantial difference between processing of the 24-byte and the 1024-byte messages.

The greater complexity of the behavior of configuration 4 is perhaps natural considering the greater complexity of the processing needed in a commercial IoT infrastructure. Consider just one aspect, authorization. In configuration 3, authorization is accomplished simply by including a username and password in the sensor agent's message to the web service. The web service sends a query to a database table located on the same node and access is authorized if the hashed passwords match.

In the commercial AWS infrastructure, authorization happens multiple times for each message. The sensor agent presents credentials to Kinesis to request the right to use the stream. The Lambda function presents credentials to Kinesis to request the right to pull data from the stream and to DynamoDB to request permission to write to the data store. In each case, the IAM service is consulted to create a context around each request, to consult the different policy documents that govern authorization, and to arrive at a conclusion as to whether each request should be allowed or denied.[36] Compared to configuration 3, the IAM authorization process involves a lot of overhead and several opportunities for variability.

Variability is not only due to authorization and IAM. Kinesis, Lambda, and DynamoDB are also likely to require multiple message communications between multiple nodes to fulfill each task. These nodes are shared across the AWS user base, so processing time can be affected by concurrent actions of other AWS users sharing the same infrastructure.

The results suggest that AWS services likely use buffering and batching strategies to improve average response time, but with the consequence that latency times can have additional sawtooth-like variability as shown in Figure 6. This figure shows the time that configuration 4's web service required to process each sensor message, from message receipt until the DynamoDB service reported that the corresponding data message had been written. As can be seen, the processing latency varied systematically from near 0 to 6 s. This kind of variability has also been reported by Bermbach in his benchmark studies of cloud services and should perhaps be taken as normal by IoT application developers.[20]

Discussion and conclusion

Security implications create obstacles for wider acceptance of smart homes. As the smart home market grows, the attack surface for an IoT network within the home grows with it. Smart home applications face attacks ranging from snooping on transmissions and traffic analysis or leak of message contents to alteration, fabrication, disruption of communications through node-capture, routing attacks, or flooding. In several commercially available devices, sensitive data pertaining to parameters from the human body are transmitted over wireless or wired links with little or no security. The goal of this study was to explore the trade-offs in security solutions for IoT applications including smart home systems.

Our cloud-based model measured processing overhead while improving on security in the prototype network and adding more scalable services to the system. Our results hinge heavily on the effectiveness of the employed tools as well as the AWS. Alternative implementations may yield different performance findings on the efficacy of communicating sensor data to the Web Store with further processing. However, outside the productivity limitations of the code and hardware we used, the current tools and frameworks used in our experiments are industry standard and widely used. Thus, we hope that our results will be representative of what might be encountered in other smart home or IoT systems with regard to the price of security and scalability.

Real-time constraints on data collected in a smart home or any other IoT application will vary based on the specifics of the application. The aim of this study was to assess the impact of security overhead on real-time availability of data. The experiment has shown a dramatic difference between custom-built solutions and the use of scalable, secured commercial infrastructure solutions such as AWS. The overhead of adding security in customized solutions as illustrated in configurations 1 to 3 was minimal compared to AWS as measured in configuration 4. There was also a great variability in latency when using AWS. It is interesting to note that, from the experiments described in this article, AWS has announced specialized IoT development kits and services marketed as "IoT Core."[37] It would be interesting to see if the use of these services reduces the performance variabilities that we have identified.

In developing real-time systems, it is vital to consider the trade-off between scalability and robust security versus performance in adopting commercial solutions. As our experiments have shown, the greater variability can make a difference in how real-time IoT systems respond to events occurring in smart homes and other IoT applications. In most cases, security cannot be compromised but the platform can be chosen in accordance with the real-time constraints of the application.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

Supplemental material

Supplemental material for this article is available online.

References

1. Pantelopoulos A and Bourbakis NG. A survey on wearable sensor-based systems for health monitoring and prognosis. *IEEE T Syst Man Cy C* 2010; 40(1): 1–12.
2. Hoffman DL and Novak TP. How to market the smart home: focus on emergent experience, not use cases. *Soc Sci Res Netw* 2016, http://ssrn.com/abstract=2840976
3. Bosche A, Crawford D, Jackson D, et al. *Unlocking opportunities in the Internet of Things*. Bain and Company, January 2016, https://www.bain.com/contentassets/5aa3a678438846289af59f62e62a3456/bain_brief_unlocking_opportunities_in_the_internet_of_things.pdf/
4. Ennajjar I, Tabii Y and Benkaddour A. Security in cloud computing approaches and solutions. In: *2014 third IEEE international colloquium in information science and technology (CIST)*, Tetouan, Morocco, 20–22 October 2014. New York: IEEE.
5. Reichherzer T, Mishra A, Kalaimannan E, et al. A case study on the trade-offs between security, scalability, and efficiency in smart home sensor networks. In: *IEEE: 2016 international conference on computational science and computational intelligence (CSCI)*, Las Vegas, NV, 15–17 December 2016. New York: IEEE.
6. Reichherzer T, Satterfield S, Belitsos J, et al. An agent-based architecture for sensor data collection and reasoning in smart home environments for independent living. In: Khoury R and Drummond C. (eds) *Lecture notes in computer science*, vol. 9673. Berlin: Springer-Verlag, 2016, pp.15–20.
7. NIST. Guide for conducting risk assessments, SP 800-30 Rev. 1, September 2012, https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-30r1.pdf
8. Lu Z and Zhou Y. The evaluation model for network security. In: *2014 fourth international conference on communication systems and network technologies*, Bhopal, India, 7–9 April 2014. New York: ACM.
9. Botterman M. Internet of Things: an early reality of the future internet. Report of the Internet of Things Workshop, for the European Commission, Information Society and Media Directorate General, Networked Enterprise & RFID Unit—D4, Prague, May 2009.
10. Weber RH. Internet of Things—new security and privacy challenges. *Comp Law Secur Rev* 2010; 26: 23–30.
11. Giusto D, Iera A, Morabito G, et al. *The Internet of Things*. Cham: Springer, 2010.
12. National Intelligence Council. Disruptive civil technologies—six technologies with potential impacts on US interests out to 2025. In: *Conference report CR 2008–2007*, Washington, DC, April 2008.

13. Harries D and Yellowlees PM. Cyberterrorism: is the U.S. healthcare system safe? *Telemed E-Health* 2013; 19(1): 61–66.
14. Niland M. *Virus disrupts train signals*. The Associated Press, 2003, http://www.cbsnews.com/stories/2003/08/21/tech/main569418.shtml
15. Greenberg A. Hackers remotely kill a jeep on the highway—with me in it. *Wired Magazine*, 2015, http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/
16. Krebs A. FBI: smart meter hacks likely to spread. *Krebs on Security*, 2012, http://krebsonsecurity.com/2012/04/fbi-smart-meter-hacks-likely-to-spread/
17. Valerio P. Your baby monitor just collapsed the internet. *The ICT Scoop*, 25 October 2016, https://theictscoop.-com/your-baby-monitor-just-collapsed-the-internet-655f1d034dd9
18. Chen D and Zhao H. Data security and privacy protection issues in cloud computing. In: *Proceeding of the international conference on computer science and electronics engineering (ICCSEE '12)*, vol. 1, Hangzhou, China, 23–25 March 2012, pp.647–651. New York: IEEE.
19. Jing Q, Vasilakos AV, Wan J, et al. Security of the Internet of Things: perspectives and challenges. *Wirel Netw* 2014; 20(1): 2481–2501.
20. Bermbach D. Quality of cloud services: expect the unexpected. *IEEE Int Comput* 2017; 21(1): 68–72.
21. Müller S, Bermbach D, Tai S, et al. Benchmarking the performance impact of transport layer security in cloud database systems. In: *IEEE international conference on cloud engineering (IC2E)*, Boston, MA, 11–14 March 2014. New York: IEEE.
22. Kardas S, Celik S, Bingol MA, et al. A new security and privacy framework for RFID in cloud computing. In: *Proceedings of the 5th IEEE international conference on cloud computing technology and science (CloudCom '13)*, Bristol, 2–5 December 2013. New York: IEEE.
23. Sun D, Chang G, Sun L, et al. Surveying and analyzing security, privacy and trust issues in cloud computing environments. *Proc Eng* 2011; 15(1): 2852–2856.
24. Zhang R, Berder O, Gorce J, et al. Energy–delay tradeoff in wireless multihop networks with unreliable links. *Ad Hoc Netw* 2012; 10(1): 1306–1321.
25. Tasiopoulos A, Tsiaras C and Toumpis S. Optimal and achievable cost/delay tradeoffs in delay-tolerant networks. *Comp Netw* 2014; 70(1): 59–74.
26. Mukherjee B, Wang S, Lu W, et al. Flexible IoT security middleware for end-to-end cloud-fog communication. *Future Gener Comp Sy* 2018; 87(1): 688–703.
27. Tewari A and Gupta BB. Security, privacy and trust of different layers in Internet-of-Things (IoTs) framework. *Future Gener Comp Sy*. Epub ahead of print 1 May 2018. DOI: 10.1016/j.future.2018.04.027.
28. Mishra A and Agrawal DP. Energy conservation and lifetime optimization of wireless body sensor networks for 24x7 physiological parameters' monitoring. *J Commun* 2015; 10(9): 685–695.
29. Grgić K, Špeh I and Heđi I. A web-based IoT solution for monitoring data using MQTT protocol. In: *2016 international conference on smart systems and technologies (SST)*, Osijek, 12–14 October 2016. IEEE.
30. Amazon. What is Amazon Kinesis Streams? http://docs.aws.amazon.com/streams/latest/dev/introduction.html (accessed 20 August 2017).
31. Amazon. Amazon kinesis streams key concepts, http://docs.aws.amazon.com/streams/latest/dev/key-concepts.html (accessed 28 August 2017).
32. Amazon. AWS Lambda—product details, https://aws.amazon.com/lambda/details/ (accessed 20 August 2017).
33. Amazon. *Amazon DynamoDB developer guide*. API Version 2012-08-10, http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/dynamodb-dg.pdf (accessed 24 August 2017).
34. Amazon. AWS identity and access management user guide, http://docs.aws.amazon.com/IAM/latest/UserGuide/iam-ug.pdf (2017, accessed 24 August 2017).
35. Amazon. Global Infrastructure, https://aws.amazon.com/about-aws/global-infrastructure/ (accessed 28 August 2017).
36. Amazon. Understanding how IAM works, https://docs.aws.amazon.com/IAM/latest/UserGuide/intro-structure.html (accessed 31 May 2018).
37. Amazon. AWS IoT core features, https://aws.amazon.com/iot-core/features/ (accessed 9 November 2019).