UNIVERSITY NAME (IN BLOCK CAPITALS)

Thesis Title

by

Author Name

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the
Faculty Name
Department or School Name

January 2020

Declaration of Authorship

I, AUTHOR NAME, declare that this thesis titled, 'THESIS TITLE' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ______________________________

Date: ______________________________

"Write a funny quote here."

If the quote is taken from someone, their name goes here

UNIVERSITY NAME (IN BLOCK CAPITALS)

Abstract

Faculty Name
Department or School Name

Doctor of Philosophy

by Author Name

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgements and the people to thank go here, don't forget to include your project advisor…

Contents

List of Figures

List of Tables

Abbreviations

LAH	**L**ist **A**bbreviations **H**ere

For/Dedicated to/To my…

Chapter 1

Introduction

The later part of the past decade has seen a tremendous increase in the research of Internet of Things (IoT) devices mostly geared towards energy efficiency. This is in part, due to the fact that IoT devices have an underlining positive factor of having to save energy usage and hence must be design as not excessively use energy in itself. Not only do these interconnected devices provide comfort but have also become critical parts of our daily lives: saving lives, expediting interactions and transactions. Another factor that fueled the advancement in the research of energy efficient IoT devices is the enormous progress made in the field of Machine learning specifically Reinforcement learning.

Maximizing a numerical reward signal by learning which actions to take in a given situation is what Reinforcement learning is about. The actions to take by the learning agent pertaining to various situations are not preprogrammed into the learner, but instead this is discovered by taking the actions that maximize the reward margin[1]. This works in the cycle of sense-action-goals and learning is from immediate interactions with the environment.

The APT-MAC protocol, which is the focus of this paper, seeks to solve the problem of the reliance on battery. The protocol utilizes the Multi-Arm Bandit algorithm, a reinforcement learning algorithm, in tandem with Radio-frequency identification (RFID) signals to enable battery-free communication of wireless devices[2].

The aim of this work is to simulate the APT-MAC protocol in NS3 and in comparison to the static TDMA protocol.

1.1 The Protocols

Even though the quality of a network service is a cooperative effort of all the stack of communication protocols, the MAC layer is of a peculiar importance as it handles the sharing of the medium on which all other upper-layered protocols depend. MAC protocols do not only solve the problem of medium sharing, hence, support for reliable communication but also, in Wireless Sensor Networks, control of energy utilization, achieve through duty cycling and retransmissions or transmission power control[3].

In wireless sensor network, the MAC protocols are broadly classified into contention-based MAC protocols and scheduled-based MAC protocols depending on how he medium is accessed[4].

The medium in a contention-base MAC protocol is accessed by all the nodes and, as the name suggests, the nodes contend for access to the medium this may result in collision hence, to prevent collision access to the medium is negotiated through probabilistic coordination. A sending node listens to shared medium before sending, if the medium is busy, sending is halted for a specified period of time before retrying. Examples of contention-based MAC protocols used in wireless sensor network are ALOHA (Additive Link On-line Hawaii System)and CSMA (Carrier Sense Multiple Access)[5].

Nodes' medium access in a schedule-based MAC protocol is split into either frequency, Frequency Division Multiple Access, or time, that is, Time Division Multiple Access, or orthogonal pseudo - noise codes (Code Division Multiple Access). Collision is prevent in the medium by making different nodes access the medium in their designated time or frequency and hence not interfering with each other.

This section deals with the description of the protocol: APT-MAC and Time Division Multiple Access (TDMA).

1.1.1 TDMA

In TDMA, the total duration of communication is divided into fixed number of time slots. Time slots are configured into time-frames that are repeated periodically. Each node is allocated a time slot in a time-frame and is allowed to only transmit in that time slot. The TDMA frame structure has extra overhead; preamble and trail bits, added to the information message or bits. Time slots are what make up the information bits/message. Figure 1.1 shows the details of the frame structure.
Preamble: contains the address and synchronization information of the base (sink) node and the identification information of the other nodes.
Guard time: necessary to prevent time-drifting over time.
Trail bit: for error detection in the form of checksum or cyclic redundancy check.

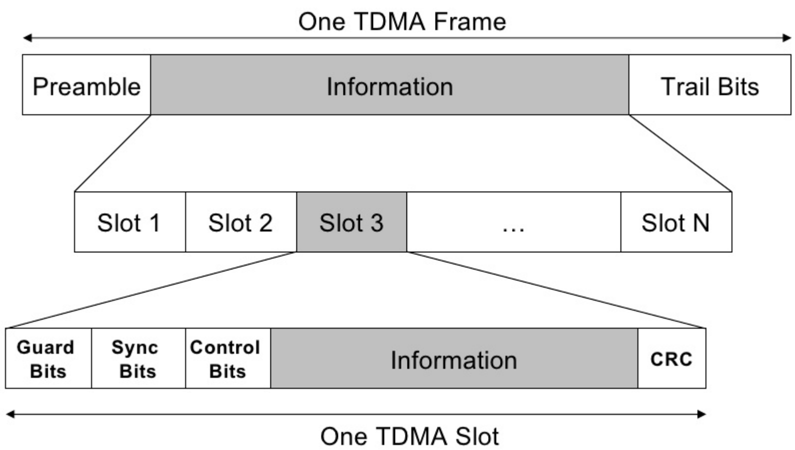


FIGURE 1.1: Frame Structure

The inadequacies of TDMA, chief among them with respect to the aim of the referenced paper [2], is static slot assignment. The slots are assigned before transmission and does not change whatsoever during the transmission process, this does not serve the purpose of a protocol that is needed to scale according to the transmission requirements of the nodes.

1.1.2 APT-MAC

APT-MAC protocol is a zero configuration - manual configuration not needed - mac protocol. Reinforcement learning is utilized to learn the required rate of transmission of each active node and continually update the transmission requirements of nodes. An overview of the reinforcement learning algorithm is given, followed by how it is used the APT-MAC protocol.

1.1.2.1 Multi-Arm Bandit

When one is made to choose between multiple actions with an unknown reward for each action taken and the goal being to maximize the profit margin through a series of actions, the problem is classified as the multi-arm bandit problem. For example, a website deciding on which adverts to show to its visitors. The proprietor must maximize advertisement revenue but does not have enough information about the visitor to pursue a specific strategy, however, there are a large pool of adverts to choose from. This begs the question of which adverts will drive the maximum revenue. The websites, hence, needs to make a series of decisions, each with unknown results and rewards.

When a set of action(s) is found to maximize reward and those are actions are selected over and over again it is termed *Exploitation*, thus the knowledge of the actions is being exploited whereas when an action is selected for which the reward is not known is *Exploration.* An optimum balance between the exploitation and exploration is needed in most cases to achieve maximum reward.

From a more scientific standpoint, Bernoulli multi-armed bandit can be described as an *action reward-function* tuple $\langle A, R \rangle$. At each time step t an action $A = a$ is taken and a reward $R = r$ is received with reward probability $\{\theta_1 ... \theta_K\}$. The expected reward which is the value of the action taken is given by $Q(a_t) = \mathbb{E}[\, R | A = a] = \theta$. The goal is to maximize $\sum_{t=1}^{T} r_t$, the cummulutative reward, hence an optimal action a^* with and optimal probability θ^* thus:

$$\theta^* = Q(a^*) = \max_{a \in A} Q(a) \tag{1.1}$$

Broadly, the various algorithmic solutions to multi-arm bandit problem can be categorized as; no exploration - greedy, random exploration and exploration intelligently with reference to probability [1].

1.1.2.2 APT-MAC protocol overview

The "battery-freeness" of the protocol is as a result of the use of sensor-augmented RFID tags.

During initialization, the reader sends discovery queries to all nodes and each is assigned a unique identification. In querying tags in such a way as to minimize the difference between the sensor's data generation time and the time of delivery of data to the reader, the Multi-Arm Bandit, describe above, is used. The reader, agent in this case, queries each node - set of actions - and can be in one of two states: querying or ready to query. Expected reward is calculated with the formula 1.2 with the expected reward of each action stored in vector Q.

$$Q(a_i)(n+1) = Q(a_i)(n) + \alpha(Reward - Q(a_i)(n)) \tag{1.2}$$

n is the time slot of the query, a_i is the action of querying tag i and α is the learning rate.
A more detailed explanation of the protocol, such as the learning rate parameter and the evaluation of the *Reward* can be found in [2].

Chapter 2

Simulation

The activities of the simulation is described in this chapter. An overview of NS-3 is given, which is then followed by the actions taken, setup used, assumptions made and overall explanation of the simulation.

2.1 NS-3

A discrete event network simulator, NS-3 is used extensively in network research and education. The workings and performance of packet data network are modelled and NS-3 provides a platform for the simulation and experimentation of various packet-based research. C++ and Python are the predominant language in which NS-3 is written with *waf* as the build system. Design of NS-3 capitalizes on the object oriented nature of these languages[6].

There are various components of packet-based network, fundamentally there are the endpoint devices, routers,NIC devices ,switches and the medium of exchange. These components,however in NS-3, are abstracted to reflect what the components actually do. The endpoint devices are called *Nodes*, exchange medium - *Channel* and the applications that are generating the packets are *Applications* just to name a few. The file/folder structure of a simulation consist of a model which depicts the fine details of the simulation, a helper that is supposed to include files containing the installation helper functions, an example folder to implement an example of the simulation. There are also doc and tests folders and the just as their names suggest they hold documentation and tests files.

Installation of the NS-3 is not needed to get a simulation running, as the examples or experiments can be run using the binary files gotten from the build process. It is actually not recommended to do installation of NS-3. Installation in this case refers to running the command ***./waf install***.

2.2 Setup

The simulation was performed on a Gentoo Linux flavor computer with a 64 bit Intel Eight-Core processor with model name *Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz* and RAM of 32GB. Version 3-30 of NS-3 provided the platform on which to undertake the research. At the time of setting up, this version was the latest most stable release.

NS-3 is a voluminous software, the faster processor is needed, as a change to any file would mean rebuilding the file and its dependencies and those that depend on it. The version is also critical. Most versions are not backward compartible hence what works on 3.29 might not work on 3.30 without a little tinkering.

Gentoo Linux, even though time consuming and quite advance in installation and setting up, turns to be one of the fastest linux operating systems available. This is partly due to the fact that source codes are compiled on the host computer with specific flags set for optimality.

2.3 Components

For a data packet to move from one endpoint to another, there are components such as, the packet generating and receiving nodes which will have NIC, the medium(s) through which the packet traverses also forms a part of the components.

There are three critical parts, worth mentioning for this simulation; the header which is attached to the packet, tag-augmented sensor devices and the reader, which, in this case, is augmented with a server.

A discussion of the various components follow next.

2.3.1 Packet Header

Various parts of a packet header which is added to a payload enable the packet to get to its intended destination. A 4-byte(32 bits) header was created, having two fields of 16 bits each representing the packet type being sent: data packet or broadcast and the state of the tag augmented sensor: whether a new data is available for transmission or not.

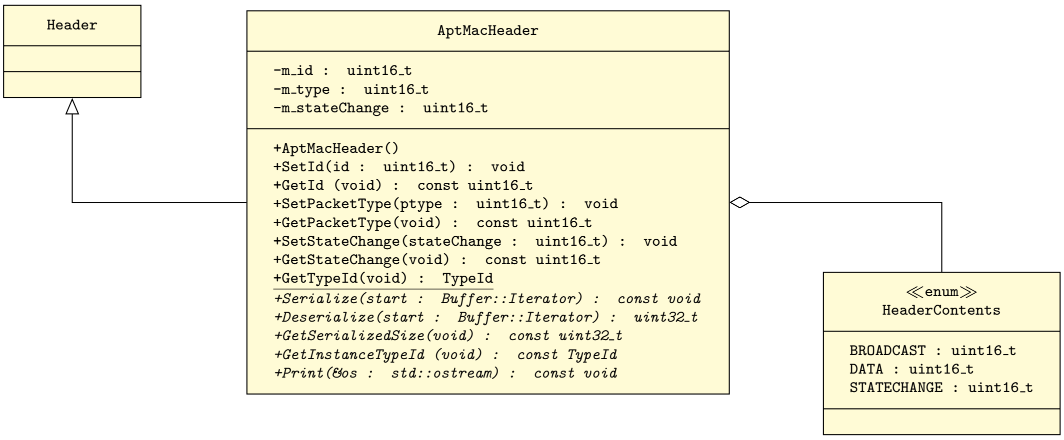
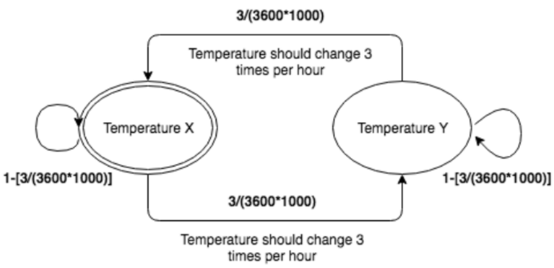


FIGURE 2.1: Class Diagram - APT-MAC Header

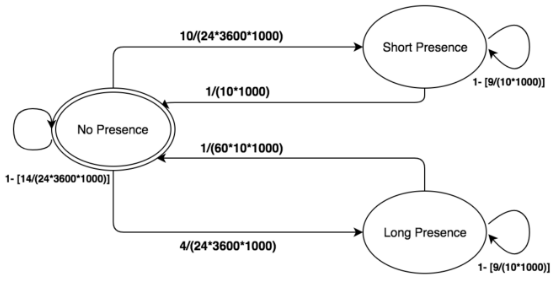
The UML class diagram is illustrated in Figure 2.1. The *constructor, getter and setter* functions are for their usual purposes. *Serialize* function puts the various fields of the header in series by writing them from host-order to network-order and the *Deserialize* reads from network-order to host-order. *Enum HeaderContents* holds the value of the various fields of the header.

2.3.2 Tag Augmented Sensor Devices

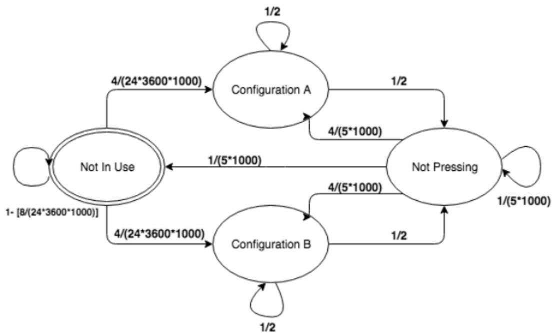
Devices in a smart home are broadly classified under three categories: periodic (e.g., temperature sensors), real-time (e.g., joystick, cameras) and event based (e.g., presence detector, remote of appliances)[2]. The state changes of the various devices were modelled according to Markov Chains[7].



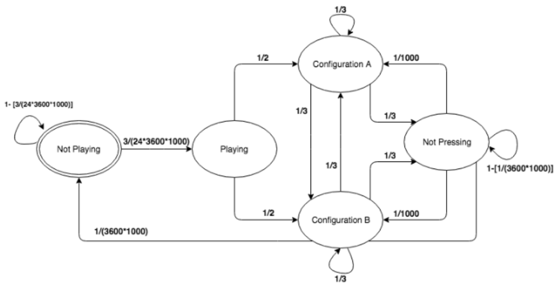
(A) Temperature Sensor model



(B) Presence Sensor model



(C) TV Remote model



(D) Joystick model

Figures 2.2a to 2.2d, taken from [2], depicts the transition probabilities of the various states a device could be in. The state change is based on time. Taking fig 2.2a for instance, it was found that a temperature sensor produces new data three times in an hour hence the various probabilities with respect to time in milliseconds. There are two states in with a temperature sensor and with a probability of $3/(3600 * 1000)$ it changes from, lets say, state X to Y or vice versa and with $1 - 3/(3600 * 1000)$ stays in its state.

Markov Chain with a transition probability, say T, and an initial state probability vector distribution q. The probability of the chain being in state S_i after n steps is the i^{th} entry in the vector given by

$$q^n = q * (T)^n \tag{2.1}$$

To simulate the state change based on equation 2.1 and also, taking into account the discrete time with which the changes occur. The next state vector probability was computed with the n^{th} step being the current time stamp of the simulation.

For each tag-augmented sensor node, when a packet is received, the header is checked to determine the type of packet. If the packet is that of broadcast, a broadcast reply is automatically sent to the reader with a payload of the current

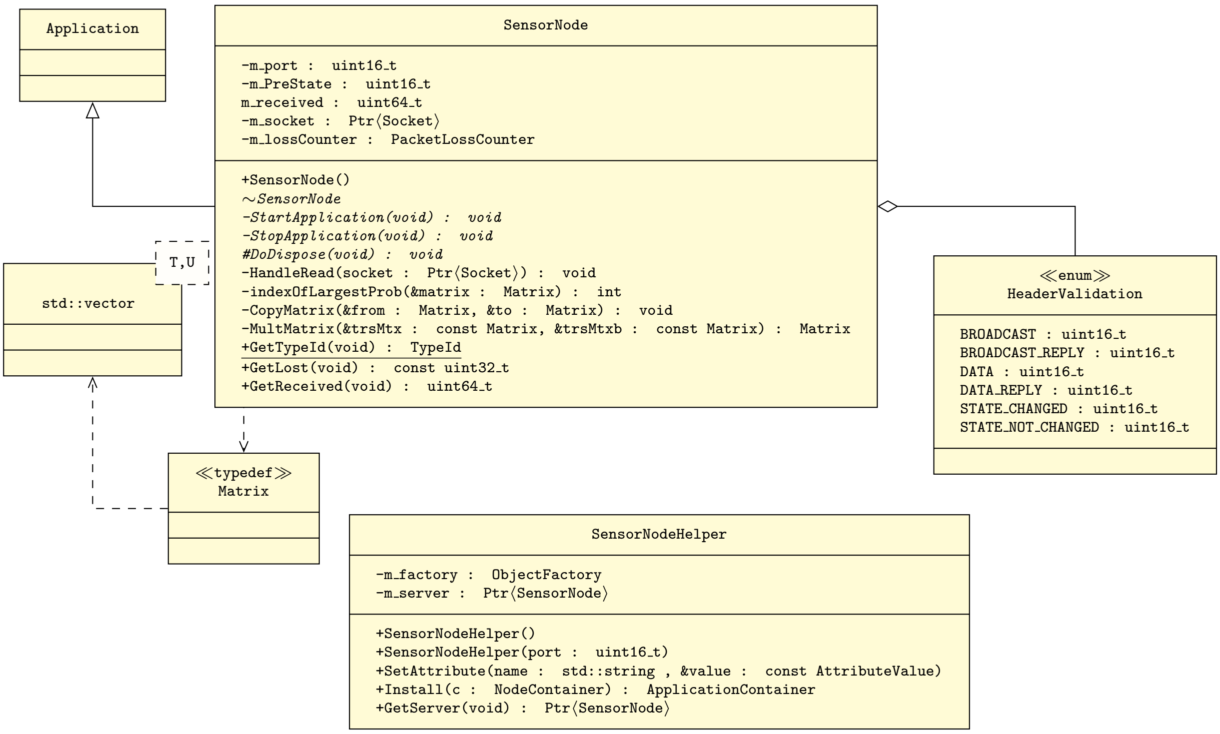


FIGURE 2.3: Class Diagram - Sensor Nodes

data. A data request packet require the sensor node to reply with data. A current state vector probability is calculated and the index with the maximum value is the state of the sensor node. If the computed current state is the same as the previous state, a *state-not-changed* flag is set in the packet header, data reply flag is also set and packet sent to the reader.

The difference in the various sensors is their transition matrix and the initial state vector, as shown in 2.2a - 2.2d, hence the one class for all the sensor nodes.

Class diagrams of the *SensorNode* class and its *helper* class are shown in figure 2.3. A brief explanation regarding the tasks of the essential functions of the classes follows.

> **StartApplication:** the first function run to start the application. It sets up a socket, binds it to a local address and configures the function to handle a received packet.

> **StopApplication:** stops the application by resetting the packet-received call back function and closing the socket.

HandleRead: this is the packet-received call back function. It is invoked when a new packet arrives in the socket or when there is data/packet to be read in the socket file. The type of packet received is determined by examining the header. A broadcast-reply is sent back to the reader if a broadcast packet is received by setting the broadcast-reply flag in the header. For a data-request packet, the current state of the sensor node is retrieved from the *NextState* function and compared with the previous state. If the states (previous and current) are equal, data-reply and state-not-changed flags are set in the packet header and if the states differ, state-changed and data-reply flags are set. The packet is then sent to the reader.

NextState: the state of the sensor node is computed by this function which takes the current time stamp as a parameter. *Next-State* is found by multiplying previous state vector probability with the time-stamp square of the transition probability matrix gotten from multiplying the transition matrix time-stamp times, hence the need for a matrix multiplication and copy functions.

SensorNodeHelper class has functions that aid in the installation of sensors onto nodes; *Install* function, setting the attributes that were declared in *GetTypeId* function of the *SensorNode* class with *SetAttribute* which takes the name of the attributes and the value to be set as parameters. There are, of course, the constructors and a function that return a pointer to the associated server.

2.3.3 Querier


Application
Querier
-m_sendEvent : ns3::EventId
-m_bonus : double
-m_malus : double
-m_learningRate : double
-m_nodeAddress : std::vector⟨ns3::Address⟩
-m_reward : std::vector⟨double⟩
-m_sReward : std::vector⟨double⟩
-m_peerPort : uint16_t
-m_peerAddress : ns3::Address
-m_socket : Ptr⟨Socket⟩
-m_sent : uint32_t
+Querier()
∼Querier()
+GetTypeId(void) : ns3::TypeId
+SetRemote(ip : ns3::Address , port : uint16_t) : void
+SetRemote(ip : ns3::Address) : void
#DoDispose(void) : void
-StartApplication(void) : void
-StopApplication(void) : void
-Send(void) : void
-HandleRead(socket : Ptr⟨ns3::Socket⟩) : void
-UpdateReward(reward : std::vector⟨double⟩ , bonusORmalus : double , index : int) : std::vector⟨double⟩
-SoftMax(reward : std::vector⟨double⟩)
QuerierHelper
-m_factory : ObjectFactory
+QuerierHelper()
+QuerierHelper(ip : Address)
+QuerierHelper(ip : Address , port : uint16_t)
+SetAttribute(name : std::string, &value : AttributeValue const)
+Install(c : NodeContainer) : ApplicationContainer
≪enum≫
HeaderValidation
BROADCAST : uint16_t
BROADCAST_REPLY : uint16_t
DATA : uint16_t
DATA_REPLY : uint16_t
STATE_CHANGED : uint16_t
STATE_NOT_CHANGED : uint16_t

Bibliography

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, An Introduction.* The MIT Press, Cambridge, Massachusetts London, England, second edition, 2018.

[2] Gaia Maselli, Mauro Piva, and John A. Stankovic. Adaptive communication for battery-free devices in smart homes. *IEEE Internet of Things*, 2019.

[3] M. Aykut Yigitel, Ozlem Durmaz, and Cem Ersoy. Qos-aware mac protocols for wireless sensor networks: A survey. *Computer Networks - Elsevier*, February 2011.

[4] Pijus Kumar Pal and Punyasha Chatterjee. International journal of emerging technology and advanced engineering. *International Journal of Emerging Technology and Advanced Engineering*, 4, June 2014. URL http://www.ijetae.com.

[5] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks.* Prentice Hall, Boston Columbus Indianapolis New York San Francisco Upper Saddle River, fifth edition, 2011.

[6] NS-3 Project. Ns-3: Conceptual overview. URL https://www.nsnam.org/docs/release/3.30/tutorial/html/conceptual-overview.html.

[7] Anders Tolver. *An Introduction to Markov Chains - Lecture Notes for Stochastic Processes.* Department of Mathematical Sciences, University of Copenhagen, Universitetsparken 5, DK-2100 Copenhagen Ø, Denmark, November 2016.