6-6-2019

Overhead Management Strategies for Internet of Things Devices

Kavin Kamaraj

Santa Clara University

Department of Computer Engineering

Date: June 6, 2019

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Kavin Kamaraj

ENTITLED

Overhead Management Strategies for Internet of Things Devices

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

B. Dezfouli

Thesis Advisor
Dr. Behnam Dezfouli

Yuhong Liu

Thesis Reader
Dr. Yuhong Liu

N. Ling

Chairman of Department
Dr. Nam Ling

Overhead Management Strategies for Internet of Things Devices

Kavin Kamaraj

Dissertation

Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computer Engineering
in the School of Engineering at
Santa Clara University, 2019

Santa Clara, California

Acknowledgments

I would like to thank Dr. Behnam Dezfouli and Dr. Yuhong Liu for their guidance, help, and support throughout my masters program at SCU. You have truly instilled in me life long skills that will enable me to be successful in the future.

I would also like to thank Alejandro Hernandez and Ramzi Nofal for their collaborative efforts and support all throughout my time at SCU. Working with you both was a great learning experience, and I am proud of what we were able to accomplish together the past couple of years.

I would also like to thank my mother Hemalatha and my father Kamaraj for their unconditional support throughout my time at SCU. None of this is possible without you.

Last but not least, thank you God always.

Overhead Management Strategies for Internet of Things Devices

Kavin Kamaraj

Department of Computer Engineering
Santa Clara University
Santa Clara, California
2019

ABSTRACT

Overhead (time and energy) management is paramount for IoT edge devices considering their typically resource-constrained nature. In this thesis we present two contributions for lowering resource consumption of IoT devices. The first contribution is minimizing the overhead of the Transport Layer Security (TLS) authentication protocol in the context of IoT networks by selecting a lightweight cipher suite configuration. TLS is the de facto authentication protocol for secure communication in Internet of Things (IoT) applications. However, the processing and energy demands of this protocol are the two essential parameters that must be taken into account with respect to the resource-constraint nature of IoT devices. For the first contribution, we study these parameters using a testbed in which an IoT board (Cypress CYW43907) communicates with a server over an 802.11 wireless link. Although TLS supports a wide-array of cipher suites, in this paper we focus on DHE_RSA, ECDHE_RSA, and ECDHE_ECDSA, which are among the most popular ciphers used due to their robustness. Our studies show that ciphers using Elliptic Curve Diffie Hellman (ECDHE) key exchange are considerably more efficient than ciphers using Diffie Hellman (DHE). Furthermore, ECDSA signature verification consumes more time and energy than RSA signature verification for ECDHE key exchange. This study helps IoT designers choose an appropriate TLS cipher suite based on application demands, computational capabilities, and energy resources available.

The second contribution of this thesis is deploying supervised machine learning anomaly detection algorithms on an IoT edge device to reduce data transmission overhead and cloud storage requirements. With continuous monitoring and sensing, millions of Internet of Things sensors all over the world generate tremendous amounts of data every minute. As a result, recent studies start to raise the question as whether to send all the sensing data directly to the cloud (i.e., direct transmission), or to preprocess such data at the network edge and only send necessary data to the cloud (i.e., preprocessing at the edge). Anomaly detection is particularly useful as an edge mining technique to reduce the transmission

overhead in such a context when the frequently monitored activities contain only a sparse set of anomalies. This paper analyzes the potential overhead-savings of machine learning based anomaly detection models on the edge in three different IoT scenarios. Our experimental results prove that by choosing the appropriate anomaly detection models, we are able to effectively reduce the total amount of transmission energy as well as minimize required cloud storage. We prove that Random Forest, Multilayer Perceptron, and Discriminant Analysis models can viably save time and energy on the edge device during data transmission. K-Nearest Neighbors, although reliable in terms of prediction accuracy, demands exorbitant overhead and results in net time and energy loss on the edge device. In addition to presenting our model results for the different IoT scenarios, we provide guidelines for potential model selections through analysis of involved tradeoffs such as training overhead, prediction overhead, and classification accuracy.

Contents

1 Introduction 1

1.1 Transport Layer Security Protocol Overhead Management 2

1.2 Data Transmission Overhead Reduction using Anomaly Detection Edge Mining 4

2 Related Work 6

2.1 Energy and Processing Demand Analysis of TLS Protocol in Internet of Things Applications 6

2.2 Edge Mining on IoT Devices using Anomaly Detection 8

3 Energy and Processing Demand Analysis of TLS Protocol in Internet of Things Applications 10

3.1 Transport Layer Security (TLS) 11

3.2 Experimental Procedure 15

4 Edge Mining on IoT Devices using Anomaly Detection 23

4.1 Anomaly Detection Models and Datasets Used 23

4.1.1 Random Forests 24

4.1.2 Multilayer Perceptron 25

4.1.3 K-Nearest Neighbors 26

4.1.4 Discriminant Analysis 26

4.1.5 Datasets Used 27

4.2 Experimentation and Analysis 28

4.2.1 Methodology 29

4.2.2 Results and Analysis 32

5 Conclusion 41

List of Figures

3.1 The components and interconnection of the testbed used. 17

3.2 The flow diagram of measurement methodology. 18

3.3 Processing time of various steps. This figure confirms that client verification of an ECDSA certificate requires about 5200% longer processing compared to RSA. In addition, signature verification using ECDSA is around 2000% longer than RSA. Furthermore, key exchange using DHE shows 300% longer processing time than using ECDHE. In general, the processing overhead of C1 and C3 are 300% and 150% higher than C2, respectively. 21

3.4 This figure confirms that client verification of an ECDSA certificate requires about 3900% more energy compared to RSA. In addition, signature verification using ECDSA requires around 1700% more energy verus RSA. Furthermore, key exchange using DHE shows 300% more energy compared with ECDHE. In general, the processing overhead of C1 and C3 are 300% and 150% higher than C2, respectively. 22

4.1 The testbed used for our experiment inclusive of three RPi3s and the EMPIOT energy measurement platform. The RPi3 hosting EMPIOT captures energy measurements of the edge RPi3 transmitting data to the cloud RPi3. 30

4.2 All models are viable for the datasets chosen. All models perform best for the KDDCup dataset, containing the lowest number of features, and post precision and recall values of 98.9% and 97.96% respectively. RF and QDA offer the best prediction accuracy for Gestures while RF and KNN offer the best prediction accuracy for Digits. 31

4.3 MLP features the most expensive training phase with RF, QDA, LDA, and KNN generally following in this order as shown in (a) and (b). MLP costs approximately 500% more time and 500% more energy than the next best performing model across all datasets. . . 32

4.4 LDA offers the most cost-effective prediction phase for both Digits as well as KDDCup datasets as shown in (a) and (b). MLP offers the most cost-effective prediction phase for the 64 feature Gestures dataset. KNN costs nearly 600% more energy and 650% more time than the next best performing model across all datasets. 33

4.5 RF, MLP, and LDA all demonstrate overhead savings when applied to data before transmission; most notably, RF applied to the Digits dataset saves 45.119 J and 16.37 seconds per 2000 image buffer. KNN is a poor choice for real-time anomaly detection scenarios given that it causes net time and energy loss across all datasets. . . 36

4.6 From (a) we observe that the models reduce approximately 90% of the observations sent from Digits' buffers, 76% of the observations sent from Gestures' buffers, and 99.5% of the observations sent from KDDCup's buffers. The resulting saved cloud storage shown in (b) is substantial, especially for Digits dataset where we saved nearly 12 MB of data sent per 2000 image buffer. 37

List of Tables

4.1 Mathematical notations and symbols. 24
4.2 Training and prediction time complexity of anomaly detection models. 24
4.3 Overview of the datasets. 25
4.4 Training and prediction data dimensions. 27
4.5 Sample use cases for different models. 40

Chapter 1

Introduction

The Internet of Things (IoT) refers to a network of billions of interconnected devices that have the ability to communicate and exchange data over the Internet. Such IoT devices range from sensors, smart phones, computers, vehicles, building appliances, and health devices. The extension of Internet connectivity to physical devices and everyday objects has substantially increased worldwide real-time data collection and transmission. As of 2019, there are approximately 9 billion IoT devices across the world and by 2020 this number will surge to over 25 billion [1]. As IoT devices and networks grow in number, the tremendous amount of data transfer leads to concerns over both device authentication overhead as well as data transmission overhead. In the first section, we detail our motivation and contribution with regard to managing TLS protocol overhead during client-host authentication in IoT networks. In the second section, we detail our motivation and contribution with regard to deploying machine learning based anomaly detection algorithms for lowering data transmission overhead as well as cloud storage requirements.

1.1 Transport Layer Security Protocol Overhead Management

Transport Layer Security (TLS) protocol is designed to provide encryption, authentication and data integrity for communicating information over the Internet. The TLS protocol is composed of two main phases [2]: The first phase is the *handshake*, which allows a client and a server to agree on TLS version and a set of cryptographic algorithms (ciphers). This enables the two communicating parties to ultimately establish a shared session key. The client and server also have the option to authenticate each other using certificates provided by a trusted third-party [3]. The second phase of the protocol is the *record layer*, in which the shared session key is used to send encrypted messages between two nodes. As of today, TLS is the most widely used protocol for securing communication between IoT devices [4, 5]. This protocol is considered to be highly effective because of its use of public key exchange and symmetric key (session key) encryption. Unfortunately, these advantages come at the cost of high computational and energy demands [6, 7].

Reducing the overhead of TLS protocol is critical for multiple reasons [8, 7, 9]. First, as IoT devices are increasing by the billions, a few milli-joules saved from each run of TLS can save millions of dollars in the big picture. Second, all networks are vulnerable to interference, timeouts and loss of connectivity; therefore, the overhead of the TLS protocol should not deter users from re-establishing lost connections between nodes. Third, understanding the overhead of TLS enables IoT designers to configure this protocol based on application requirements. For instance, a user may intend to establish a very short-lived connection for brief information transfer between two nodes. In this case, the user may decide to use a lighter cipher (possibly compromising some aspects of security) in the interest of

quickly transmitting the data. Furthermore, the use of a light cipher suite might be justified based on the limited resources of IoT devices and to minimize the impact of security on energy consumption. Although there are several studies on the implementation and adoption of TLS on general purpose processing platforms and mobile devices [10, 4, 9, 11, 12, 13], unfortunately, analyzing the computation and energy cost of TLS on resource-constrained IoT devices has not received enough attention from the research community.

In this study, we analyze the performance of TLS protocol using three popular and robust ciphers:

- C1: DHE_RSA_WITH_AES_256_CBC_SHA256
- C2: ECDHE_RSA_WITH_AES_256_CBC_SHA384
- C3: ECDHE_ECDSA_WITH_AES_256_CBC_SHA384

To this end, an IoT device communicates with a server through establishing TLS connection over an 802.11 link. We have modified the TLS code of the IoT device to measure the processing time and energy consumption of each step of the TLS handshake using a logic analyzer and an energy measurement tool. Our results show that ECDHE ciphers are considerably more efficient than DHE ciphers. We also conclude that an ECDHE cipher has lower resource consumption when using RSA encryption as opposed to ECDSA encryption. This is attributed to the fact that ECDSA certificate verification is a longer process. The results reported in this work help IoT designers choose TLS cipher suite based on application demands as well as the computational capabilities and energy resources of IoT devices.

1.2 Data Transmission Overhead Reduction using Anomaly Detection Edge Mining

As IoT devices grow in number, the tremendous amount of sensing data collected has raised great challenges for data transmission overhead (time and energy) and cloud storage. Applications of cost-cutting edge mining techniques to reduce packet transmission and remote storage requirements are rapidly growing throughout IoT networks [14], [15]. One of the most basic edge mining techniques is random sampling to reduce the number of observations sent to the cloud. More sophisticated methods, such as anomaly detection, isolate and transmit only the contextually relevant observations [16]. The guiding principle behind anomaly detection is that only unexpected behavior needs to be notified to the centralized cloud. Contemporary works specify different anomaly detection methods ranging from basic thresholding to machine learning algorithms [17], [18]. However, as the resources available at each IoT edge device can be rather limited in terms of power, memory, connectivity, bandwidth, and computation, it is critical to choose appropriate anomaly detection algorithms that can not only effectively identify abnormal behaviors but also consume limited resources at the edge devices.

In this work we present supervised machine learning based anomaly detection that can substantially reduce energy consumption and transmission overhead on the edge and storage requirements on the cloud. We conduct our experiments on a custom testbed inclusive of an edge device, the cloud, and an energy measurement platform. Four classes of machine learning algorithms, Random Forest Classifier (RF), Multilayer Perceptron Classifier (MLP), K-Nearest Neighbor Classifier (KNN), and Discriminant Analysis Classifier (DA) are benchmarked on a Raspberry Pi 3 (RPi3) edge device for different anomaly detection scenarios. We use

both Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA) variants of DA for this study. Our work makes several novel contributions to anomaly detection used in the context of edge mining. First, we benchmark training and prediction phase overhead (i.e., time and energy consumption) at the edge device for each model on multiple datasets. Second, real data rather than synthetic data have been adopted for experiments. Third, we demonstrate tangible transmission cost-savings using machine learning based anomaly detection for multiple datasets.

We demonstrate significant overhead-savings achieved using MLP, RF, and DA anomaly detection model classes during the data processing and transmission period. Furthermore, we identify the best anomaly detection model for different application scenarios. For example, MLP features one of the shortest prediction phases, which is recommended for time-sensitive scenarios. However, it also has one of the most costly training phases which is undesirable if there are resource constraints for training. By analyzing these tradeoffs, we better understand why the models achieve different levels of performance in different scenarios.

Chapter 2
Related Work

This chapter is organized as follows. In the first section, we present related work for our first contribution pertaining to TLS overhead management in IoT scenarios. These works generally present varying benchmarking procedures for the TLS protocol (i.e., different hardware platforms and/or different protocol phases measured). In the second section, we present related work for our second contribution pertaining to anomaly detection edge mining. These works focus primarily on different applications of anomaly detection in IoT scenarios. Generally, each work uses a different anomaly detection algorithm(s), different dataset(s), and/or different testbed architecture(s).

2.1 Energy and Processing Demand Analysis of TLS Protocol in Internet of Things Applications

In [19] and [9], the overhead of the TLS protocol is measured for hand-held mobile devices. The authors in [19] examine the overhead of TLS for 1 MB file transfer on a handheld HP (Compaq) iPAQ H3630 device with a 206 MHz StrongARM processor and 32MB RAM (16MB ROM) over a Wi-Fi access point communication link. The authors use Microsoft cyrptographic library's implementation of TLS

opposed to our mbedTLS implementation. This study benchmarks the protocol using 1024 and 2048 bit RSA keys and shows time consumption of signature verification. However, it does not present any quantitative energy consumption values nor considers individual steps of the TLS handshake. Another study examines the overhead of TLS during a data transfer scenario between a Symbian Nokia 95 and several popular web services over both WLAN and 3G [9]. Miranda et. al [9] present energy consumption results of this protocol with different data buffer size configurations and indicate that the handshake phase is generally more expensive over 3G. Furthermore, they conduct their analysis using 4 different ciphers which is greater than most other related works.

Potlapally et al. [10] conduct a very comprehensive study on the overhead of cryptographic algorithms both in and out of the context of the TLS protocol. Potlapally's study benchmarks the the energy cost of digital signature algorithms (RSA, DSA, ECDSA) and key exchange algorithms (DH, ECDH) with various key sizes. Furthermore, the study concludes that when there is no client authentication, an RSA handshake is more efficient than an ECC handshake and vice-versa. Akshay et al. [20] benchmark the TLS protocol using two 2012 Nexus 7 Android tablets with an Nvidia Tegra 3 SoC, 1 GB of RAM, and 4326 mAh batteries. The study presents charts showing how the TLS depletes battery life by nearly 15% for just a 1 MB download over HTTPS. In this way, the work demonstrates a clear connection to a resource constrained IoT scenario.

Authors Koschuch et al. [21] present the overhead of different operations within TLS using processing cycle count rather than energy metrics. They compare the cycle counts for digital signature algorithm, RSA, and key exchange algorithm, ECDSA, between a one-way authentication context and two-way authentication context. They conclude that Elliptic Curve Cryptography (ECC)

is considerably more efficient than its RSA/DSA counterparts for resource constrained devices because of smaller memory requirements.

2.2 Edge Mining on IoT Devices using Anomaly Detection

Anomaly detection is one of the the most popular edge mining techniques explored in IoT scenarios [22], [23]. In [24], [25], [26], and [27] anomaly detection is used as a method to implement an Intrusion Detection System (IDS) for Wireless Sensor Networks (WSN). Sommer et al. [26] propose the use of LDA to reduce the dimensionality of network intrusion datasets and applies both Naive Bayes and KNN algorithms for anomaly classification. In [27], the authors benchmark the performance of anomaly detection (i.e., false positive rates) using an unsupervised outlier detection technique based on the RF algorithm. Furthermore, [28] demonstrates the effectiveness of autoencoders for an unsupervised IDS and proposes a novel splitting and learning mechanism to lower false positive detection. Although these works explore novel applications of anomaly detection on the IoT edge, they do not focus on resource constrained scenarios and therefore do not delve into the time and energy consumption of these methods.

In [29], [30], and [31] anomaly detection is investigated in healthcare applications. Arijit et. al [30] propose cardiac anomaly detection with low false negative counts and stress the importance of capturing outliers in healthcare applications. Similar to the IDS studies, this work focuses extensively on anomaly detection implementation but does not consider the factor of resource consumption. Several works also tackle anomaly detection in IoT applications outside of IDS and

healthcare. In [32] non-machine learning anomaly detection algorithms are proposed for a set of heterogenous sensors in an IoT WSN. In [33] an autoencoder neural network is used for determining anomaly readings from a testbed of eight temperature and humidity sensors. These works also, however, do not consider overhead nor consider pros and cons using different anomaly detection methods.

Few works consider resource constrained IoT scenarios. For example, Sedjelmaci et. al [34] test a reputation model based on game theory to predict attack signatures on a resource constrained IoT device. In addition to this, the work of Lyu et. al [35] is one of the few works which presents cost-savings potential of anomaly detection using both real and synthetic datasets on a resource constrained IoT platform. However, this work only evaluates unsupervised hyper-ellipsoidal clustering and does not include supervised machine learning algorithms. Unsupervised methods are useful in the absence of ground-truth information but are generally not as accurate as supervised methods for classification tasks.

Chapter 3

Energy and Processing Demand Analysis of TLS Protocol in Internet of Things Applications

Transport Layer Security (TLS) protocol provides encryption, authentication and data integrity between two communicating parties over the Internet. The TLS protocol consists of two phases. The handshake phase allows a client and a server to agree on TLS version and a set of cryptographic algorithms (ciphers). This enables the two communicating parties to ultimately establish a shared session key. The record phase follows in which the shared session key is used to send encrypted messages between two nodes. In this study we focus on choosing a lightweight cipher suite for the handshake phase to minimize the overhead of the TLS protocol as a whole. Reducing the overhead of TLS protocol is paramount for many IoT devices given that they may be resource constrained and battery-powered. Ironically, overhead management of the TLS protocol has not been very thoroughly examined given that TLS is the de facto authentication protocol for IoT devices. In this study, we analyze the performance of TLS protocol using C1, C2, and C3. We hope that these results help IoT specialists who are seeking to speed up device authentication within their IoT networks.

3.1 Transport Layer Security (TLS)

The TLS protocol specifies a well-defined handshake that consists of 15 steps. Before delving into the details of each step, we provide a high level overview of the protocol first. In the beginning of the handshake phase, hello messages are exchanged between client and server to agree on an encryption algorithm and to exchange random values. Then, depending upon the cipher chosen, the client and server exchange the corresponding cryptographic parameters to agree on a pre-master secret. Afterwards, the client authenticates the server using a pre-installed certificate from a trusted third-party; the server may optionally authenticate the client as well. Finally, a master secret is generated from the pre-master secret and the random values. The session (symmetric) key is derived from the master secret and is used to encrypt subsequent data exchange between the client and server in the record layer phase [36]. We present each step in detail as follows:

1. `client_hello`. The client sends this message to the server to establish an initial contact. This message, in particular, contains: (i) the TLS version that the client intends to use, (ii) a list of cipher suites supported by the client, (iii) a random number, (iv) compression method, and (v) a session id. The server then checks its compatibility with the specified version of TLS and the list of ciphers specified in the message.

2. `server_hello`. If the server's TLS version and supported cipher suites are compatible with that of the client, this message is sent by the server in response to the client's hello message. This marks the completion of a successful negotiation.

3. `server_certificate`. The server sends to the client a certificate contain-

ing its public key. The client can authenticate the server by comparing the certificate it receives from the server to its pre-installed certificate. An authentication failure is raised in the case that the server's certificate does not match any of the certificates pre-installed in the client.

4. `server_key_exchange`. In this message, the server exchanges Diffie-Hellman cryptographic parameters (modulus, generator, newly-generated public key) with the client so that it can convey a pre-master secret. The resource consumption of this step can be attributed to the client verifying the signature of these parameters.

5. `certificate_request`. This message is sent by the server to request a certificate from the client in the case that the user has configured the server to require client authentication. In our study, the server sends a request to the client to adhere to the protocol, however, the server does not functionally verify the client.

6. `server_hello_done`. This message is sent by the server to indicate the end of the hello message exchange sequence. While this message is prepared and sent, the client is verifying the validity of server's certificate.

7. `client_certificate`. This message is sent by the client if the server has requested the client to send its certificate. Even if the client does not have a suitable certificate, it must send a certificate message that does not contain any certificate. In this paper, since the client sends a blank certificate message, this step is one of the fastest to execute.

8. `client_key_exchange`. In this step, the client sends its Diffie-Hellman public key to the server.

9. `certificate_verify`. This message is sent by the server to indicate that it has successfully verified the client's certificate. A digitally signed structure of all the handshake messages sent or received is included in this message.

10. `client_change_cipher_spec`. This message is sent by client to inform the server that subsequent data transfer will be protected by the newly negotiated ciphers.

11. `client_finished`. This message verifies that the client has successfully completed the authentication processes and key exchange.

12. `server_change_cipher_spec`. The server sends this message to notify the client that subsequent data transfer will be protected by the newly negotiated cipher and keys. The client then reacts by setting the session key parameters accordingly.

13. `server_finished`. This message verifies that the server has successfully completed the authentication processes and the key exchange.

14. `flush_buffers`. In this step, the temporary data that is a byproduct of the handshake process is deleted on both the client and server nodes.

15. `handshake_over`. This message marks the completion of the handshake phase and the start of the record layer phase.

The Internet Assigned Numbers Authority (IANA) has named over 300 cipher suites compatible with TLS in early 2016. BSI, a federal IT security agency in Germany, recommends using only 16 of those ciphers for TLS [3]. Based on recommendation, in this paper, we study the performance of the TLS protocol using three of these ciphers. All ciphers share the following naming convention:

KeyExchange, CryptographicAlgorithm, "WITH", SessionKey and SignatureType (each section separated by an underscore), as follows:

- `C1: DHE_RSA_WITH_AES_256_CBC_SHA256`. This cipher uses DHE key exchange and RSA encryption with an AES256 session key and SHA256 hash function. SHA256 is the highest bit hash function compatible with DHE_RSA for mbed TLS [37].
- `C2: ECDHE_RSA_WITH_AES_256_CBC_SHA384`. This cipher uses ECDHE key exchange and RSA encryption with an AES256 session key and SHA384 hash function.
- `C3: ECDHE_ECDSA_WITH_AES_256_CBC_SHA384`. This cipher uses ECDHE key exchange and ECDSA encryption with an AES256 session key and SHA384 hash function.

Note that all three ciphers use the same session key, AES_256_CBC. AES is a symmetric key block cipher algorithm that is used to protect information [38]. It is proven to be a highly reliable cipher and consists of three block ciphers: AES-128, AES-192 and AES-256. Using CBC (Cipher Block Chaining) with AES ensures that each block decryption is contingent upon the previous one. This measure improves the security of AES block ciphers larger than 256 bits. In our study, AES-256 is used to encrypt and decrypt all information transfer in the record layer phase of the protocol. Hash functions SHA256 and SHA384 are used to create digital signatures of the data. As a one-way function, SHA easily authenticates messages while securing their content. There are other variations for SHA (i.e., SHA224 and SHA512), but they are not available for use in mbed TLS because they do not offer added security or efficiency.

We have kept the session key and hash function consistent among all the ciphers so that differences in resource consumption can be conclusively attributed to either the key exchange method or choice of encryption algorithm. The two key exchange methods used in this list of ciphers are DHE and ECDHE. DHE uses modular arithmetic to compute the shared secret. In contrast, ECDHE uses elliptic curves to generate the secret, thereby, ECDHE is considerably more efficient than DHE. It is clear that the key exchange method selected significantly impacts the resource consumption of Step 8 of the handshake, which takes care of pre-master secret generation. Aside from the key exchange method, our ciphers make use of two public key encryption schemes, i.e., RSA and ECDSA. ECDSA signatures tend to be much smaller than RSA signatures given the same method of exchange. However, RSA signature verification tends to be much faster than ECDSA verification [10].

3.2 Experimental Procedure

We run the TLS protocol using a Cypress CYW43907 IoT device as a client and a Raspberry Pi as a server. CYW43907 is an embedded wireless system-on-a-chip (SoC) that features an ARM Cortex-R4 32-bit RISC processor [39, 40]. The Cypress IoT device supports the WICED Development Platform which offers SDKs for system development. The TLS code used on the client and server is derived from the mbed TLS implementation [37]. Note that we measure TLS resource consumption on the client side.

For measuring the energy consumption of the TLS protocol we use EMPIOT (Energy Measurement Platform for IoT Devices) [41], that is connected to the client. The EMPIOT platform is composed of a shield installed on top of a Rasp-

berry Pi. We measure processing time using a logic analyzer. Figure 3.1 shows our experimental setup for time and power measurement. The client connects with the access point through an 802.11 link. When performing TLS handshaking, message exchange intervals can be relatively large and are heavily influenced by network conditions. In order to ignore the overhead of message exchange, we have modified the mbed TLS code to toggle pins on the client right before and after the processing duration of each step. A logic analyzer that is connected to these pins is used to measure the computation duration of each step.

The energy measurement platform, EMPIOT, is connected to the client as follows. First, the EMPIOT powers the client via a USB connection and measures the bus voltage and current drawn by the client. Second, the tool detects the start and finish of each step of the handshake by connecting to four pins toggled by the client. In other words, the EMPIOT software performs energy measurement based on the triggers received from the client. We measure the computation and energy demand of TLS's step 3, 4, 8, and 9, which are all longer than 1ms. We have observed that the processing time and energy consumption of the remaining steps are negligible and therefore these steps will not factor into our study.

The TLS handshake may occasionally fail due to network timeout caused by packet loss over the wireless link. To ensure the accuracy of our measurements is not affected by these error cases, we configure the client to notify EMPIOT if the handshake is failed. In this way, we can discard measurements collected from incomplete runs. The detail of the power measurement process is shown in Figure 3.2. Our data collection procedure is as follows: We perform 1000 iterations of successful TLS handshakes for each cipher, i.e., C1, C2, C3. We measure the computation time and energy of the steps mentioned earlier for each iteration.

Figure 3.3 shows the processing time of each TLS step for the three cipher

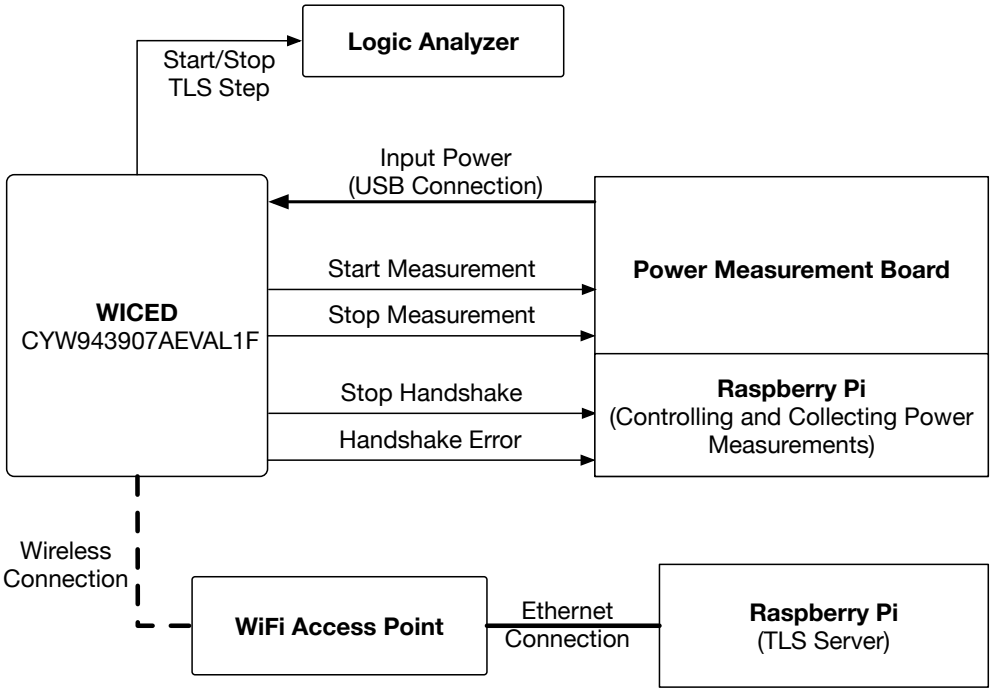


Figure 3.1: The components and interconnection of the testbed used.

suites. Figure 3.4 shows the power consumption in Joules per TLS step for each cipher. Each marker is the median of 1000 iterations, and error bars show the higher and lower quartiles. The value on top of each error bar is the median of the result.

Analyzing Step 3. In Step 3 the server sends to the client a certificate with its public key and the client verifies this certificate. The results obtained from this step indicate that C3 has considerably higher values for both processing time and energy consumption compared to both C1 and C2. Specifically, client verification of an ECDSA certificate is heavier than client verification of an RSA certificate. This is a trend that has been generally identified and our results corroborate this trend.

Analyzing Step 4. Similar to Step 3, C3 has considerably higher values for resource consumption than both C1 and C2. This indicates that signature

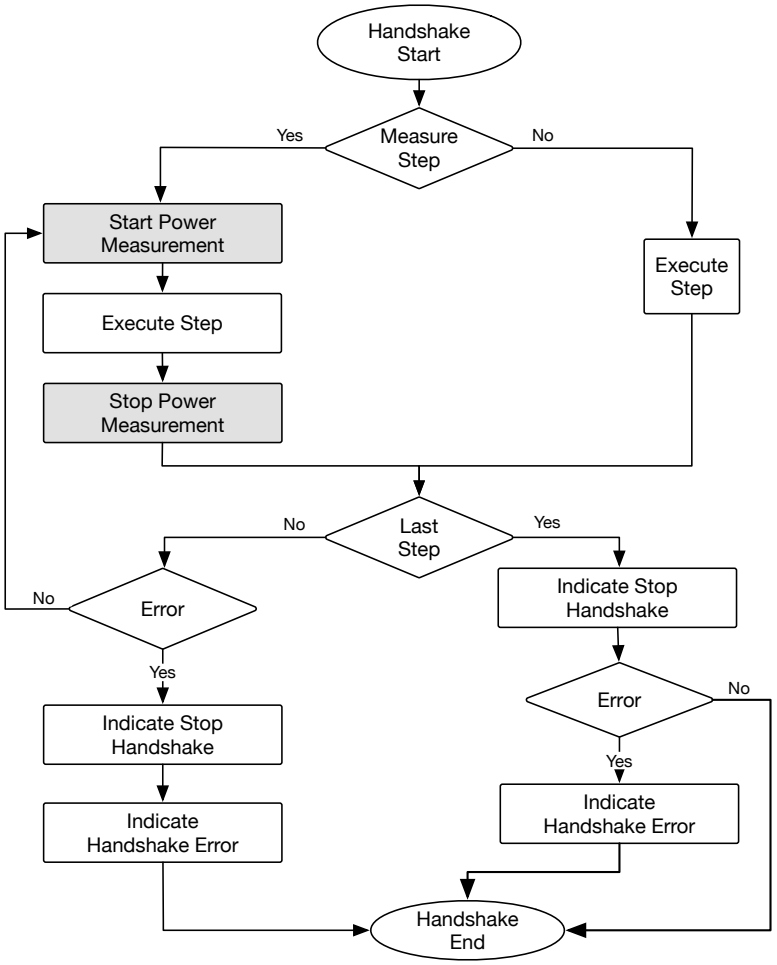


Figure 3.2: The flow diagram of measurement methodology.

verification is more demanding when using ECDHE_ECDSA compared to the other ciphers.

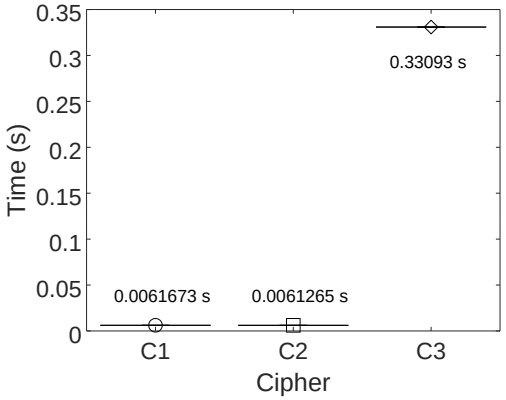
Analyzing Step 8. C1 has considerably higher values for both processing time and energy consumption compared to both C2 and C3. We know that DHE key exchange is generally much heavier than ECDHE and this trend is quantitatively proven. Compared to all other steps, Step 8 is clearly the heaviest, which confirms that key generation factors into most of the handshake's computational cost.

Analyzing Step 9. Note that our testbed does not functionally perform client authentication. In Step 7, the client sends a blank certificate that our server is configured to always accept. The client processes the server's default verification very quickly; the results indicate that C1 has the shortest processing time, while C2 and C3 also show relatively low values.

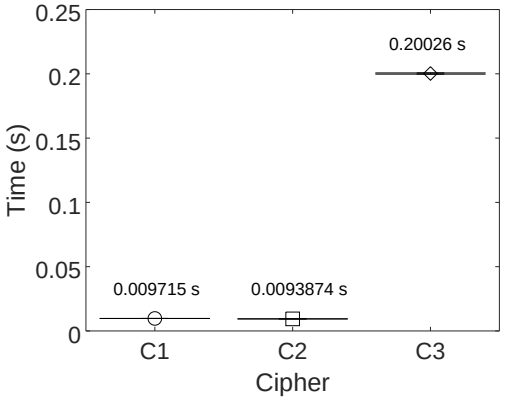
With the exception of Step 4 for C1, the upper and lower limit of all the error bars are very close to the median value. This indicates that the processing time and energy consumption measurements for each cipher are consistent throughout all the iterations. Because there are no major outlier values, the median value accurately represents the energy consumption for each step. The cumulative time and energy measurements show that C1 is the heaviest cipher, and C3 is heavier than C2.

We conclude that the order of the ciphers from least to greatest energy efficiency is the following: C1, C3, C2. Specifically, C1 consumes 55% more energy than C3, and C3 consumes 150% more than C2. Clearly, the most efficient cipher based on the results is C2 (i.e., ECDHE using RSA). However, when choosing between RSA and ECDSA encryption, there are two considerations: RSA generally

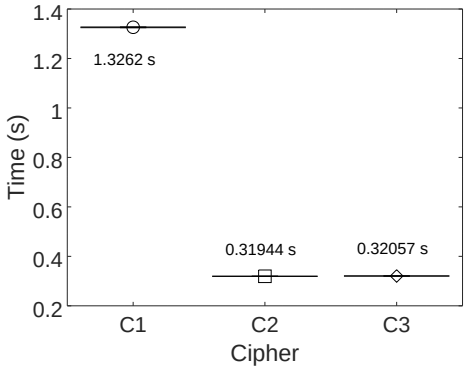
has heavier signatures than ECDSA, however, ECDSA requires more computation for certificate verification. Our study proves that the energy consumption of ECDSA certificate verification considerably outweighs RSA's energy consumption for heavier signature generation. We plan on expanding this study by extending our pool of ciphers, ranking the ciphers based on level of security, and including network overhead, using various types of IoT devices.



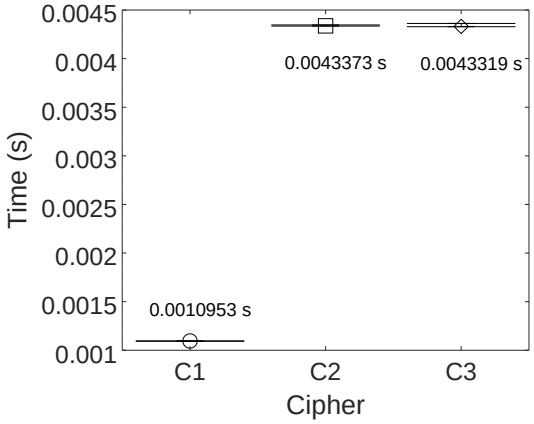
(a) Step 3



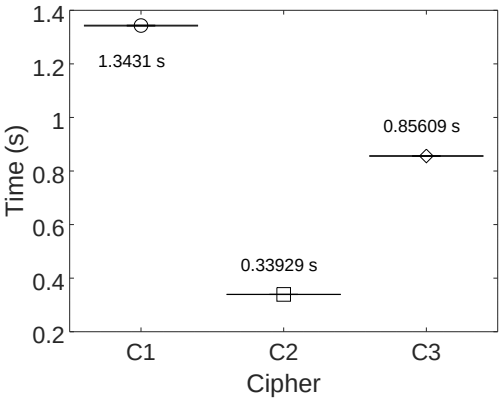
(b) Step 4



(c) Step 8

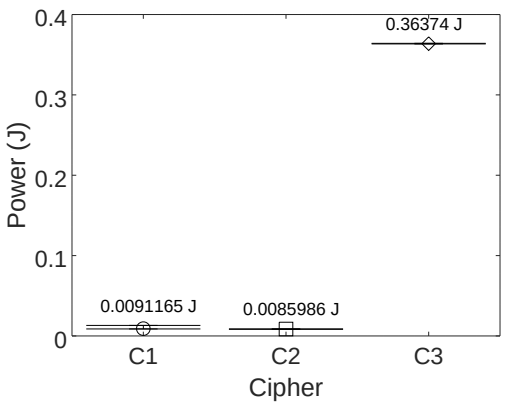


(d) Step 9

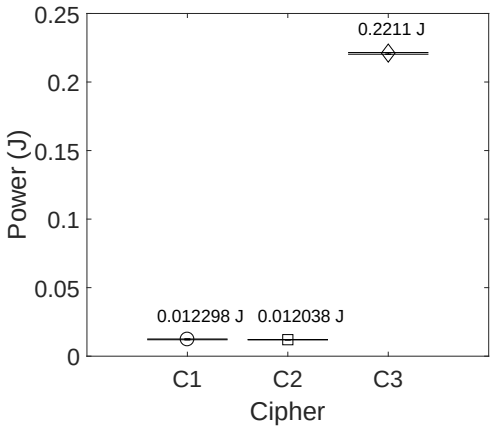


(e) Total

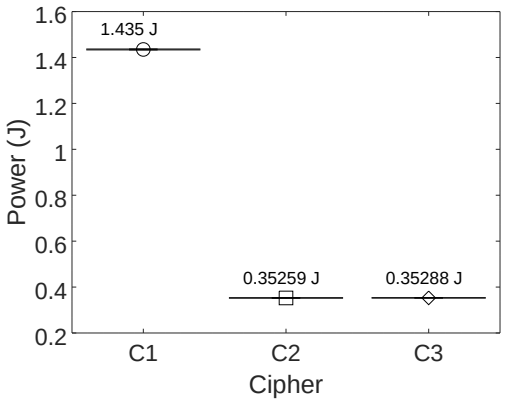
Figure 3.3: Processing time of various steps. This figure confirms that client verification of an ECDSA certificate requires about 5200% longer processing compared to RSA. In addition, signature verification using ECDSA is around 2000% longer than RSA. Furthermore, key exchange using DHE shows 300% longer processing time than using ECDHE. In general, the processing overhead of C1 and C3 are 300% and 150% higher than C2, respectively.



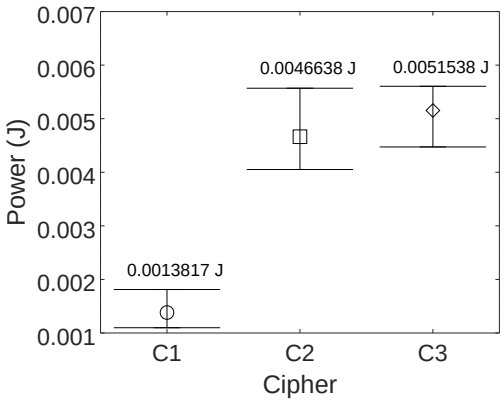
(a) Step 3



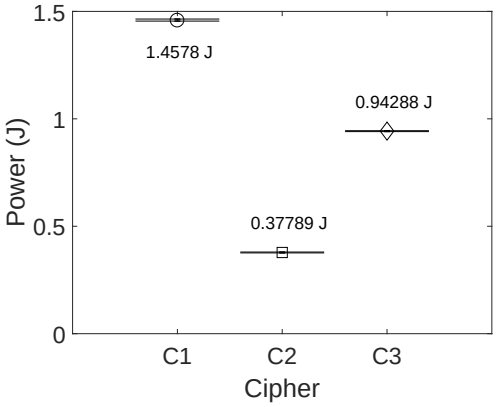
(b) Step 4



(c) Step 8



(d) Step 9



(e) Total

Figure 3.4: This figure confirms that client verification of an ECDSA certificate requires about 3900% more energy compared to RSA. In addition, signature verification using ECDSA requires around 1700% more energy verus RSA. Furthermore, key exchange using DHE shows 300% more energy compared with ECDHE. In general, the processing overhead of C1 and C3 are 300% and 150% higher than C2, respectively.

Chapter 4

Edge Mining on IoT Devices using Anomaly Detection

In this chapter, we investigate the use of supervised machine learning based anomaly detection to reduce data transmission overhead on the edge and storage requirements on the cloud. We benchmark four classical machine learning algorithm classifiers, RF, MLP, KNN, and DA (i.e. LDA/QDA) on a RPi3 edge device for three different IoT scenarios (i.e., datasets). We show tangible data transmission overhead savings when using MLP, RF, LDA, and QDA models across all scenarios. In addition to this we use real data as opposed to synthetic data to corroborate the predictive accuracy of the models. Furthermore, we analyze the tradeoffs involved in model selection for different use cases and provide explanations for why the models achieve different levels of performance in different scenarios.

4.1 Anomaly Detection Models and Datasets Used

In this section below, we provide an analysis of each machine learning method benchmarked in our study and discuss their general use cases, time complexity, and performance tradeoffs involved in their training and prediction phases. We have chosen RF, MLP, KNN, LDA, and QDA (i.e., two variants of DA) for this study because they are among the most popular machine learning classification

Table 4.1: Mathematical notations and symbols.

d	Number of features
h	Number of neurons per hidden layer
i	Number of iterations
k	Number of hidden layers
K	Number of neighbors
M	Number of testing samples
N	Number of training samples
o	Number of output neurons
P	Distance-metric complexity
T	Number of trees

Classifier	**Training Complexity**	**Testing Complexity**
RF	$O(N * \sqrt{(d * T)})$	$O(Td)$
MLP	$O(Ndh^k oi)$	$O(Nhk)$
KNN	$O(1)$	$O(NPMlogK)$
LDA	if $N > d$: $O(Nd^2)$ if $d > N$: $O(d^3)$	$O(M)$
QDA	$O(d^4)$	$O(M \log M)$

Table 4.2: Training and prediction time complexity of anomaly detection models.

methods [42]. Furthermore, all chosen models have distinct underlying mathematical mechanisms which account for varying model performance across different scenarios. Table 4.1 denotes all mathematical notations. Table 4.2 provides each model's *theoretical* training and testing time complexity.

4.1.1 Random Forests

RF models are applicable to a wide range of classification problems [42]. In a random forest, each node is split using a subset of features randomly chosen at that node. This strategy is robust against overfitting and enables RF to perform better than many other classifiers, including discriminant analysis, support vector machines, and neural networks [42]. The only major downside of RF is that a large number of trees can slow down the algorithm for real-time predictions. Suppose there are T randomized trees, d features, and N training samples, RF's training

Dataset	Dimensionality	% of Anomalies	Anomaly Type	IoT Scenario
KDDCup	567497x3	0.35%	Attack Detected on Network	Intrusion Detection
Digits	30000x784	11%	Digit 0	Anomalous Image Detection
Gestures	11674x64	25.3%	A Hand Contraction of Patient	Health Monitoring

Table 4.3: Overview of the datasets.

time complexity is $O(N^2\sqrt{dT})$ [43]. The computational complexity at test time for a RF with T trees and d features is $O(Td)$ [43].

4.1.2 Multilayer Perceptron

MLP is the most known and frequently used type of neural network using the backpropagation training algorithm. In recent years, neural networks have been extensively used for pattern recognition and optimization. MLP models contain three types of layers: input layer, output layer, and hidden layer. Each node in the input layer, from top to bottom, passes an input data point to each neuron in the first hidden layer. Then, each hidden layer neuron multiplies each value with a weight vector and computes the sum of the multiplied values. Subsequently each hidden layer neuron applies its activation function to this sum, and sends the resulting value to the next layer and eventually to the output layer. Suppose there are N training samples, d features, k hidden layers each containing h neurons, o output neurons, and i iterations. The time complexity of MLP training is $O(Ndh^koi)$ [44]. It is advisable to start with a small number of hidden layers and nodes given the high time complexity of MLP's backpropagation algorithm and time-consuming grid search procedure [44]. One of the most notable advantages of MLP, is its low prediction complexity, $O(Nhk)$, which makes it suitable for time-sensitive anomaly detection tasks.

4.1.3 K-Nearest Neighbors

K-Nearest Neighbor Classifier (KNN) is a relatively simple learning algorithm. It is very commonly used in text mining, agricultural predictions, and stock market forecasting [45]. Because KNN does not make any assumptions about the underlying data distribution, it is particularly suitable for applications with little or no prior knowledge about the distribution of the dataset. Each data point injected into the model is classified as part of the class containing the majority of its K nearest neighbors. We may find the nearest neighbors using various metrics such as Euclidian distance, k-d tree, ball tree, or other user defined metrics [46].

KNNs are generally reputed for high prediction accuracy with respect to precision and recall. Furthermore, the training phase of KNN classifiers is very efficient. The training time complexity of KNN is only $O(1)$. Nevertheless, there are several drawbacks of KNNs. First, the model has high space complexity as it stores all training data instances. Second, finding the most optimal value for K is not trivial [47]. KNN's most significant drawback, however, is its exorbitant prediction phase overhead, which is $O(NPMlogK)$ with N training samples, M test samples, K neighbors, and P distance metric time complexity. The high overhead at the prediction phase may make it less suitable to be carried on by resource constrained IoT edge devices.

4.1.4 Discriminant Analysis

In this work, we use both LDA and QDA. LDA first projects a dataset onto lower-dimensional space to prevent overfitting and to generate linear class-separability. QDA also performs dimensionality reduction but generates a quadratic line to fit the training data. [48]. Despite its potential for non-linear data patterns, the

Dataset	Train Data Size	Prediction Buffer Size
KDDCup	10000	50000
Digits	3000	2000
Gestures	3000	3000

Table 4.4: Training and prediction data dimensions.

number of the parameters needed by QDA scales quadratically with that of the variables, making it slower for very high dimensional datasets. Both LDA and QDA are extensively used for bankruptcy status classification and face classification. LDA's training complexity is $O(d^3)$ if $d > N$ (i.e., more features than training samples) and $O(Nd^2)$ if $N < d$ (i.e., more training samples than features). QDA's training complexity is generally $O(d^4)$ [48]. LDA's prediction complexity is $O(N)$ whereas QDA's prediction complexity is $O(N^2)$.

4.1.5 Datasets Used

In this paper, we benchmark all four classes of classification algorithms on the following datasets: KDDCup 1999 (KDDCup) [49], Digits 0-9 (Digits) [50], and Hand Gestures (Gestures) [51]. We have chosen these datasets for the following reasons. First, they represent different application scenarios. Second, they represent different percentages of anomalies at 0.35%, 10%, and 25.3% respectively. Third, they represent different orders of dimensionality at 3, 784, and 64 respectively. Table 4.3 provides an overview of each dataset with regard to anomaly detection scenario, percentage of anomalies, and dimensionality. The original KDDCup dataset from UCI machine learning repository contains 41 attributes. ODDS Library from Stonybrook University, New York has reduced the dataset to 3 attributes: duration of communication, number of incoming bytes, and number of outgoing bytes. Each training sample contains 3 features and an output value indicating whether the network is in a secure or compromised state. The original data set has 3,925,651

attacks (80.1%) out of 4,898,431 records. ODDS has forged this dataset to contain only 3,377 attacks (0.35%) out of 567,497 records. The end goal in this scenario is to only notify the cloud of the sparsely occurring attacks on the network.

Digits is a dataset containing images (28×28) of digits between 0-9. There are a total of 784 features with each feature corresponding to each constituent pixel of a given image. Digit 0 is considered the anomalous class and constitutes roughly 10% of the dataset. We consider digit 0 to represent an unexpected entity or intruder captured within a collection of image frames in a video stream. Anomaly detection can substantially lower transmission overhead when filtration is applied to high dimensional data such as images.

Hand Gestures is a dataset with 64 total attributes each representing sensor measurements of a person's hand while playing the game of rock-paper-scissors. The original dataset classifies each observed motion as either rock (closed fist), paper (open fist), scissors (two fingers pointed), and neutral (flat hand). For our study, we designate rock as the anomalous gesture and group the other three gestures into the norm class. We consider the rock symbol to be emblematic of a patient's hand contraction which requires medical assistance. In this process, we introduce a scenario where 25.3% of the closed fist instances (i.e., anomalies) are representative of a health condition requiring notification to the cloud.

4.2 Experimentation and Analysis

The organization of this section is as follows. First, we specify our testbed, model creation process, and model benchmarking process. Then, we show how each anomaly detection model performs on each dataset (i.e., scenario) with respect to both overhead and anomaly detection accuracy. Furthermore, we show the best

model for each scenario and analyze tradeoffs involved in model selection such as training overhead, prediction overhead, and prediction accuracy.

4.2.1 Methodology

The testbed includes three RPi3s, which are used as the edge device, the cloud, and the interface to connect to our energy measurement platform, EMPIOT [52]. All RPi3s feature a BCM2837 SoC, a 1.2 GHZ quad-core ARM Cortex A53 processor, and 1 GB LPDDR2-900 SDRAM and run Debian OS [53]. Both the client and cloud RPi3s connect to an access point (i.e., router) through an 802.11 link. We run and benchmark all machine learning algorithms for anomaly detection on the edge RPi3. Time measurements are captured by setting timestamps in the Python code before and after algorithm execution. Energy measurements are captured by EMPIOT. EMPIOT is composed of a shield and is installed on top of the host RPi3. Figure 4.1 shows our experimental setup inclusive of edge, cloud, and EMPIOT.

We use scikit-learn implementations of RF, MLP, KNN, LDA, and QDA for our anomaly detection experiment. There is minimal data preprocessing for the three datasets used in this experiment because they do not contain missing column values nor incorrect formatting. All code related to anomaly detection such as data aggregation and model training/validation is written in Python 3.6 making use of bcrypt, pandas, numpy, and scikit-learn libraries. For each dataset, we first tune and validate each model by running an offline grid search (i.e., hyperparameter tuning). We configure the grid search to rank model configurations based on precision and recall. Note that we consider grid search to be a purely offline operation performed on the cloud and do not consider its overhead. We also

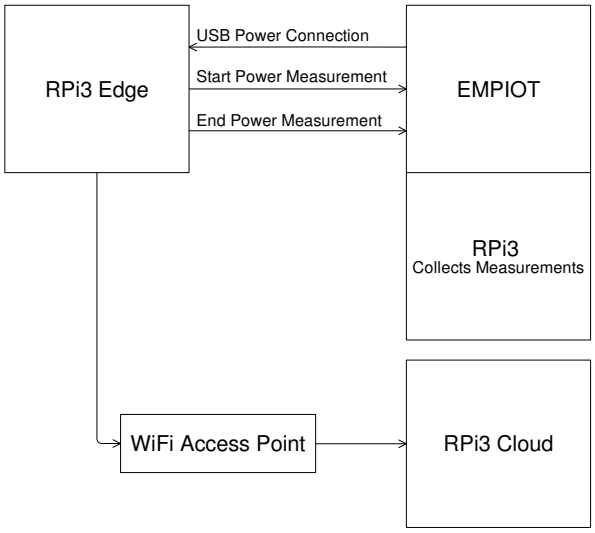
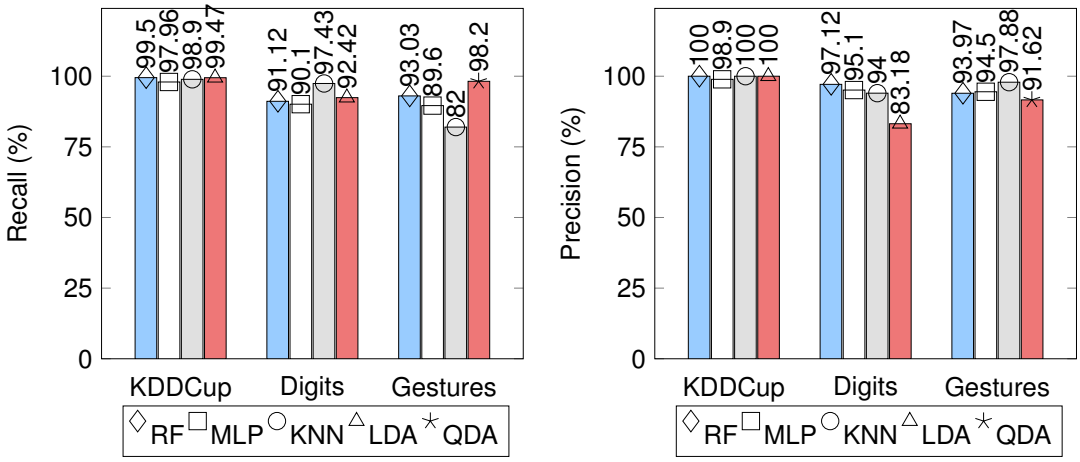


Figure 4.1: The testbed used for our experiment inclusive of three RPi3s and the EMPIOT energy measurement platform. The RPi3 hosting EMPIOT captures energy measurements of the edge RPi3 transmitting data to the cloud RPi3.

note that grid search is not necessarily exhaustive in all scenarios because not all models require extensive hyperparameter tuning. For example, RF generally demonstrates high prediction accuracy on all datasets using default scikit-learn parameters. MLP, on the other hand, requires extensive hyperparameter tuning to converge on hidden layer and node count. Having decided on hyperparameter configuration, we are left with one-time model training. For each scenario, we train a model with the least number of samples that enables it to predict with both precision and recall greater than 80% and either precision or recall greater than 90%. Table 4.4 shows the training data size for each dataset. We choose to benchmark model training on the RPi3 edge to provide reference for cases when the edge needs to carry on model training. Note that we also consider model training to be an offline operation and therefore will not factor this cost into our overhead-savings analysis.

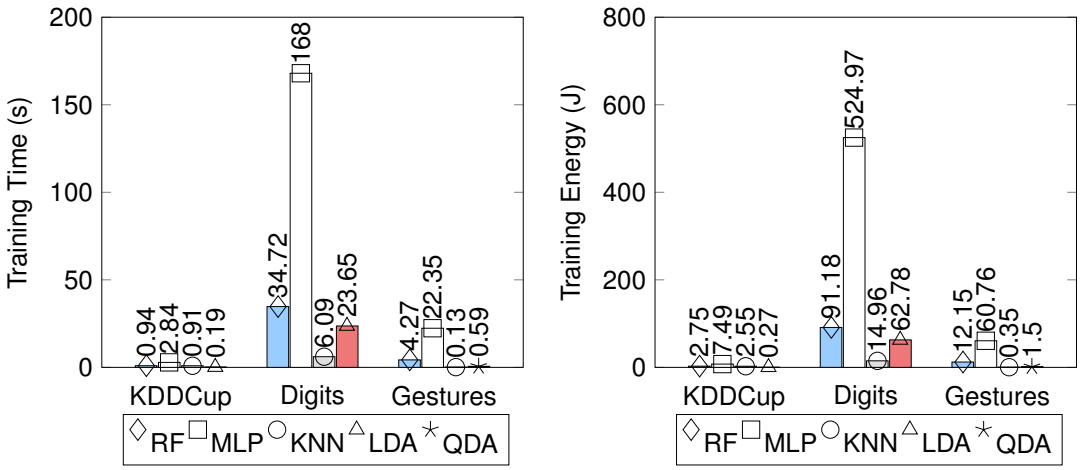
We benchmark model precision and recall for each dataset. Precision indicates the percentage of relevant results $Precision = \frac{TruePositiveCount}{TruePositiveCount+FalsePositiveCount}$.



(a) (b)

Figure 4.2: All models are viable for the datasets chosen. All models perform best for the KDDCup dataset, containing the lowest number of features, and post precision and recall values of 98.9% and 97.96% respectively. RF and QDA offer the best prediction accuracy for Gestures while RF and KNN offer the best prediction accuracy for Digits.

Recall indicates the percentage of results accurately classified by the algorithm $Recall = \frac{TruePositiveCount}{TruePositiveCount+FalseNegativeCount}$. We also tabulate the time and energy consumption of each model's training phase and prediction phase. The training phase entails instantiating a model with a hyperparameter configuration and fitting the model on training samples. The prediction phase entails executing the model predict function on a buffer of test samples. Time is measured by setting timers within the Python code while energy is measured using EMPIOT. All data transmitted to the cloud is secured using 256-bit AES encryption. Note that encryption overhead is factored into our cost analysis. For each model and dataset, we show precision, recall, prediction time and energy, training time and energy, the number of observations and MBs saved by the cloud, and lastly, the time and energy saved using anomaly detection. We then closely analyze the quantitative results and study the performance tradeoffs of different models in different scenarios.



(a) (b)

Figure 4.3: MLP features the most expensive training phase with RF, QDA, LDA, and KNN generally following in this order as shown in (a) and (b). MLP costs approximately 500% more time and 500% more energy than the next best performing model across all datasets.

4.2.2 Results and Analysis

We present the model performance and model overhead results obtained from each dataset. We recommend the best model for each scenario based on training and prediction overhead and model performance. Note that the time and energy savings reported for each anomaly detection model consider prediction cost and not the offline training cost. We conclude this section with an analysis of general model behavior observed across all datasets and propose recommended use cases for each model.

KDDCup Dataset

For the 3 feature KDDCup dataset, all models are trained using 10000 network observations out of a total of 567497 labeled training samples with 0.35% anomaly rate. Note that this dataset has the lowest dimensionality as well as the lowest anomaly rate among the three datasets. In this scenario, the edge device aggre-

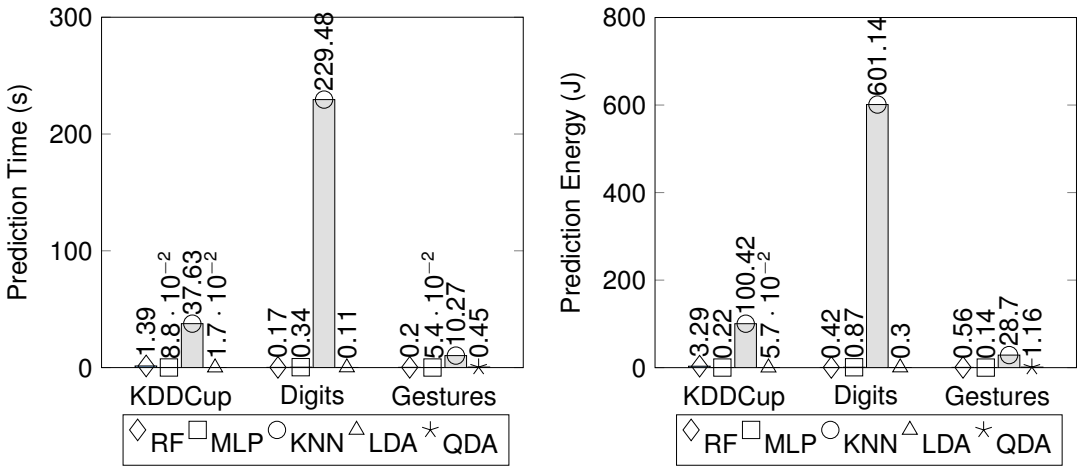


Figure 4.4: LDA offers the most cost-effective prediction phase for both Digits as well as KDDCup datasets as shown in (a) and (b). MLP offers the most cost-effective prediction phase for the 64 feature Gestures dataset. KNN costs nearly 600% more energy and 650% more time than the next best performing model across all datasets.

gates 50000 size network status buffers for the prediction phase (i.e., anomaly detection) and subsequent transmission.

Figure 4.2 shows the precision and recall values observed when applying each model for each scenario. We see that all models post exceptional precision and recall values on the sparse anomaly KDDCup dataset. Relatively speaking, MLP posts the lowest precision and recall values at 98.9% and 97.9% respectively. All other models post over 99% recall and 100% precision. We subsequently examined that one of the features in this dataset has a Gaussian distribution and that the anomalies primarily occur when the feature's value lied $\pm 3\sigma$ outside of the mean. Given this level of accuracy in anomaly detection, we assert that the edge device can save around 99.5% of the data (i.e., approximately 1.2 MB) per buffer sent to the cloud.

Figure 4.3 shows the training time and energy for each model applied on different datasets. Specifically for KDDCup, LDA has the most time and energy

efficient training phase among all models. This is because LDA uses an underlying dimensionality-reduction technique for generating a class-separating boundary that is efficiently performed on a 3 dimensional dataset. KNN, RF, and MLP follow in order. RF takes a significantly longer time proportional to KNN for this dataset due to the relatively large test data buffer size.

Figure 4.4 shows the prediction time and energy when each model is applied. As shown in Figure 4, when applied on the KDDCup data, LDA is also the most cost-effective model for prediction. The low dimensionality of inputs to the model enables LDA to classify anomalies very efficiently. It is followed by MLP, RF, and KNN. MLP is marginally worse than LDA for this scenario considering the fact that it is slightly inferior in terms of prediction accuracy. RF is a well-rounded choice with respect to both prediction accuracy and overhead. Note that RF's prediction overhead exceeds LDA's prediction overhead due to the complexity involved in processing test samples at multiple nodes at multiple tree levels. KNN is rendered slow and ineffective for this scenario as its prediction phase consumes nearly 38 seconds and 101J for a 50000×3 buffer.

Digits Dataset

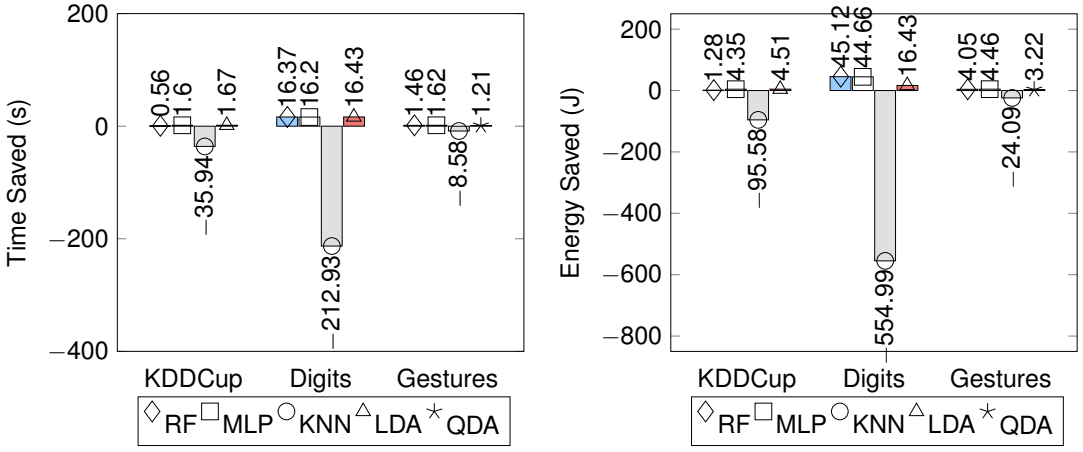
The 784 feature Digits dataset has the highest dimensionality among all our datasets and is emblematic of an anomalous image detection scenario. For this dataset, all models are trained using 3000 images out of the total 20000 images with a 10% anomaly rate. The edge device performs anomaly detection on buffers containing 2000 images and transmits the anomalous images identified.

Regarding prediction accuracy, Figure 4.2 shows that KNN offers the best overall precision and recall among all models at 94% and 97.4% respectively. RF

posts the highest precision out of all the models at 97.1%. LDA performs the poorest in this scenario offering 92.42% recall but only 83.18% precision. Given the overall high level of accuracy in anomaly detection, we assert that the edge device can save around 89.5% of the data (i.e., approximately 11.3 MB) per buffer sent to the cloud.

As far as training overhead, Figure 4.3 shows that KNN has the most efficient training phase on the Digits dataset. KNN only stores training samples as part of its training phase rather than formulating a mapping between inputs and outputs. Therefore, for this 784 dimensional dataset, the other algorithms have a considerably more demanding training phase. KNN is followed by LDA, RF, and MLP in order. MLP, which has the most expensive training phase, costs 168 seconds and 524.97J. This result is expected given the computationally expensive nature of MLP's backpropagation algorithm and high data dimensionality.

In terms of prediction overhead, Figure 4.4 shows that LDA has the most efficient prediction phase among all models and costs only 0.11 sec and 0.3J for prediction on a 2000 image buffer. It is followed by RF, MLP, and KNN in order. KNN is very expensive for prediction and costs nearly 230 seconds and 600J. This is because the distance computation between K neighbors and all M test samples is costly for 784 dimensional data points. If we prioritize prediction accuracy much higher than prediction time, KNN offers the best precision and recall values at the expense of costly prediction overhead. If the primary objective is to minimize transmission time delay, LDA offers a robust solution at the expense of low precision (i.e., 83.18%). RF and MLP lie in the middle ground and are the two most well-rounded solutions for this scenario. Because RF also has significantly less training overhead, we propose that RF is the best anomaly detection method for this image classification scenario.



(a) (b)

Figure 4.5: RF, MLP, and LDA all demonstrate overhead savings when applied to data before transmission; most notably, RF applied to the Digits dataset saves 45.119 J and 16.37 seconds per 2000 image buffer. KNN is a poor choice for real-time anomaly detection scenarios given that it causes net time and energy loss across all datasets.

Gestures Dataset

For this 64 feature dataset, all the models are trained using 3000 hand gesture observations out of a total of 12000 hand gesture observations with a 25.3% anomaly rate. This dataset has the second highest dimensionality and the highest anomaly rate among all datasets. In this scenario, the edge device aggregates 3000 observations (gestures) per buffer for the prediction phase and subsequent transmission. Note that we use QDA only on this dataset instead of LDA as LDA posted less than 20% precision. This is because LDA could only generate a linear fit for certain features in this dataset that exhibited quadratic patterns. Nevertheless, we benchmarked LDA and observed that it takes 0.43 seconds for the training phase and 0.051 seconds for the prediction phase. Therefore, if LDA demonstrated acceptable prediction accuracy for Gestures, it would have claimed the second most efficient training phase and the most efficient prediction phase among all models.

With regard to prediction accuracy, Figure 4.2 shows that QDA posts the

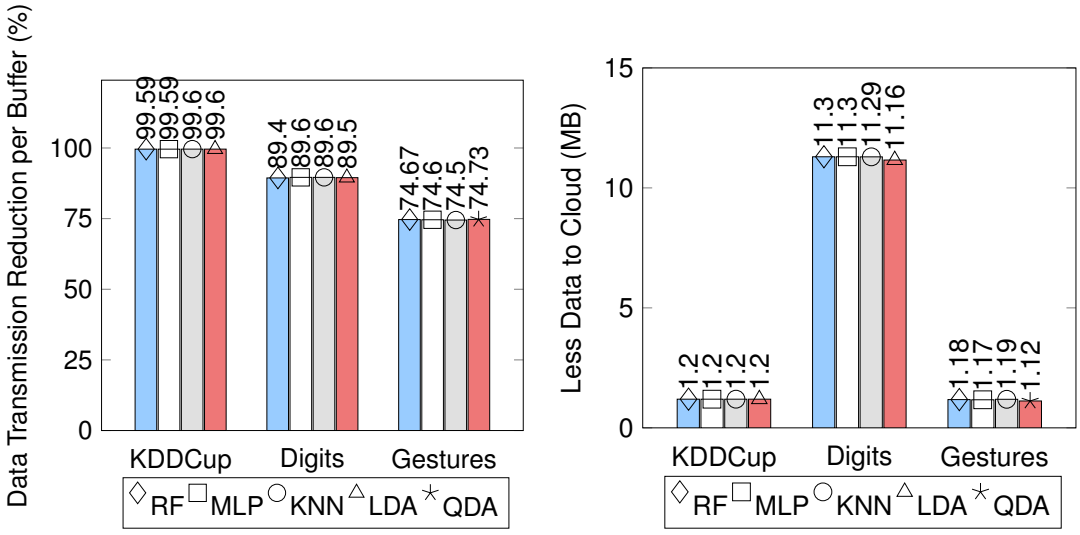


Figure 4.6: From (a) we observe that the models reduce approximately 90% of the observations sent from Digits' buffers, 76% of the observations sent from Gestures' buffers, and 99.5% of the observations sent from KDDCup's buffers. The resulting saved cloud storage shown in (b) is substantial, especially for Digits dataset where we saved nearly 12 MB of data sent per 2000 image buffer.

highest recall out of all models at 98.2%. RF also performs well with 93.03% recall and 93.97% precision. It is followed by MLP and KNN in order. Given the overall high level of accuracy in anomaly detection, we assert that the edge device can save around 75% of the data (i.e., approximately 1.15 MB) per buffer sent to the cloud.

KNN training, as shown in Figure 4.3, is the most efficient and MLP training is the least efficient. RF training costs substantially more overhead than QDA. This is explained by the computation involved in creating a set of randomized decision trees for a pool of 64 features.

As far as prediction overhead, MLP has the most efficient prediction phase for the Gestures dataset and consumes 0.054 seconds and 0.142J per buffer. This is 274% more time efficient and 294% more energy efficient than the next best RF prediction phase. QDA and KNN follow in order. MLP is the clear choice for this

scenario because it offers fast anomaly detection and the best cost-savings among all four models.

General Observations

We note that time and energy savings for each anomaly detection model is calculated by subtracting both prediction phase overhead and anomalous buffer transmission overhead from full buffer transmission overhead. Figure 4.5 shows the overhead savings observed when applying each model on each dataset. LDA provides the most data transmission overhead savings among all models. MLP, QDA, RF, and KNN follow in order. Considering that the deployment of KNN leads to net overhead loss across all benchmarked scenarios, its use may be eliminated from real-time anomaly detection scenarios.

Comparing both Figure 4.5 and Figure 4.4, we note that the rank ordering of models' overhead savings from least to greatest matches the rank ordering of models' prediction phase overhead from greatest to least. From Figure 4.4, note that QDA, which is used in place of LDA for the Gestures dataset, substantially exceeds LDA in prediction phase overhead consumption. This explains why MLP has the fastest prediction phase for the Gestures dataset but has the second and third slowest prediction phase for KDDCup and Digits datasets respectively. Also note that RF has the second most efficient prediction phase for Digits but is substantially less efficient than MLP for KDDCup. This suggests that RFs are more sensitive to testing buffer size than dimensionality. For reference purposes, the rank ordering of training time complexity from greatest to least is generally MLP, RF, QDA, LDA, and KNN across all datasets. Note that all rankings presented are based on testing with Python's scikit-learn library and experimental results may vary when using other software implementations of the machine learning models.

Figure 4.6 shows percentages of buffer size reduction and fewer MBs of data sent to the cloud upon applying each model on each dataset. The amount of data storage conserved on the cloud depends on the dateset's anomaly rate and dimensionality. Low anomaly rate indicates that there will be a proportionally smaller number of observations sent to the cloud. Low dimensionality indicates that there will be fewer bytes of data per observation sent to the cloud. The cloud would benefit the most when applying anomaly detection for a scenario with very high dimensional data and low anomaly detection rate. In this way, the cloud will not receive the vast majority of data points sent from the edge. With this in mind, we will examine the cloud storage savings observed for each discussed scenario. KDDCup has the lowest dimensionality (i.e., 3) and lowest anomaly rate (i.e., 0.35%) among all the datasets. Therefore, we are able to filter over 99% of the observations captured on the edge but save only 3 features per observation. This explains why we are only able to save approximately 1.2 MB sent to the cloud during prediction phase even though the buffer size is 50000. Digits has the highest dimensionality (i.e., 784) among all datasets and falls between the other two datasets with regard to anomaly rate (i.e., 10%). Despite having a considerably higher anomaly rate than KDDCup, Digit's dimensionality ensures that we save 784 data points sent to the cloud per observation. Thus, by filtering 90% of each Digits buffer we save roughly 12.3 MB sent to the cloud. Lastly, Gestures has the highest anomaly detection rate (i.e., 25.3%) and ranks second as far as dimensionality (i.e., 64). This indicates that there is a substantially higher proportion of observations needed to be notified to the cloud compared to the other scenarios and a moderate level of data points saved per filtered observation. These two factors slightly downgrade cloud storage savings and result in approximately 1.15 MB saved by the cloud per data buffer.

Model	Suitable Applications
RF	Minimal training samples Delay sensitive applications
MLP	Non-linear relationship between training inputs and outputs Extremely time-sensitive application
KNN	Resource constrained training Delay insensitive application
DA	Resource constrained training and prediction Delay sensitive and mission critical application

Table 4.5: Sample use cases for different models.

Chapter 5
Conclusion

Both studies in this thesis offer ways to reduce resource consumption of IoT devices. In Energy and Processing Demand Analysis of TLS Protocol in Internet of Things Applications, we benchmark three different ciphers and present the most lightweight configuration when running the TLS handshake. Every fraction of time and energy saved per TLS handshake pays enormous dividends in the long run considering that TLS authentication is utilized by many billions of IoT devices worldwide. In Edge Mining on IoT Devices using Anomaly Detection, we propose using supervised machine learning anomaly detection techniques to considerably reduce data sent to the cloud. This can considerably reduce edge device data transmission overhead costs as well as cloud storage requirements.

In our TLS benchmarking study, we conclude that the order of the ciphers from least to greatest energy efficiency is the following: C1, C3, C2. Specifically, C1 consumes 55% more energy than C3, and C3 consumes 150% more than C2. Clearly, the most efficient cipher based on the results is C2 (i.e., ECDHE using RSA). However, when choosing between RSA and ECDSA encryption, there are two considerations: RSA generally has heavier signatures than ECDSA, however, ECDSA requires more computation for certificate verification. Our study proves that the energy consumption of ECDSA certificate verification considerably outweighs RSA's energy consumption for heavier signature generation. We plan on expanding this study by extending our pool of ciphers, ranking the ciphers based

on level of security, and including network overhead, using various types of IoT devices. For future work, we would like to benchmark TLS with additional cipher suite configurations using multiple hardware platforms.

In our edge mining anomaly detection study, we proved that RF, MLP, LDA, and QDA anomaly detection models have considerable potential to save edge device transmission overhead as well as cloud storage. The overhead-savings for a generic IoT scenario varies depending upon the anomaly rate and dimensionality of the transmitted data. We conclude that LDA has the most cost-effective anomaly detection phase that we have benchmarked across all scenarios and should be the primary choice for an extremely resource constrained edge device. QDA also has a very efficient anomaly detection phase but undoubtedly demands more overhead than LDA for the quadratic fit operation. We also conclude that RF is the most well-rounded anomaly detection method among all models featuring a comparably lightweight prediction phase and offering exceptional precision and recall. MLP also works very well for time-critical prediction tasks given that there are not significant resource constraints for training. The KNN classifier, despite its reliable prediction accuracy, demands excessive amounts of time and energy for anomaly detection, which rules out its use case in most IoT scenarios. The only reason to consider using KNN is in a case of stringent resource constraints for model training.

This work clearly demonstrates the overhead-savings potential of machine learning based anomaly detection on both edge and cloud. We also have provided a comprehensive overview of the tradeoffs involved in the deployment of these models. For future work, we aim to benchmark unsupervised classification methods for anomaly detection. Unsupervised machine learning methods are very useful when we do not have ground truth labeling but can infer properties from

the training dataset. For example, Elliptical Envelope is a suitable technique for a dataset which expresses a multivariate gaussian distribution and an Isolation Forest is optimal for a dataset which expresses a multimodal distribution. For future contribution, we also aim to scale our experimental setup to other IoT platforms such as Cypress CYW43907.

Bibliography

[1] M. Shirvanimoghaddam, M. Dohler, and S. J. Johnson, "Massive non-orthogonal multiple access for cellular iot: Potentials and limitations," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 55–61, 2017. 1

[2] R. Mzid, M. Boujelben, H. Youssef, and M. Abid, "Adapting tls handshake protocol for heterogenous ip-based wsn using identity based cryptography," in *International Conference on Communication in Wireless Environments and Ubiquitous Systems: New Challenges (ICWUS)*. IEEE, 2010, pp. 1–8. 2

[3] D. E. Simos, K. Kleine, A. G. Voyiatzis, R. Kuhn, and R. Kacker, "Tls cipher suites recommendations: A combinatorial coverage measurement approach," in *International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2016, pp. 69–73. 2, 13

[4] U. Banerjee, C. Juvekar, A. Wright, A. P. Chandrakasan *et al.*, "An energy-efficient reconfigurable dtls cryptographic engine for end-to-end security in iot applications," in *International Solid-State Circuits Conference (ISSCC)*. IEEE, 2018, pp. 42–44. 2, 3

[5] N. J. Al Fardan and K. G. Paterson, "Lucky thirteen: Breaking the tls and dtls record protocols," in *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2013, pp. 526–540. 2

[6] A. K. Ranjan, V. Kumar, and M. Hussain, "Security analysis of tls authentication," in *International Conference on Contemporary Computing and Informatics (IC3I)*. IEEE, 2014, pp. 1356–1360. 2

[7] C. Peng, Q. Zhang, and C. Tang, "Improved tls handshake protocols using identity-based cryptography," in *International Symposium on Information Engineering and Electronic Commerce (IEEC'09)*. IEEE, 2009, pp. 135–139. 2

[8] M. Atighetchi, N. Soule, P. Pal, J. Loyall, A. Sinclair, and R. Grant, "Safe configuration of tls connections," in *Conference on Communications and Network Security (CNS)*. IEEE, 2013, pp. 415–422. 2

[9] P. Miranda, M. Siekkinen, and H. Waris, "Tls and energy consumption on a mobile device: A measurement study," in *Symposium on Computers and Communications (ISCC)*. IEEE, 2011, pp. 983–989. 2, 3, 6, 7

[10] N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha, "A study of the energy consumption characteristics of cryptographic algorithms and security protocols," *IEEE Transactions on mobile computing*, vol. 5, no. 2, pp. 128–143, 2006. 3, 7, 15

[11] A. Emdadi, R. Karne, and A. Wijesinha, "Implementing the tls protocol on a bare pc," in *Second International Conference on Computer Research and Development*. IEEE, 2010, pp. 293–297. 3

[12] S. Gueron and V. Krasnov, "Fast prime field elliptic-curve cryptography with 256-bit primes," *Journal of Cryptographic Engineering*, vol. 5, no. 2, pp. 141–151, 2015. 3

[13] L.-S. Huang, S. Adhikarla, D. Boneh, and C. Jackson, "An experimental study of tls forward secrecy deployments," *IEEE Internet Computing*, vol. 18, no. 6, pp. 43–51, 2014. 3

[14] E. I. Gaura, J. Brusey, M. Allen, R. Wilkins, D. Goldsmith, and R. Rednic, "Edge mining the internet of things," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3816–3825, 2013. 4

[15] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing.* ACM, 2012, pp. 13–16. 4

[16] K. Bhargava and S. Ivanov, "Collaborative edge mining for predicting heat stress in dairy cattle," in *2016 Wireless Days (WD).* IEEE, 2016, pp. 1–6. 4

[17] I. Butun, B. Kantarci, and M. Erol-Kantarci, "Anomaly detection and privacy preservation in cloud-centric internet of things," in *IEEE International Conference on Communication Workshop (ICCW).* IEEE, 2015, pp. 2610–2615. 4

[18] S. Bin, L. Yuan, and W. Xiaoyi, "Research on data mining models for the internet of things," in *International Conference on Image Analysis and Signal Processing.* IEEE, 2010, pp. 127–132. 4

[19] P. G. Argyroudis, R. Verma, H. Tewari, and D. O'Mahony, "Performance analysis of cryptographic protocols on handheld devices," in *Third IEEE International Symposium on Network Computing and Applications, 2004.(NCA 2004). Proceedings.* IEEE, 2004, pp. 169–174. 6

[20] A. Narayan and M. Chen, "Measuring the energy impact of security protocols," Ph.D. dissertation, 2016. 7

[21] M. Koschuch, M. Hudler, and M. Krüger, "Performance evaluation of the tls handshake in the context of embedded devices," in *2010 International Conference on Data Communication Networking (DCNET)*. IEEE, 2010, pp. 1–10. 7

[22] F. Giannoni, M. Mancini, and F. Marinelli, "Anomaly detection models for iot time series data," *arXiv preprint arXiv:1812.00890*, 2018. 8

[23] S. A. Aljawarneh and R. Vangipuram, "Garuda: Gaussian dissimilarity measure for feature representation and anomaly detection in internet of things," *The Journal of Supercomputing*, pp. 1–38, 2018. 8

[24] H. Haddad Pajouh, R. Javidan, R. Khayami, D. Ali, and K. R. Choo, "A two-layer dimension reduction and two-tier classification model for anomaly-based intrusion detection in iot backbone networks," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2019. 8

[25] S. Raza, L. Wallgren, and T. Voigt, "Svelte: Real-time intrusion detection in the internet of things," *Ad hoc networks*, vol. 11, no. 8, pp. 2661–2674, 2013. 8

[26] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *IEEE Symposium on Security and Privacy*, May 2010, pp. 305–316. 8

[27] J. Zhang and M. Zulkernine, "Anomaly based network intrusion detection with unsupervised outlier detection," in *IEEE International Conference on Communications*, vol. 5, June 2006, pp. 2388–2393. 8

[28] G. Kotani and Y. Sekiya, "Unsupervised scanning behavior detection based on distribution of network traffic features using robust autoencoders," in *IEEE*

International Conference on Data Mining Workshops (ICDMW), Nov 2018, pp. 35–38. 8

[29] S. R. Islam, D. Kwak, M. H. Kabir, M. Hossain, and K.-S. Kwak, "The internet of things for health care: a comprehensive survey," *IEEE Access*, vol. 3, pp. 678–708, 2015. 8

[30] A. Ukil, S. Bandyoapdhyay, C. Puri, and A. Pal, "Iot healthcare analytics: The importance of anomaly detection," in *IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, March 2016, pp. 994–997. 8

[31] R. K. Mishra and R. Pandey, "Aspects of network architecture for remote healthcare systems," in *2nd International Conference on Computational Intelligence and Networks (CINE)*, Jan 2016, pp. 47–53. 8

[32] D. Stiawan, M. Y. Idris, R. F. Malik, S. Nurmaini, and R. Budiarto, "Anomaly detection and monitoring in internet of things communication," in *8th International Conference on Information Technology and Electrical Engineering (ICITEE)*, Oct 2016, pp. 1–4. 9

[33] T. Luo and S. G. Nagarajan, "Distributed anomaly detection using autoencoder neural networks in wsn for iot," in *IEEE International Conference on Communications (ICC)*, May 2018, pp. 1–6. 9

[34] H. Sedjelmaci, S. M. Senouci, and T. Taleb, "An accurate security game for low-resource iot devices," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 10, pp. 9381–9393, 2017. 9

[35] L. Lyu, J. Jin, S. Rajasegarar, X. He, and M. Palaniswami, "Fog-empowered anomaly detection in iot using hyperellipsoidal clustering," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1174–1184, 2017. 9

[36] T. Dierks, "The transport layer security (tls) protocol version 1.2," 2008. [Online]. Available: https://tools.ietf.org/html/rfc5246 11

[37] ARM Limited, "mbed tls," https://github.com/ARMmbed/mbedtls, Mar. 2017. 14, 15

[38] J. Salowey, A. Choudhury, and D. McGrew, "Aes galois counter mode (gcm) cipher suites for tls," Tech. Rep., 2008. 14

[39] Cypress Semiconductor. CYW43907: IEEE 802.11 a/b/g/n SoC with an Embedded Applications Processor. [Online]. Available: http://www.cypress.com/file/298236/download 15

[40] ——. CYW943907AEVAL1F Evaluation Kit. [Online]. Available: http://www.cypress.com/documentation/development-kitsboards/cyw943907aeval1f-evaluation-kit 15

[41] B. Dezfouli, I. Amirtharaj, and C.-C. Li, "EMPIOT: An Energy Measurement Platform for Wireless IoT Devices," 2018. 15

[42] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001. 24

[43] G. Louppe, "Understanding random forests: From theory to practice," *arXiv preprint arXiv:1407.7502*, 2014. 25

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Ma-

chine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011. 25

[45] S. B. Imandoust and M. Bolandraftar, "Application of k-nearest neighbor (knn) approach for predicting economic events: Theoretical background," *International Journal of Engineering Research and Applications*, vol. 3, no. 5, pp. 605–610, 2013. 26

[46] H. Neeb and C. Kurrus, "Distributed k-nearest neighbors," 2016. 26

[47] G. Guo, H. Wang, D. Bell, Y. Bi, and K. Greer, "Knn model-based approach in classification," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2003, pp. 986–996. 26

[48] B. Jiang, X. Wang, and C. Leng, "A direct approach for sparse quadratic discriminant analysis," *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 1098–1134, 2018. 26, 27

[49] "Http (kddcup99) dataset," http://odds.cs.stonybrook.edu/http-kddcup99-dataset/, accessed: 2019-01-30. 27

[50] "Mnist digits," http://yann.lecun.com/exdb/mnist/, accessed: 2019-01-30. 27

[51] "Classify gestures by reading muscle activity," https://www.kaggle.com/kyr7plus/emg-4/metadata, accessed: 2019-01-30. 27

[52] B. Dezfouli, I. Amirtharaj, and C.-C. C. Li, "Empiot: An energy measurement platform for wireless iot devices," *Journal of Network and Computer Applications*, vol. 121, pp. 135–148, 2018. 29

[53] E. Upton and G. Halfacree, *Raspberry Pi user guide*. John Wiley & Sons, 2014. 29