7-2021

Belief Networks as Approximated Models for Testing the Design in Production

Timothy James Atkinson

Belief Networks as Approximated Models for Testing the Design in Production

by

Timothy James Atkinson

Master of Science
Computer Science
Florida Institute of Technology
2007

Bachelor of Science
Computer Science
Florida Institute of Technology
2003

A dissertation
submitted to the College of Engineering and Science
at Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Doctorate of Philosophy
in
Computer Science

Melbourne, Florida
July, 2021

We the undersigned committee
hereby approve the attached dissertation

Belief Networks as Approximated Models for Testing the Design in Production

by Timothy James Atkinson

Marius C. Silaghi, Ph.D.
Professor
Computer Engineering and Sciences
Committee Chair

Eugene Dshalalow, Ph.D.
Professor
Mathematics
Outside Committee Member

Debasis Mitra, Ph. D.
Professor
Computer Engineering and Sciences
Committee Member

Lucas Stephane, Ph.D.
Senior Researcher
Institute for Energy Technology
Committee Member

Philip Bernhard, Ph.D.
Associate Professor and Department Head
Computer Engineering and Sciences

ABSTRACT

Title:

Belief Networks as Approximated Models for Testing the Design in Production

Author:

Timothy James Atkinson

Major Advisor:

Marius C. Silaghi, Ph.D.

A software design methodology is proposed involving the development of approximate models based on reputation systems and Bayesian Networks capturing probabilistic representations of expected behavior, which are further used in developing and running tests that can dynamically diagnose bugs and attacks during production. While automation of the Software Design itself is still a very remote goal, it can already benefit from AI tools and ideas. One of the main challenges with automating software design methods, for any product with modest complexity, is the mere intractability of enumerating all requirements of the product usage, when also taking into account all (including malicious) user intentions, leading to the intractability of generating exact design specifications and exhaustive tests.

We show how approximate models of the design can exploit AI techniques to represent the system sufficiently well to derive meaningful tests, warning when the environment is not behaving as designed and detecting both bugs and attacks. We also show how the Bayesian models can be converted into reputation models.

We validate the proposed methodologies with two different applications: a de-

vice driver for Wi-Fi Direct, and a website, MindBlog.com.

In the Wi-Fi Direct use case, we successfully build simple classic Bayesian networks using expert knowledge, further transforming some Bayesian networks into reputation networks. In the MindBlog.com use case, we show that the procedure is flexible and can even detect when the developers found bugs and were attempting to debug their application yielding anomalous behavior.

Our methodology not only extends comparative A/B testing by allowing the engineer to know what design element did not match the user expectations; but also informs the engineer in real time when the user's expectations have changed away from the original design. Thus it enables the engineer to be proactive in addressing such user expectation changes. Furthermore, by using Bayesian networks we can focus on only the design elements that give use concern helping both with the communication of high level desired properties and with testing of deviations.

Table of Contents

List of Figures

List of Tables

List of Symbols

AR	Above Range
AI	Artificial Intelligence
AP	Access Point
BR	Below Range
DoS	Denial of Service
DVL	Domain-specific Visual Language
FFBD	False Friend Battery Attack
FPR	False Positive Rate
GO	Group Owner
IE	Information Element
IFML	Information Flow Modeling Language
IoT	Internet of Things
INCOSE	International Council on Systems Engineering
IV	Intent Value
MAC	Media Access Control Address
MBE	Model Based Engineering
MBSE	Model Based System Engineering

MDD	Model Driven Design
MDE	Model Driven Engineering
LTE	Long Term Evolution
OCL	Object Constraint Language
OUI	Organizationally Unique Identifier
R	In Range
SysML	System Modeling Language
TBB	Tie-Breaker Bit
UML	Unified Modeling Language
WAR	Well Above Range
WBR	Well Below Range

Chapter 1

Introduction

A method of designing complex software suitable to embed as a form of test into the final product is proposed based on approximate models employing Bayesian probabilistic networks and reputation systems. Common software design methods aim at exact specifications that may or may not be directly executable and face known challenges with complex systems due to intractability of full requirement specifications, are particularly well known when referring to distributed, cloud, or Internet of Things (IoT) heterogeneous systems. Statistical models based on Bayesian Networks offer a chance to focus on essential components of the design while ignoring implementation details, expressing the expert knowledge of relationships between important features that represent the desires of the developers and customers, and exposing them for tests.

Traditionally, model based engineering looks at models from two perspectives. Besides the correctness/completeness of the model, it also validates that source code matches the model either by generating the source code or generating tests to run against the source code [55]. Our approach however only focuses on how well current component behavior matches design models.

Bayesian networks provide a principled mechanism for dealing with uncertainty while taming complexity. Software engineering develops techniques for taking large numbers of requirements, and managing the life-cycle of an application from initial concept to development to retirement. The software development life-cycle starts with encapsulating requirements into a design that is used to direct the remaining steps like development and test. Therefore the use of Bayesian networks for helping encapsulating complex requirements as part of the software development life-cycle is a natural development in software engineering.

This section summarizes: motivation, approach, evaluation, and contribution.

1.1 Motivation

Investigations into reducing costs through tests that raise real-time awareness of bugs and attacks point towards solutions based on efficient management of approximations. This is due to impracticality of approaches based on pure logic rules due to the sheer size of the requirements in complex systems. Management of approximations is commonly addressed in artificial intelligence research.

In some software engineering communities there is a debate as to how much modeling is the appropriate amount. Some believe that modeling should continue until the source code can be generated from the model – even creating new modeling languages if necessary [26]. Ignoring scaling issues, this also leads to a training problem as new engineers must first learn the modelling language before they can work on the problem at hand.

An alternative is to focus on what makes models so powerful in the first place: abstracting out the details and focusing on the most important aspects of the problem for a given test or audience.

Bayesian networks are a statistical technique where the modeler models the problem with directed acyclic graphs to graphically show the conditional dependencies between variables. Each node within the graph represents a variable whose value distribution is described probabilistically given its dependencies [43]. They are the ideal modeling tool for the problem at hand as they contain a strong mathematical foundation, show the relationships between variables making easy explanation for any decisions made, and most importantly non-represented elements, left off either intentionally or accidentally, become noise to the variable being modeled.

1.1.1 Use Cases

We decided to focus on two use cases for our approach described below.

Devices in a smart home may come from different vendors and have to communicate for achieving higher-level tasks such as regulating light and sound levels in larger spaces or regulating power, saving the homeowner on utility bills. For example, devices from some vendors may exploit devices of a competitor by coaxing them into taking a service function in a disproportionate way with the intention to deplete their batteries faster. This will cause a negative impact on the user's perception about those competitors. More dangerously, a device that has been taken over by an attacker can be used as a proxy in an attack on the batteries of a more security-sensitive device, such as a door-keeper, by similarly

manipulating it.

As a second example, at the time this paper was written, there were approximately 1.8 billion websites connected to the Internet [52]. The owners of websites have an additional problem other engineering disciplines do not have so acutely: potential new customers can abandon the website without ever leaving a trace as to why they have quit. One potential reason a new customer might leave is due to the interface being frustrating to use. By modeling normal values of the communication framework, we can help the website designer understand better their own design and help them detect when a user is struggling to use their website.

1.2 Software Modelling Technology

A summary of the main relevant concepts in software modelling approaches is introduced. A set of closely related such concepts are Model Based Engineering (MBE), Model Driven Engineering (MDE), and Model Driven Design (MDD). While industry practitioners use the terms with overlapping and sometimes contradictory meanings, we will use the definitions listed below [1].

Definition 1 (MBE) *Model Based Engineering (MBE) is a practice where models are used to inform software or system design.*

Definition 2 (MDE) *Model Driven Engineering (MDE), a subset of MBE, is a practice where models are used to inform test and maintenance.*

Definition 3 (MDD) *Model Driven Design (MDD), a subset of MBE, is a practice where models are used to generate code.*

Another closely related definition used by International Council on Systems Engineering (INCOSE) is Model Based System Engineering (MBSE) [8]:

Definition 4 (MBSE) *Model Based System Engineering (MBSE), is an approach to engineering that uses models as an integral part of the technical baseline that includes the requirements, analysis, design, implementation, and verification of a capability, system, and/or product throughout the acquisition life cycle.*

Software modeling has two dominant modeling languages: Unified Modeling Language (UML) and SysML (System Modeling Language). UML came out of combining the top three modeling languages that preceded it: Object-Oriented Analysis and Design, Object-Oriented Software Engineering approach, and the Object Modeling Technique [47]. SysML primarily targets modeling the system as a whole rather than targeting just the software. As such they eliminated some diagrams that were too specific to software, added requirement and parametric diagrams, and added hardware considerations to the diagrams allowing for the entire system to be considered and not just the software [27].

Once a problem is described in a model, a common operation is to convert from one model to another. One such transformation was from an extension to SysML, called SysML-sec, an extension focused on security concerns, to a ProVerif model which looks for vulnerabilities in the proposed architecture while assuming each crytographic primitive is a black box [5]. However, powerful symbolic modelers such as ProVerif are not perfect – nor will they ever be. For example, in a case study of the ARINC823 protocol, an attack against the shared-key protocol did not appear using the ProVerif model [16]. In the case

of the corresponding study, they were able to detect the vulnerability with a different model checker.

In safety critical development, it is important to trace each piece of source code to the requirement that produced it in order to ensure that all code has been thought about in terms of the overall design and has sufficient testing. One method to ensure such traceability is to have the model generate the corresponding source code. A recent example of this approach, the multifunction vehicle bus controller was modeled using Uppaal. After analyzing the protocol and finding a specification error, they generated source code from their Uppaal model [32]. However, their particular process was slow, it took around a minute for nine requirements. Furthermore, there was no verification that the generated source code was implemented correctly or that the OS which the software runs is free of defects – only that the generated source was traceable to the model.

Another approach for ensuring that the system matches the model is to produce tests from the model. For example, one can convert unstructured activity diagrams into test scenarios [57]. Or one can extend the SysML metalanguage to include Media Access Controller (MAC) addressing allowing a neural network to be trained for normal operations and linking attacks to the components of the diagram providing realtime insight to what might be affected [14]. One promising technique was to map failure modes into Bayesian networks and add them to the SysML model [38]. However, their work stopped short of the true potential of Bayesian networks by focusing on the hardware components and ensuring all elements of the Bayesian network were described in SysML, which weakened the potential of the Bayesian network.

Similar to [38], we look to a real time detection of faults; however, we design

Bayesian networks as a new type of test that is run in production which allows the engineer to know if users are employing the software consistent with how it was designed.

1.3 Approach

The approach we take in this work is focused on modeling, in a machine interpretable representation, the expected value of various signals (monitored input and/or output variables) of the unit under test (which can be the system at large or an individual module within the system). This representation can subsequently be used to test the system's function during execution.

The representation we favor is based on Bayesian Networks. Bayesian Networks are a commonly used probabilistic knowledge representation aimed at passing the mini-Turing test – the computer can reason about the facts of a counter-factual by merely reassessing the hidden states of model given the counter-factual [44]. Its strength comes from its ability to support evaluation of actual data for conformance, as well as for diagnosis.

As the signals we are working with can be noisy, a sliding window of arbitrary size is chosen per signal. Furthermore, the thresholds for all enumerations and when to declare an anomaly are also chosen arbitrarily and should be selected carefully based on the problem being modeled and the expected noise within the system.

We propose the following procedure for converting elements from the design into Bayesian networks. For each which is an interesting element of the design:

- Place all data within the sliding window

- Decide how to discretize the data within the sliding window to an enumeration: well below range, below range, in range, above range, and well above range
- Decide how many points are required before a partial decision can be made, and when a full decision can be made
- Either take this signal by itself, or group this signal with other signals and ask the Bayesian network: "What is the probability this signal(s) is normal?"
- Take action based on probability.

1.4 Contributions

The following are our contributions in the domains o the studied used cases as well as in the general approach:

- Using a Bayesian Network to detect the misuse of the Tie Breaker Bit (TBB) bit during Group Owner (GO) negotiations.
- Using secure bit commitment to enforce that the TBB bit has been randomly set.
- Demonstration the general approach can be applied to a different domain by targeting the user interface of a website
- Introduction of reputation based Bayesian Networks allowing devices that partially cheat to still be caught

- Using Bayesian Networks as a testing technique that can be used in production

The rest of of the paper is structured as follows: Related works will sample through similar research which attempts to tackle similar problems to our own, Learning Strategies for Resisting Power Attacks will examine our early investigations into the TBB during GO negotiations, Approximate Belief Network Models for Testing will look into generalizing and characterizing our work, Reputation Bayesian Networks will introduce a technique to smooth out the noise in our earlier work, finally we shall conclude the dissertation.

Chapter 2

Related Work

Artificial Intelligence and Software Engineering recent work related to our efforts is described, showing that the trends in the community points in the direction of our contribution.

2.1 SysML and Bayesian Networks

In the paper, "Mapping SysML Diagrams Into Bayesian Networks: A Systems Engineering Approach for Fault Diagnosis", Melani and Souza describe a new technique for using Bayesian Networks to detect system-level faults by extending SysML with some new keywords and a procedure to convert the SysML diagrams into Bayesian Networks [38]. This section reviews their study.

Let us consider a long endurance, competitive, amateur cyclist who wishes to use monitoring technology on non-qualifying races to help the cyclist know if she should quit the race before a major injury occurs. All races are multi-day events with occasional service stops (snacks, sometimes mandatory weight checks, water, etc...). The cyclist has targets for all sensor readings.

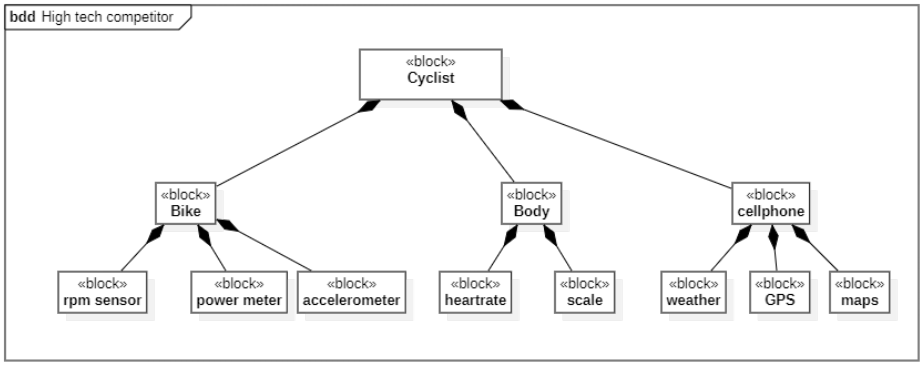

Figure 2.1: Sample block definition diagram

Standard SysML has nine diagram types: activity, block definition, internal block, package, parametric, requirement, sequence, state machine, and use case. One of the main drawbacks to SysML is that each diagram type can and should reference other diagram types so that there are many views into the system where each diagram type is better at expressing some aspect of the system compared to the other diagram types.

The block diagram is SysML's equivalent of UML's class diagram and describes the static structure of the system. The system block diagram is the first diagram that must be developed because it identifies all components of the system and helps limit the scope of where faults might occur. In our example case, we have a collection of sensors the cyclist can use to get an indication as to how they are doing from rpm sensors to heart rate sensors. An example block definition diagram is presented in Figure 2.1.

Once the system block diagram has been constructed, a state machine diagram containing all the states the system can be in, needs to be constructed, as each system state could provide a unique explanation for a particular sensor

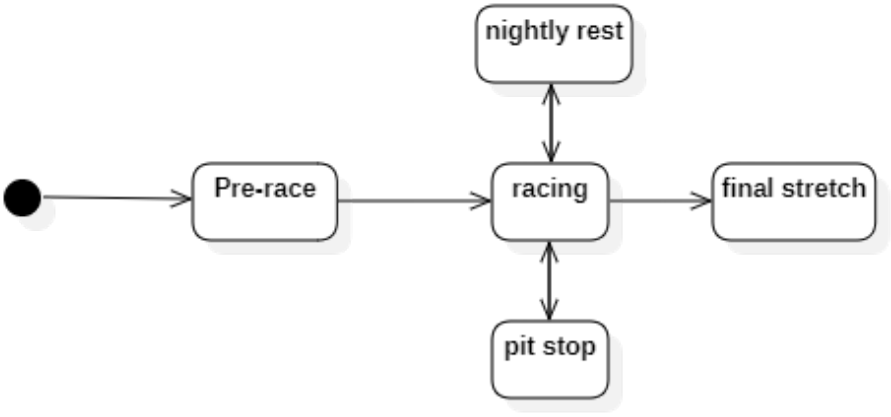

Figure 2.2: Sample system state diagram

reading. Each system state will result in a new Bayesian Network diagram as different system states can dramatically change the behavior of the equipment and therefore the readings of the sensor. In our example, as seen in Figure 2.2, we have four states: pre-race, racing, pit stop, nightly rest, the final stretch as the cyclist may want to push extra hard in those final monuments knowing the race is about to be over. While not drawn to avoid clutter, the cyclist can quit while in any state.

A state diagram also needs to be constructed for each sensor; however the example provided in the paper is pretty standard where the sensor is either reading above normal, below normal, as expected, or fluctuating. In the original study, the author proposed a sensor with four states in a fully connected graph: higher than expected, lower than expected, expected, and intermittent/inactive. We keep the same state machine for our example.

Finally an activity diagram showing all the components and activities and a modified block diagram showing the off-nominal behavior is needed to finish

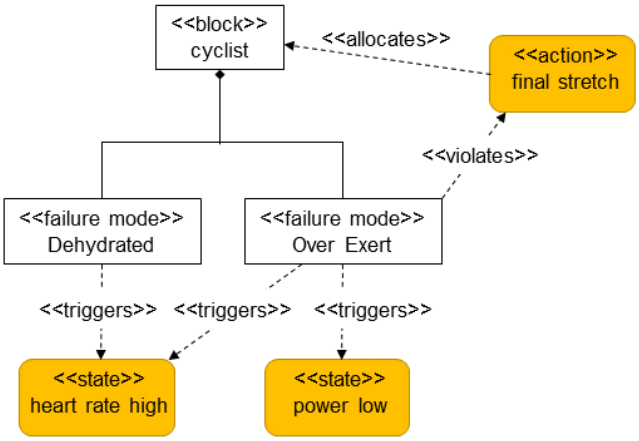

Figure 2.3: Example Off Nominal Diagram

off the required diagrams. In our example, we only have one component which takes actions: the cyclist; therefore, our activity diagram looks identical to our state diagram as shown in Figure 2.2.

As shown in Figure 2.3, we consider two failure modes: dehydrated and over exerted. To show that a high heart rate is indicative of dehydration, the keyword triggers is added to the dependency. The over exerted failure mode impacts how well the cyclist can perform the final stretch which is indicated with the 'violates' keyword.

Now that all the diagrams have been constructed, the input knowledge needed to construct the Bayesian network is present. The off nominal diagram is converted into a Bayesian network by considering each failure mode and each sensor is a node in the Bayesian network. The triggers relationship is the casual relationship as seen in Figure 2.4. An expert is expected to fill out the CPT for each node.

Melani and Souza offer a promising technique to map the system model to a causal relationship model of system failures to sensor readings. They have

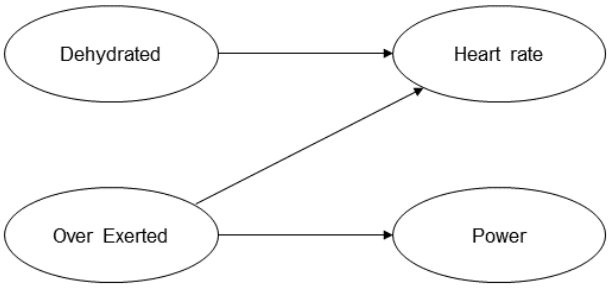

Figure 2.4: Example Off Nominal Bayesian Network

demonstrated that their technique is general by providing two case studies focused on self-contained mechanical systems with the corresponding SysML and the generated Bayesian network. However, because their technique maps all SysML elements to Bayesian nodes one-to-one, their technique struggles with non-fault explanations for abnormal sensor readings. For example, a cyclist may not have enough data to interpret the sensor readings and receive false positives in early monitoring of a particular system state. Or the cyclist may encounter wind, hills, drafting, or any combination of all three – none of which are faults and can affect the posterior distribution of the sensor readings. To use their technique with this additional stressors would require adding an additional 2^4 system states and this does not even count potential malicious actors who may wish to subvert the cyclist through cyber-attacks. As these additional states come from outside the system and are therefore not implementable, adding these additional states for the purposes of fault diagnosis will complicate implementation of the model as developers will need to be told to ignore those states.

Our technique, in contrast, treats Bayesian networks as a causal relationship model distinct from the other diagram types found in either SysML or UML

worthy of expert debate to their correctness. It can cover highly abstract properties. Furthermore, as new attacks are discovered, the casual model can be updated directly without needing to update the system model.

2.2 Domain-specific Visual Models

One method for making models specific enough to generate the source code is to create a modeling language specific to the problem [26]. As each meta-requirement, requirements on what kinds of requirements need to be collected, are understood; a Domain-specific Visual Language (DVL) can be created to help visually show the relationship between the requirements.

As the DVL is specific to the domain being modeled, source code can be generated directly from the model. However, as DVL is specific to the domain being modeled, to use this approach in industry, each company would need to have compiler experts on staff to write the model translation layer. This leads to two sources of defects that can escape to production: the model has a defect or the compiler has a defect [20]. To counter those concerns, the authors proposed future work for an improved defect reporting system.

While generating code from a model has many advantages such as ensuring the same requirement is implemented the same way across the system allowing a defect found and fixed in one location to be fixed in all other locations automatically; generating source code from the model limits testing to users self-reporting defects.

However, many users may simply go to a competitors product rather than report an issue they had. Furthermore, malicious users would prefer if defects

were left unresolved so they can continue to exploit them and will never self-report.

2.3 Reputation Systems

Reputation systems can be split into two main categories: centralized and decentralized. In a centralized reputation systems, like Amazon or eBay, the reputation of participants is stored in a central location. Whereas, in a decentralized reputation system there is no central authority for participants to query. As such they must either calculate reputation scores based on their own interactions with various participants or receive advice from other participants.

As a relevant example, the Beta reputation gains its name by converting positive and negative ratings into a beta distribution [29]. The beta distribution takes two parameters, α and β, and produce a probability that the observed outcome will occur again. They assigned $\alpha = p + 1$ where p is the number of positive ratings and assigned $\beta = n + 1$ where n is the number of negative ratings. The resulting 'probability' is scaled to a number between -1 and 1 where -1 represents a non-reputable participant and 1 represents a reputable participant. Their study considered history discounting of old votes so that if an agent changes from honest to dishonest from dishonest to honest the system will remain responsive, combining advice from external peers, and discounting reputation scores from neighbors based on their reputation.

Reputation systems can help protect against pollution attacks in peer-to-peer networks. A pollution attack occurs if unauthorized or unauthenticated information is inserted into the segments. To complicate detection of pollution

attacks, malicious actors will often combine the pollution attack with a collusion attack by having multiple malicious users up vote each other and down vote regular users. DRank is a reputation system which used two beta functions one for the reputation and one for trust [53]. They add a beta distribution for trust where the expected value from shared reputation is compared to the current reputation of the participant under consideration. If the difference between the two reputation scores are below a threshold they increase the trust in the participant; otherwise they decrease the trust in the participant. If the trust rating falls too far, they blacklist the participant from sharing their reputation information.

A recent work used a similar technique with similar results to our earlier results in [36]; however, they applied the technique to vehicle networks [13]. Their experiments show yet another domain our techniques can work. Their application made several assumptions we did not make: they only ran their simulation for 100 seconds rather than for the life of an object, honest nodes only appeared to be dishonest if the router was full, malicious attackers send packets at fixed intervals, all traffic was constant bit rate. Their algorithm attempted to adjust the number of false positives by establishing a lambda variable which adjusted the threshold a malicious user was declared.

We consider a decentralized model where elimination of false positives is impossible due to the random nature of the variable we are tracking. Rather than looking to the Beta distribution and tying the reputation to positive votes and negative votes; we instead look to a causal model where we feed the posterior distribution into the prior distribution.

2.4 Denial of Service attacks

Denial of Service (DoS) attacks are a type of attack where a service is prevented from working by attacking the infrastructure around the service. For example, one can attack a service by flooding it with multiple service requests. There are several different methods to perform a DoS attack against a wireless network, and this list is not exhaustive.

Jamming is an obvious method to deny wireless devices access to the network. Further, if the device is battery-powered, the victim's battery can be rapidly drained as it attempts to overcome the jammer. While challenging to defend against, one method is to sleep and test the channel periodically [17].

Wormhole attacks affect ad hoc wireless networks when two malicious nodes collude by creating what appears like a shorter distance connection within the network. This connection can then be used to institute a man-in-the-middle attack or a denial of service attack by dropping all messages.

Amish et al. introduced the idea of modifying the Ad-hoc On-Demand Multipath DiscoVery (AOMDV) protocol to time the duration that discovery messages take for every other node within the network. When a particular route has an abnormal timing, a fake discovery message can be sent through the wormhole while a message is sent via an alternative path informing the victim, thus disabling the wormhole [6].

A carousel attack primarily affects ad-hoc networks where loops are introduced explicitly into the message's routing to force nodes to route the same message multiple times, draining their batteries. A simple no-cost defense against carousel attacks is for the sending node to search for itself from the end of the

routing list instead of the beginning [17].

The battery exhaustion attack is a classic DoS attack that attacks the battery of the device rather than the applications or network of the device [50, 37]. According to [37] there are three different classes of these attacks on battery-powered devices: (1) malignant attacks, in which a virus can be used to make the device consume significant power, (2) benign attacks, any program can consume energy very fast, and (3) flooding service requests attacks, in which repeated requests are made to a particular device. Another effective depletion-of-battery attack is to only partially jam a node forcing it to increase its power (which increases its overall QoS due to the increased power) which depletes the battery faster [50].

To counter the depletion-of-battery attacks, [48] showed how to map CPU utilization to estimate battery depletion rate for malicious applications. It, however, does not protect the device from network or system attacks. Battery exhaustion attacks have been studied in Wireless Sensors Networks and Internet of Things [45].

Chapter 3

Learning Strategies for Resisting Power Attacks on Wi-Fi Direct Group Formation

The Wi-Fi Direct standard was introduced with the intention of simplifying peer-to-peer connections in home applications while helping devices to save power by centralizing effort into a single group owner device negotiated on start-up. Attacks on the group formation stage can be based on manipulating a victim device to have it frequently end up being assigned the group owner function, thereby depleting its batteries at faster rates than its peer devices. This manipulation is made easy by the group formation process adopted by the standard. We propose to enhance the group formation process with secure features ensuring fairness by relying on commitments and learning about the behavior observed for peer devices in the past.

The 802.11 standard allows vendors to extend the specification through their Information Element (IE) Frame. The IE has a general format consisting of a

one-octet element ID field, a one-byte length field, three bytes Organizationally Unique Identifier (OUI) field, one-byte OUI type, and up to 251 bytes of payload [28]. Wi-Fi Direct defines the P2P IE Frame using attributes composed of a one-byte attribute ID, two-bytes attribute length, and attribute data. Multiple attributes can be placed within a single IE Frame, and compliant vendors are required to ignore attribute IDs they do not understand (including attribute IDs marked as reserved in the current standard). New attributes can be used for extensions as proposed here.

This protocol allows communication among peers within a single group [2]. The roles of Group Owners (GOs) and clients are defined during the group formation process. The P2P GO implements an AP-like functionality [4], with maintenance and advertisement performed through periodically sent beacon frames, increasing its power consumption. After the election process, the role of GO or client remains unchanged during the entire group session. When the GO owner, for any reason, leaves the group, the devices become disconnected.

There are three methods for creating a group: standard, persistent, and autonomous. However, our focus in this paper is only on the standard group formation method.

3.1 Standard Group formation

The device discovery phase consists of two sub-phases: scan and find. During the scan phase, devices try to find other devices/groups or Wi-Fi networks while locating the best channels to establish a group. Devices can not reply to request frames when they are also scanning. During the find phase, the device selects

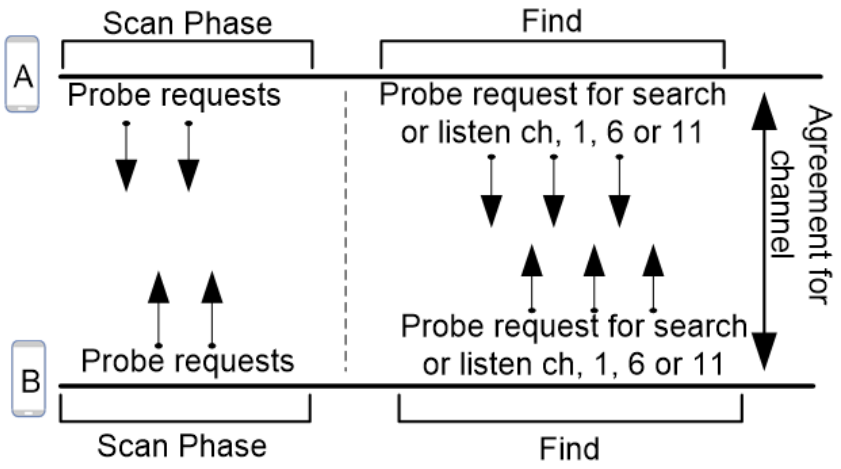

Figure 3.1: P2P Device discovery process

one of the channels 1, 6, or 11 in the 2.4 GHz or 5Ghz bands as the listening channel. After this, the device alternates between a scanning phase by sending Probe Requests in each of these channels and a listen state to which it listens on a channel for a probe requests frame. The discovery phase algorithm is illustrated in Figure 3.1.

The Group Owner Negotiation phase starts once a device has found another device it wants to connect with that is not yet involved in a group The details regarding this three-way handshake communication are illustrated in Figure 3.2. When the target device is already in a group, the group owner is contacted, skipping the GO negotiation phase.

3.2 Power Consumption

Energy consumption is one of the main requirements for IoT device-to-device communication as such it has been the focus of recent scientific works [54, 33,

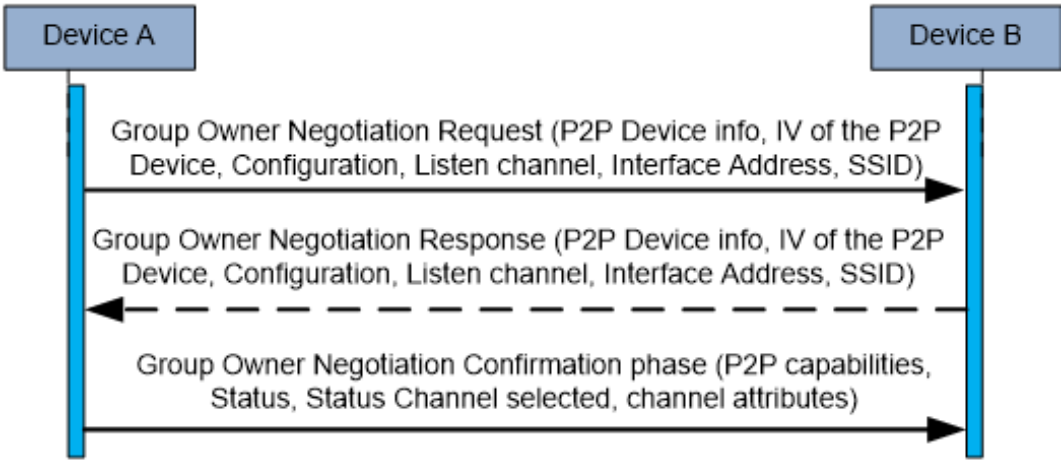

Figure 3.2: Group owner negotiation process

9, 18, 23].

Since the P2P GO might be battery powered, its energy consumption is critical [39, 58]. The P2P power management protocol defines an availability period, called Client Traffic Window (CTWindow) during which a P2P GO is present. Power management for GO consists of delivery mechanisms defined for Power Save and Wi-Fi multimedia power save (WMM-PS), but there also exist power saving mechanism that allow for a GO to be sometimes absent: the Opportunistic Power Save (OPS), and the Notice of Absence (NoA) [2].

Definition *A False Friends Battery Depletion (FFBD) attack is when a victim device is intentionally induced into depleting its battery by volunteering services.*

Gracefully detecting and handling an FFBD attack requires complex reasoning or a robust classifier. Designing a set of logic rules leading to a rigid classification of a peer as an attacker is possible but fragile due to uncertainty. A probabilistic classifier implemented as a Bayesian Network or an Artificial

Neural Network can more robustly be used to associate an FFBD attack to a posterior probability, given a history of communication.

Once an attack probability is associated with each peer profile, this can be communicated to the device owner (or manufacturer). Further, automatic acceptance or rejection of a connection as GO with a peer can be made using a utilitarian approach, based on the current batteries' levels, a fairness measure, and a parameter of risk adversity that can be configured.

We propose to consider that a GO device has **prematurely quit** a group, as an attack strategy, if it either quits before the end of a group formation negotiation where it would end up as GO, or it quits before the group can be used to perform any concrete application-level task.

3.3 Approach

Let us now describe the investigated solution based on a Bayesian Network. The module is assumed to maintain a profile for each peer with which it creates groups. The profile stores statistics used as features of the learning process. These statistics can be maintained over a sliding time window (a month, in our experiments). To maintain the sliding window, the statistics for a time window can be assembled dynamically from smaller buckets (we use one bucket per day), such that for each new bucket added; the oldest bucket is removed from the totals. Identified features, computed by a device D for each peer over the sliding window, are:

- **iGO**: percentage, of GO negotiations with a peer, where D was elected GO. The possible values, as used in our experiments, are: low (V_L) for

0-25%, small (V_S) for 25-40%, average (V_A) for 40-60%, above (V_B) for 60-75%, and high (V_H) for 75-100%.

- **pGO**: percentage, of GO negotiations where the peer is GO, where it quits prematurely, discretized as for iGO.

- **tGO**: The percentage, of communication time with the peer, in which D was GO.

- **Data**: the number of GO negotiations with the peer. The domain of Data is discretized as "insufficient knowledge" (I) below 10 rounds, "some knowledge" (S) below 100 rounds, and "reasonably informed" (R).

The device learning module uses the above features to output peer characterization values:

- False Friend Battery Depletion attack ignorance: the current ignorance of the device as to whether there exists sufficient data to judge whether the peer is an FFBD attacker, in the sense of the Dempster-Shafer theory [49]. It is mapped from the values of the *Data* feature.

- Peer Fairness (PF): how fair has been, so far, the GO time distribution with this peer. It is estimated as the percentage of communication time with the peer in which D was GO, namely tGO. E.g., ratios below .6 are considered fair.

- False Friend Battery Depletion attack (AT) probability: the probability associated with being under an FFBD attack from the peer, estimated using the Bayesian Network in Figure 3.3, where the thick margin circles

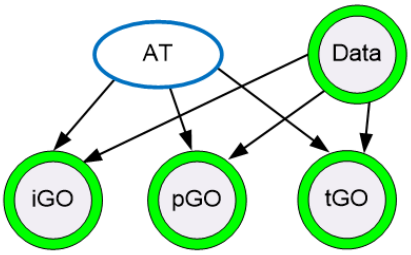

Figure 3.3: Bayesian Network.

are evidence variables, and the conditional probability tables (CPT) are shown in Table 3.1.

FFBD attacks can be at different intensity. The attacker models represented by the multi-value random variable AT, are classified as (see Table 3.1):

- "Strong attacker" (A_{SA}), where the percentage of negotiations where peer attacks is high (V_H)
- "Medium attacker" (A_{MA}), where the percentage of negotiations where peer attacks is above average (V_B)
- "Fair" (A_F), where the percentage of negotiations where peer attacks is average (V_A)
- "Better than average" (A_{BA}), where the percentage of negotiations where peer attacks is small (V_S)
- "Altruist" (A_{AL}), where the percentage of negotiations where peer attacks is very low (V_L)

The prior probability of other devices being attackers can depend on the manufacturer of the other devices. In our experiments, this multivalued random

	AT	P(V_L)	P(V_S)	P(V_A)	P(V_B)	P(V_H)
Data=R	A_{SA}	0.05	0.1	0.2	0.2	0.45
	A_{MA}	0.05	0.1	0.2	0.45	0.2
	A_F	0.1	0.2	0.4	0.2	0.1
	A_{BA}	0.2	0.4	0.2	0.1	0.1
	A_{AL}	0.4	0.2	0.2	0.1	0.1
	AT	P(V_L)	P(V_S)	P(V_A)	P(V_B)	P(V_H)
Data=S	A_{SA}	0.14	0.14	0.14	0.22	0.36
	A_{MA}	0.13	0.13	0.20	0.34	0.20
	A_F	0.13	0.20	0.34	0.20	0.13
	A_{BA}	0.20	0.34	0.20	0.13	0.13
	A_{AL}	0.36	0.22	0.14	0.14	0.14
	AT	P(V_L)	P(V_S)	P(V_A)	P(V_B)	P(V_H)
Data=I	A_{SA}	0.17	0.17	0.17	0.24	0.25
	A_{MA}	0.15	0.15	0.23	0.24	0.23
	A_F	0.15	0.23	0.24	0.23	0.15
	A_{BA}	0.23	0.24	0.23	0.15	0.15
	A_{AL}	0.25	0.24	0.17	0.17	0.17

Table 3.1: Conditional Probability Tables of the leaf nodes, built as a definition of the attacker types.

variable prior is the vector of 5 values $[0.15, 0.2, 0.45, 0.1, 0.1]$, and in practice can be adjusted based on inputs from the end-used concerning their happiness with the relative battery performance of the device. The FFBD attack probability for a peer is checked each time the device becomes GO, and if $PF > 0.6$, computing:

$$P(AT|iGO, pGO, tGO, Data)$$
$$= \alpha P(iGO, pGO, tGO|AT, Data)P(AT)$$

3.4 Exploiting Commitments

In addressing FFBD attacks, solutions have to consider the trade-off between efficiency and security. While, the introduction of a fair coin-flipping mechanism provides strong evidence of manipulation, it is unable to detect when a peer quits group owner negotiation prematurely. The technique still has value as it prevents blindly grounding the TBB and has small computation requirements.

While the IV value of a device may depend on the peer with whom it communicates, the TBB is supposed to be independent and a commitment to

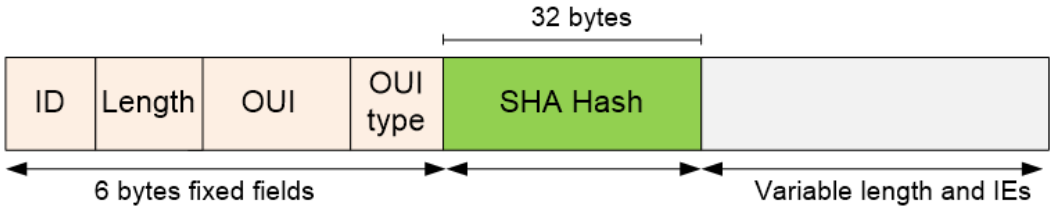

Figure 3.4: Proposed Vendor IE frame format

it can be advertised in Probe Request frames. A tentative IV value can also be advertised with the probe, but it is understood that the device may change the value based on the peer identity. In a version that we propose along with our learning strategies, each peer A of the GO negotiation will commit to its current TBB_A by sending its commitment $H(R_A|IV_A|TBB_A)$ inside a new type of Information Element broadcast with each Probe Request, called Tie Breaker Bit Commitment (TBBC) IE.

The number of bytes of overhead required for a SHA-256 hash is 38, namely 32 bytes for the hash and 6 bytes for the IE header; if the message is implemented as a *Vendor IE* (see Figure 3.4). Instead, the overhead is 35 bytes if the message uses one of the *Reserved P2P attributes*. Further, for compatibility, the GO Negotiation Request may contain a random TBB'_A that is not correlated with the TBB_A committed to. This adds 4 bytes: 3 as overhead for a P2P attribute header and 1 byte of payload for the TBB bit.

The device B deciding the identity of the GO will then use as tie breaking value the result TBB of the computation $TBB = TBB_A \ XOR \ TBB_B$. Then, device B opens its commitment using the GO negotiation response to which it can add a P2P IE attribute for opening the commitment by stating its R_B. It will also communicate the IV_B actually used in the decision. Differences

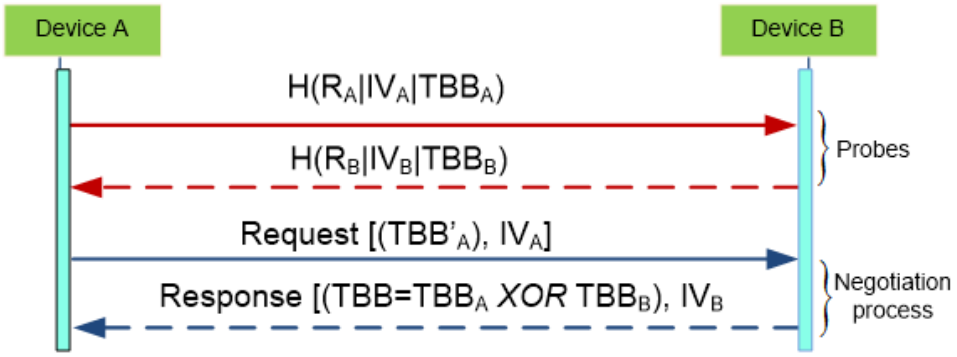

Figure 3.5: Proposed probe and negotiation process

between this IV, IV_B and past values are also reliable features usable by the learning system to build a profile for device B.

An advantage of this solution is that the need of GO negotiation confirmation may be reduced and eventually removed from the protocol, shortening the setup delay in future versions of P2P Direct, which will improve its usability to applications sensible to start-up latency, such as vehicle-to-vehicle communication.

An alternative approach is to integrate the TBB commitment fully in the GO negotiation process. The device A integrates the commitment $H(R_A|IV_A|TBB_A)$ in the GO negotiation request. Further, the device B generates both sets of responses for the cases where a GO is assigned to device A and to device B. Both outcomes are presented in the GO negotiation response, together with TBB_B and IV_B. In the end, the device A returns the GO negotiation confirmation with the selected result, and opening the commitments for TBB_A and IV_A.

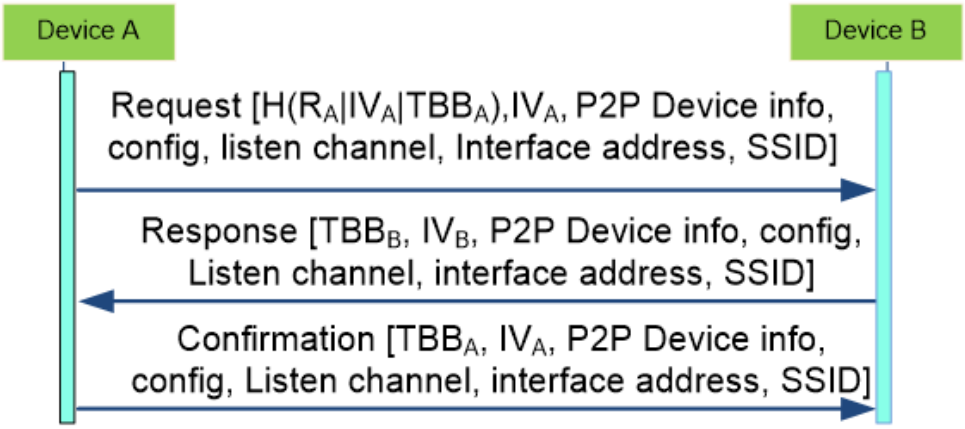

Figure 3.6: Commitment in the GO negotiation

3.5 Experiments

A simulator was implemented and used to run experiments with 2, 5, and 10 devices. Each device is assumed to start with a battery that can last 365 days in the non-communicating mode. The simulation assumes that the non-communicating mode uses one energy unit per second. The client mode uses an additional energy unit per second. The GO mode uses ten additional energy units per second, besides the unit used by the non-communicating mode. The IV is always set to 0 to focus on the tie-breaking algorithm. In all the simulations, the device 0 may be a victim of FFBD attacks by a subset of the remaining devices. The efficiency of the tested algorithms is computed as the time until the victim battery depletes. As expected, the battery lasts longer when the victim defends itself by either rejecting connections from detected attackers classified using the Bayesian Network ("L"), using secure bit commitment ("C"), or both (L+C), rather than the standard method ("S"). Each point is averaged over 10 runs performed with different random seed values. Simulations where

the attackers also quit prematurely, rejecting assigned GO roles and retrying connections are marked with "R". A victim drops a GO Request from peers detected as attackers if their peer fairness is $PF > 60\%$.

Figure 3.7 describes the effect of the *TBB attack strength* (percentage of manipulated TBBs). In this set of experiments, there are only two devices, namely the victim and the FFBD attacker. For each simulation, the attacker follows a repeated communication schedule with groups lasting for 1 minute every 6 minutes. The attacks are detected by the Bayesian Network starting at the strength of 30%, while commitments can enforce fair behavior.

The Figures 3.8 and 3.9 describe the performance with 5 and 10 devices, respectively. The devices follow a repeated communication schedule with groups lasting for 1 hour every 12 hours. For these sets of experiments, a fraction of the devices acts as attackers by manipulating the TBB in each group formation process that they initiate. It is also observed that the battery of the victim will last longer with method L and many attackers since the victim device will detect and avoid unfairly communicating with attackers in GO mode, saving more battery.

Figure 3.10 describes the effect of the *R attack strength* (percentage of negotiations where the attacker quits prematurely when becoming GO). All points are computed at a TBB attack strength of 50%, while the R attack strength varies. While it is observed that, as expected, secure bit commitment cannot defend from a strong R attack, learning allows for longer battery life of victims as they drop communication with attackers.

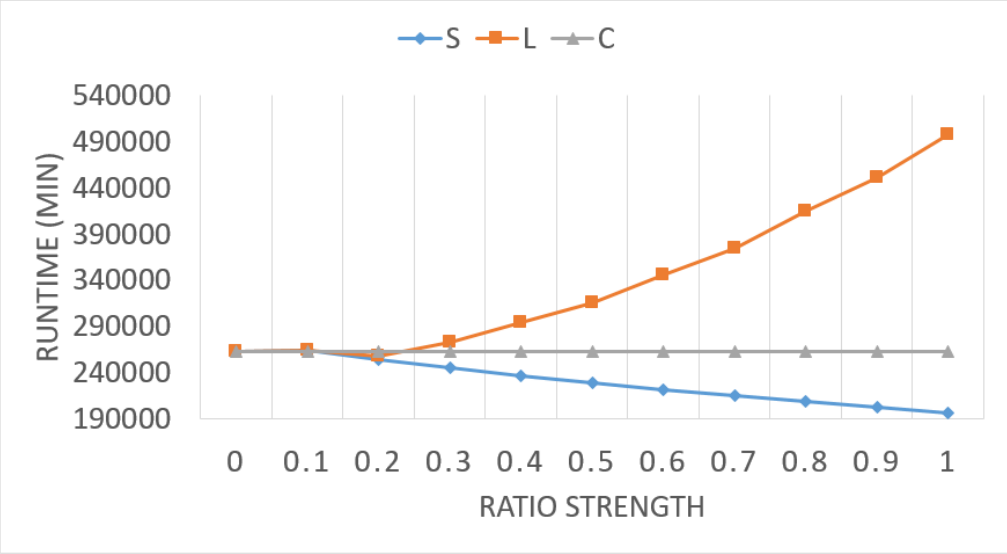

Figure 3.7: Effect of various TBB attack strengths

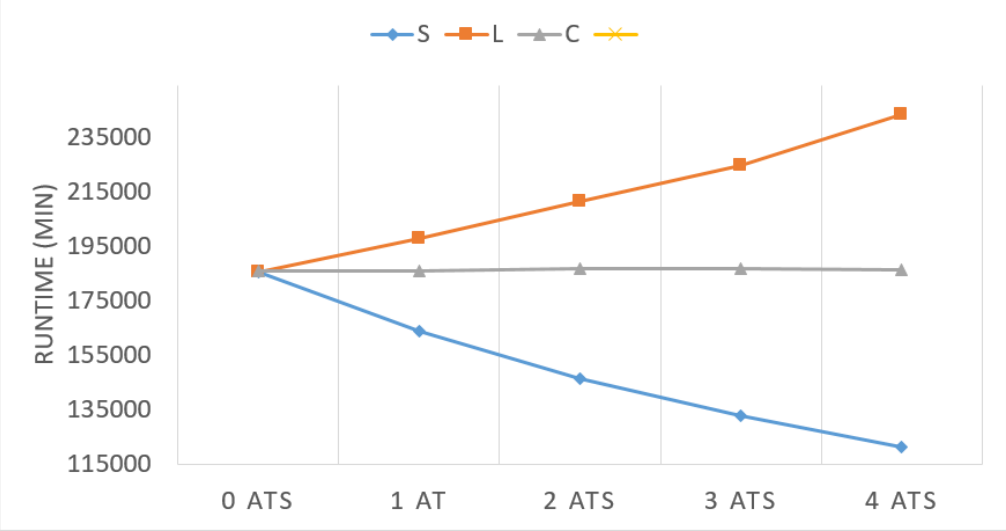

Figure 3.8: Effect of ratio of attackers with 5 devices

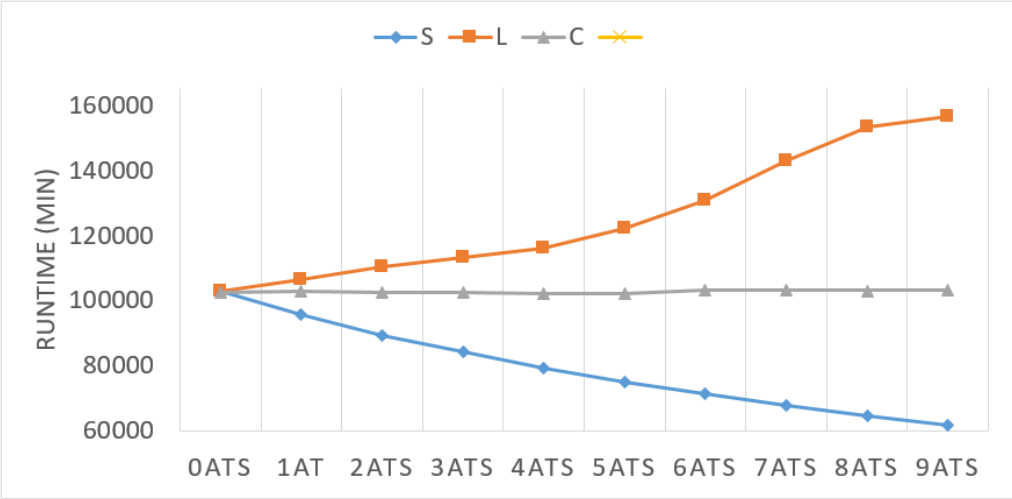

Figure 3.9: Effect of ratio of attackers with 10 devices

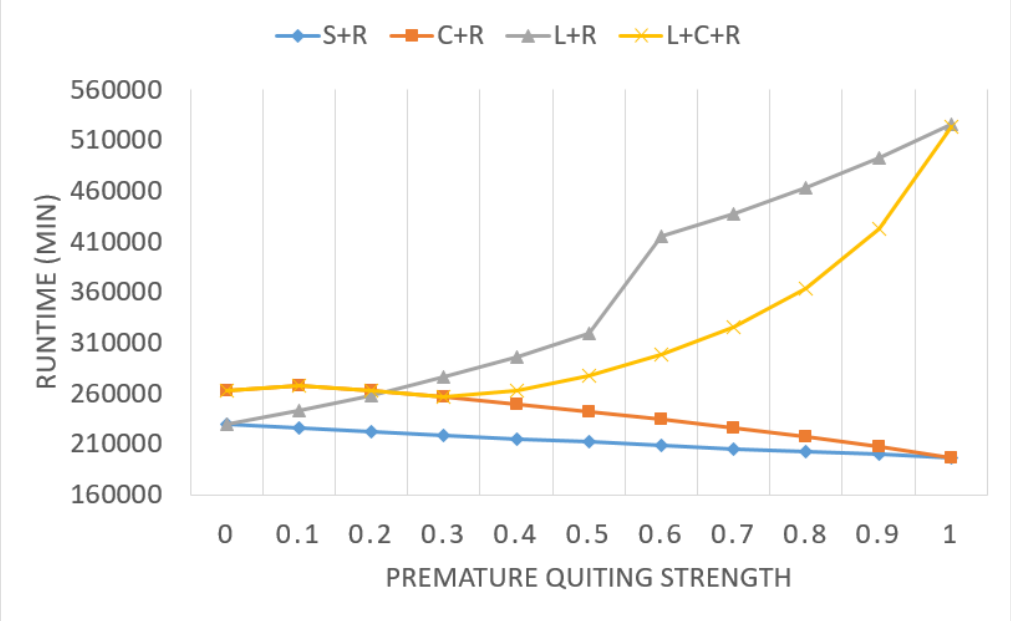

Figure 3.10: Effect of various R attack strengths

Chapter 4

Approximated Belief Network Models for Testing the Design in Production

Extending our previous research, we wanted to know:

- Can we simplify and codify what we did in our previous study and make it generic and debatable?
- How much power does being Group Owner cost?
- What are the performance characteristics of our algorithms if we let them run?
- Can we apply our technique to something other than Wi-Fi Direct?

4.1 Testable Models

Our first causal model described in Figure 3.3, states that an attacker is likely to use all known attacks (grounding the TBB bit, prematurely quitting before data is transmitted, and simply not transferring equal amounts of data) if they attack. However, in our previous experiments we considered the case where the attacker only grounded the TBB or only prematurely disconnected. The result is a cooling effect against an attack given the contradictory evidence.

Rather than describing the probability of an attack, we instead decided to describe the probability everything is behaving as expected and to focus on one variable at a time. To satisfy this new focus, we created a new generic Bayesian network as shown in Figure 4.1 which can be used on any variable which has an expected value. The "`State`" hidden variable represents the state of some aspect of the unit under test and has two values: "Operating Normally," "Operating Abnormally." The "`Is Informed`" node helps determine if there is sufficient knowledge to determine if the feature is "Operating Normally," similar to the concept behind Dempster-Shafer reasoning. Finally, the "`Variable Under Test`" is the variable being modeled which has an expected value it tends to be and its name will change based on the specific test being modeled. To simplifying processing we discrete-ized the "`Variable Under Test`" values into well below range (WBR), below range (BR), in range (R), above range (AB), and well above range (WAR); as presented in Table 4.1. Applying our technique to our previous work [2] yielded similar discreet-ization; however, a naive approach (immediate implementation of specifications) would only yield `iGO`, i.e., the fraction of negotiations assigning them as GO.

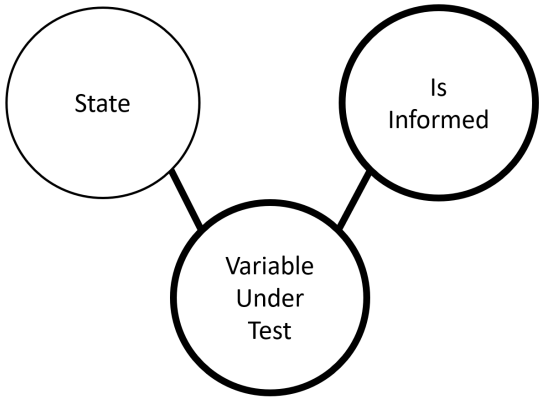

Figure 4.1: Differencing Generic Bayesian

	WBR	BR	R	AR	WAR
ignorant	0.4	0.499	0.5	0.499	0.4
some	0.25	0.48	0.6	0.48	0.25
informed	0.1	0.35	0.9	0.35	0.1

Table 4.1: Generic Conditional Probability Table Solved For State

As in the battery attack considered in the aforementioned literature, a person might try and cheat around being GO considering an expected requirement (not specified in [2]) that equal intentions of being GO would result in equal time spent as GO. For example, a peer-to-peer file transfer system could detect if it is GO or not and elect to transfer less information if it is a GO than if it is a client. Such an attack would only show up in `tGO`, namely the fraction of time spent as GO. This yields more likely the true intention of the protocol authors of [2] when they created the TBB, equal time of GO between participants. We consider it relevant to show how well our simplified Bayesian models compare with the original.

State	Amps
base	0.41
idle	0.48
discovery	0.51
client	0.59
GO	0.67

Table 4.2: States and their associated cost in Amps

4.2 Cost of Group owner

We need to quantify the device burden for being GO compares to being just a client, and furthermore, how much of an effect our technique would help when applied to detect and address battery attacks. To achieve this goal, we configured two identical Raspberry Pi 4b, each with an external COMFAST CF-WU825N Wi-Fi USB adapter (a RTL8192EU chipset).

All experiments used the external Wi-Fi adapter. Both devices were within six inches of each other. The experiment was conducted at an institution where there were approximately a dozen Wi-Fi infrastructure access points, and there was one nearby printer advertising itself for Wi-Fi Direct.

We then measured the Raspberry PI in several different states: base unit idle (no external adapter, no communication), idle (no communication), performing discovery, as a Wi-Fi Direct client, as Wi-Fi Direct GO as seen in Table 4.2.

For Rasberry Pi, which is a pretty powerful device, we note that the idle power of Raspberry PI caused the delta threshold between client and GO only to be 7%. However, another arm microprocessor, Teensy 4.1, that contains a USB port, only uses 100 mA [51]. We then scaled all measurements as expected if we

State	Amps
idle	0.17
discovery	0.2
client	0.28
GO	0.36

Table 4.3: Modified states and their associated cost in Amps

had run off the Teensy 4.1 instead of the Raspberry PI, and obtain the results in Figure 4.3. With the modified current costs for Teensy, perpetually being the client can provide up to a 15% power advantage over sharing the duties, allowing a comparison to show a cheater device appear to last 30% longer than the victim device.

When looking up different battery Amps-hour ratings, it became apparent we could not power these devices for a year – picking a $100 deep cycle battery would only provide about 180Ah of power [11] or about 44 days. To compensate for such a short time, we made the buckets only one deep (essentially eliminating them), but kept the 30 slots for the window size. We then simulated a two minute round until the device ran out of power that looked like the following:

- Spend 5 seconds in discovery
- Allow the potentially cheating device to flip for GO (or cheat)
- Allow the cheating device to potentially disconnect and try again (until the round's time is used up)
- If protecting via a Bayesian network, check whether we should disconnect or not

- If still connected, spend 49.9 seconds in either GO or as client depending on the outcome

- Idle any remaining time left in the round

We ran two sets of experiments with a new simulator written by an independent researcher from our original simulator focusing on only two devices and their interaction. In the first experiment, we changed the probability of cheating by grounding the TBB from 0% to 100% . In the second set of experiments, we changed the probability of disconnecting when GO from 0% to 100%. In both experiments we measured the first device's life and the percentage of time it declared an attack. In each set of experiments, we referred to the baseline as the "No defense" strategy, to see how the curve looked if the devices took no protection.

We added four additional experiments to each set. With our generic CPT replacing the variable under test: "our `iGO`" shows how `iGO` variable performs alone (number of times appointed `GO` divided by number of total connections with given peer), "our `pGO`" shows how `pGO` performs alone (fraction of cases with peer prematurely quitting from `GO` state), "our `tGO`" shows how `tGO` performs alone (fraction of communication with given peer spent as `GO` for current device). Finally, we wanted to show what would happen if we considered a single abnormal behavior affecting all three variables. The "Bayesian defend" curve corresponds to the existing battery attack defense in literature to show how our measured real hardware costs impact defense benefits for state of the art approaches. Finally we also repeated the bit commitment defense mechanism, shown in Figure 4.2 as the curve "bit commit". When considering the per-

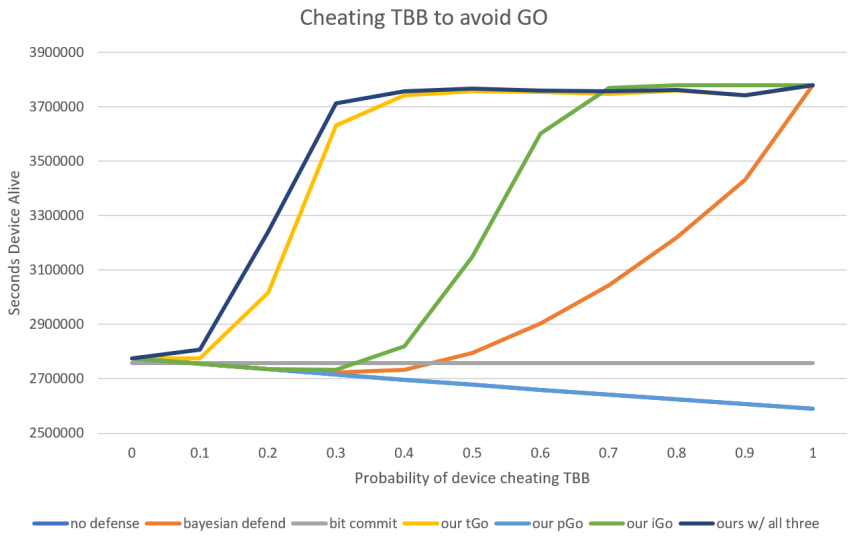

Figure 4.2: Cheating TBB Results

centage of time an attack was declared we only looked at the various Bayesian networks and its chart can be seen in [**?**].

It is not surprising that `pGO` could not detect a cheating TBB coin flip and acted identically to not having any defenses as in this experiment the cheating device was not prematurely disconnecting. What surprised us the most was how fast modeling only `tGO` responded to a cheating TBB. As both devices contained the Bayesian network, the cheating device would detect the first device disconnecting due to being cheated and start disconnecting, amplifying the speed at which the devices stopped communicating. The attacker device may avoid such extreme behavior by circumventing its own Bayesian defense, but the victim will still save power. This also explains why combining all three had an even faster response. This can be seen in Figure 4.2 and Figure 4.3.

In our second set of experiments, where the attacker on being assigned GO disconnects prematurely, modeling only `iGO` fairness requirements behaves as

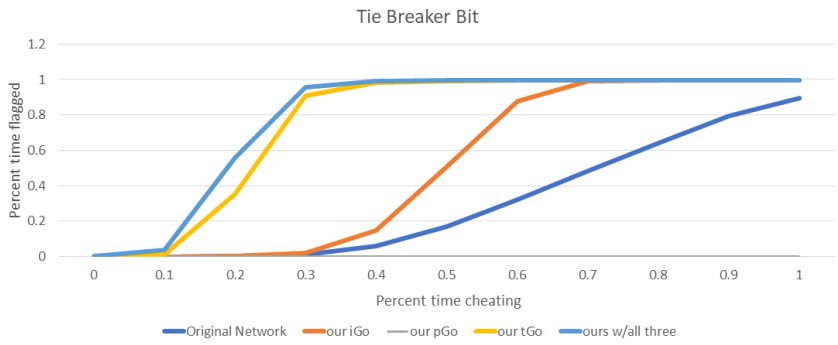

Figure 4.3: Percent time TBB cheating detected

one would expect and follows the curve for having no defense. Modeling only `pGO` has a swift reaction as this variable was modeled specifically for this purpose. Unlike attacking by manipulating the TBB, attacking by disconnect is not guaranteed to result in the victim becoming GO. Due to the reduced success rate, the use of `tGO` to detect the attack is slower.

When considering all three variables: `pGO`, `iGO`, and `tGO`; working together as independent sensors of the connection with the selected CPTs; there is initially a quicker response than with any of the variables individually; however, it takes much longer to fully distrust a disconnecting attacker device as `iGO` is always within range and therefore modifying the probability towards normal as can be seen in Figure 4.4 and Figure 4.5.

4.3 Mind Blog

As mentioned in the introduction of this chapter, we wanted a separate case study to use our technique with. The developers of `MindBlog.com` presented us with the following sequence diagram described in Figure 4.6 whose purpose was

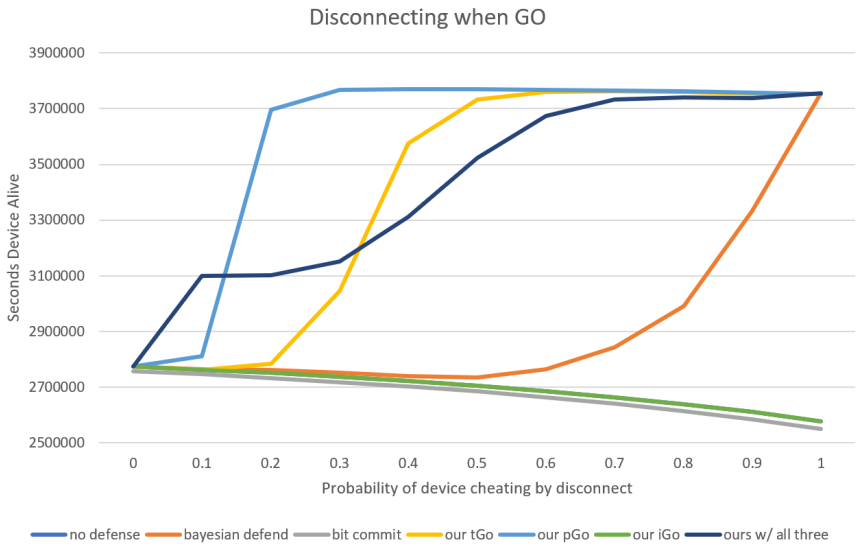

Figure 4.4: Premature Disconnect Results

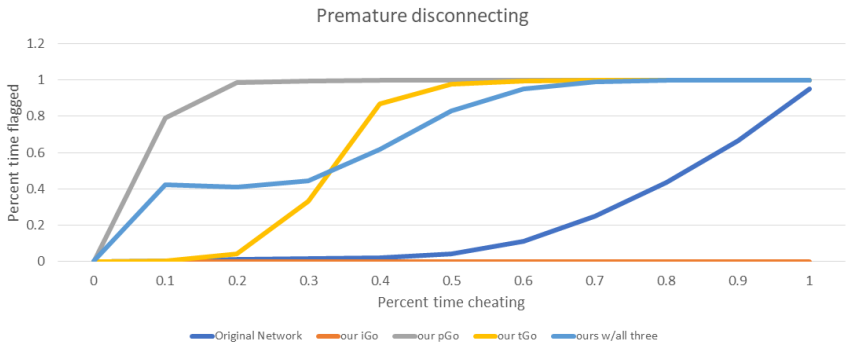

Figure 4.5: Percent time premature disconnect detected

to keep the communication low while the user was still undecided and changing things on the screen; then commit once it appeared they had made up their mind. During the final commit, the server was allowed making final corrections in case a conflict occurred.

The developers of `MindBlog.com` instrumented their recent version of the software with the ability to save log files for our analysis. Each log entry contained the commands being sent from the client to the server prior to the server

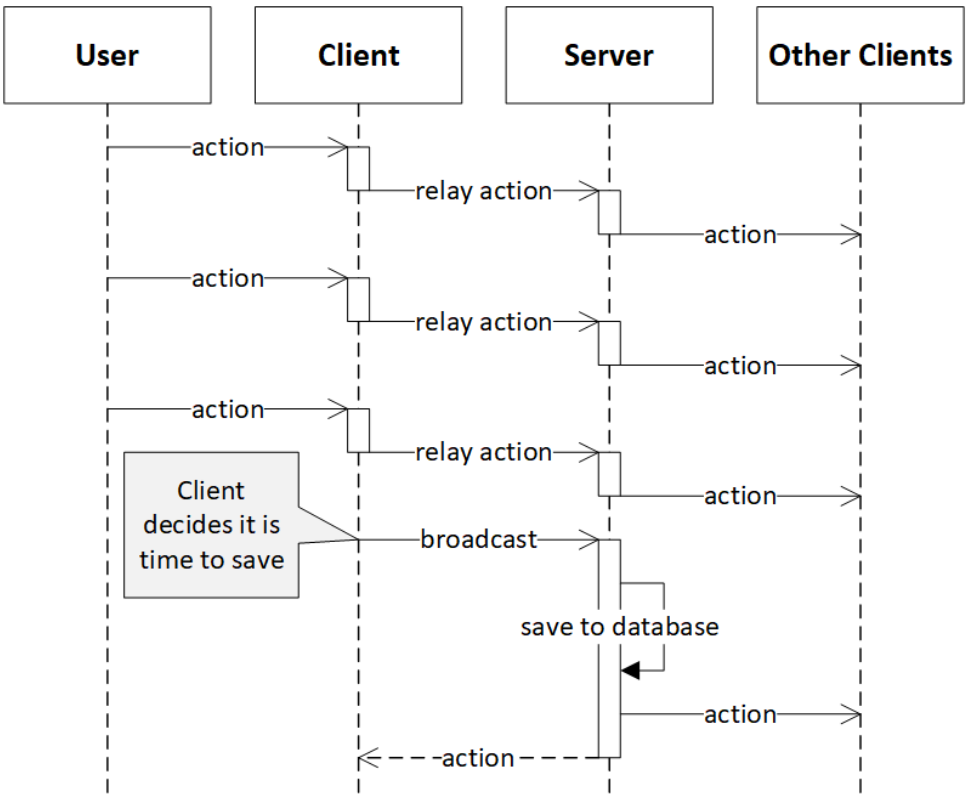

Figure 4.6: MindBlog.com communication

processing the command. The developers sometimes labeled a log file with a meaningful name, and other times left the log file with just the time it was saved. They then categorized the log file into two categories: 'debugging' with 15 files, and 'good' with three files. Logs within the debugging category where any log file generated while the developers were testing their software and log file placed in the good category were sessions where the developers were using the tool as it was intended and did not encounter any issues (see Figure 4.7).

We modeled the number of relay commands as a random variable under the assumption that most users would send approximately the same number of relay actions allowing the developers to detect frustrated users. We used the

category	# unlabeled	# labeled
debugging	6	9
good	0	3

Figure 4.7: Number of log files and their category

Bayesian network described in 4.1 where the number of commands for in range was trained based on one of the good logs (the other good logs were not used in training).

There were only three log files where an anomaly was detected; the first was when the developers were trying to debug selecting nodes, the second was when the developers were trying to figure out why nodes were not connecting, the third was when the developers first discovered a bug when delete was not working.

There were four other log files with relevant events: the log where the testing user of `MindBlog.com` attempted to pretend something was wrong, the log where menus did not always show up, and two additional log files where the developers continued to work on the delete bug mentioned above.

4.4 Characteristics of Our Bayesian Networks

Let us consider the variable under test to be a fair coin. As such, for a given window size we expect half the entries to be heads and half the entries to be tails. However, the binomial distribution describes a probability that all elements within our window have a single value regardless of the window size. Therefore, all algorithms will eventually declare a false positive given enough samples. We wanted to characterize our algorithm and we also wanted to know

Percent GO	min	1st quarter	median	mean	3rd quarter	max
0.5	110k	18m	42m	60m	80m	390m
0.55	1440	214k	500k	697k	961k	4m
0.60	101	6k	13k	20k	29k	149k
0.65	101	531	1k	2k	2k	10k
0.70	101	129	236	310	406	2k
0.75	101	101	104	137	157	653

Table 4.4: First detect Original Bayesian Network

if an attacker knew our algorithm; could they engineer a fixed pattern of coin flips such that we never detect them?

Our experiments used the Xosiro256** algorithm as described in [15] with a 1000 trials where each trial used the same starting seed (trial 1 used a seed of 8000, trail 2 used a seed of 8001, etc) so that each algorithm was provided the same set of random values and provided them in the same sequence. We ran the algorithm with a fair coin, and then varying levels of unfair coins. We wanted to capture the statistics on three values: when does the algorithm first detect a positive; once a positive has been declared, for how long; and finally after it has been cleared, how long until it declares a positive again?

For our original Bayesian network, we saw a false positive anywhere after 110 thousand flips and 390 million flips with an average of 60 million flips. Inducing even a slight bias into the coin caused first detection to shift considerably sooner as can bee seen in Table 4.4. The duration of false positives seems reasonable; however, as can be seen in Table 4.5, it does not improve with biased coins. Ideally, one would expect an unfair coin to be declared cheating and stay cheating. Finally, as can be seen in Table 4.6, the second detection always occurs more sooner than the first detection.

Percent GO	min	1st quarter	median	mean	3rd quarter	max
0.5	1	1	2	3.8	4	50
9 0.55	1	1	2	4.6	5	41
0.60	1	1	3	6.5	7	75
0.65	1	2	4	9.9	10	153
0.70	1	2	5	15	15	171
0.75	1	4	104	46	65	505

Table 4.5: Duration Original Bayesian Network

Percent GO	min	1st quarter	median	mean	3rd quarter	max
0.5	1	4	17m	40m	80m	390m
0.55	1	4	116k	398k	564k	4m
0.60	1	3	437	10k	16k	105k
0.65	1	2	10	568	734k	7k
0.70	1	2	7	89	89	1k
0.75	1	2	7	35	30	480

Table 4.6: Second detect Original Bayesian Network

Percent GO	min	1st quarter	median	mean	3rd quarter	max
0.50	13k	9m	21m	31m	42m	263m
0.55	1440	214k	500k	697k	961k	4m
0.60	101	6k	13k	20k	29k	149k
0.65	101	531	1k	2k	2k	10k
0.70	101	129	236	310	406	2k
0.75	101	101	104	137	157	653

Table 4.7: First Detect Simplified Bayesian Network

Our simplified Bayesian network had a similar performance profile; however, it was declaring false positives more frequently as can be seen in Table 4.7. Both duration and second detect saw a similar pattern to the original network as can be seen in Table 4.8 and Table 4.9

Both networks performed identically where the coins had to favor cheating

Percent GO	min	1st quarter	median	mean	3rd quarter	max
0.50	1	1	2	4	4	50
0.55	1	1	2	5	5	41
0.60	1	1	3	6	7	75
0.65	1	2	4	10	10	153
0.70	1	2	5	15	15	171
0.75	1	4	18	46	65	505

Table 4.8: Duration of Simplified Bayesian Network

Percent GO	min	1st quarter	median	mean	3rd quarter	max
0.50	1	5	8m	20m	30m	185m
0.55	1	4	115k	400k	564k	4m
0.60	1	3	437	6	16k	105k
0.65	1	2	10	568	734	7k
0.70	1	2	7	89	89	1k
0.75	1	2	7	34	30	480

Table 4.9: Second Detection of Simplified Bayesian Network

by 76% to be detected. However if the coin was cheating 76% of the time, the duration was permanently flagged.

Chapter 5

Reputation Bayesian Networks

Given we were able to devise an attack against our previous work, we wanted to know if we could improve the performance characteristics of our generic Bayesian network by modifying the prior probability with the posterior. Furthermore, we wanted to be able to train a network where the average false positive was higher than a standard Bayesian network with the same causal model.

Let us consider the prior probability of a Bayesian network to be the reputation for a given connection. After the network runs, let us update the prior probability from the posterior probability and cap the prior probability between 0.1 and 0.99 respectively as seen in Figure 5.2 where $p(t)$ is the prior probability and $p(t)'$ is the posterior probability. The caps provide forgetfulness to the reputation value allowing peers who change their behavior to be detected and to provide known prior probabilities to base the CPT off of. Similar to other systems, a trigger value is specified when to declare a fault.

We will consider the simplified Bayesian network described in Figure 4.1. The posterior probability follows the recurrence relation given in 5.1 where x is

an entry in the CPT table.

$$p(t)' = \frac{p(t) * x}{(1 - p(t)) * (1 - x) + p(t) * x} \tag{5.1}$$

$$p(t+1) = \begin{cases} 0.01, & \text{if} p(t)' \leq 0.01 \\ 0.99, & \text{if} p(t)' \geq 0.99 \\ p(t)' & \text{otherwise} \end{cases} \tag{5.2}$$

By feeding the posterior into the prior, the 'probabilities' in the CPT are no longer the probability of the event occurring. Instead they are conditioned off the number of times the event should occur to move the reputation from either max reputation to trigger threshold; or the number of times the event should occur to move the reputation from the minimum value to the trigger threshold. A value of 0.5 indicates the observed condition does not contribute to the reputation either positively or negatively. Values above 0.5 will increase the reputation score, where as values below 0.5 will decrease the reputation score.

In a typical setup, normal values should increase the reputation of the peer device and increment the reputation score. As normal values are by definition normal, for an honest node, the reputation score will be maximum. Even though any prior could work, perhaps based on the percentage of devices cheating from a particular company; we initialize the prior to the maximum value.

As CPT values are conditioned off the number of times necessary to move the reputation from either the minimum to the maximum; one can calculate x directly by solving the recurrence relation for the number of times the value

should be repeated consecutively to move from the extreme value to the trigger. For example, suppose the designer wanted trust a device after receiving 28 in range events in a row. Solving the equation 5.1 with $p(t) = 0.01, p(t+29) \geq 0.1$ gives $x \approx 0.51397$.

While each entry and the CPT could be solved in this manner; we would prefer to train the reputation network to target a specific false positive rate, the number of samples before an honest node is declared dishonest due to bad luck. We set the CPT for R to 0.5003 or 1999 consecutive times from the minimum prior to the trigger threshold. We observed that the recurrence relation described in Figure 5.1 was a monotonically incrementing smooth function between the ranges of 0 and 1 exclusive allowing a binary search of various probability values. However, complicating the search is that any particular run can be shorter or longer than what the value should produce so one must test the value multiple times. We required a minimum set of runs before we made a decision and that the high count be twice that of the low count to declare the x too high or a low count twice that of the high count to declare x too low as seen in Figure 1.

Data: $TFPR$, CPT, $MinTimes$

Result: x

while x *still undecided* **do**

Reset input data stream if necessary;

$lowCount = 0$, $highCount = 0$, $targetCount = 0$, $totalCount = 0$;

while *direction to move x is undecided* **do**

Run until $first$ anomaly detected and store in t;

totalCount = totalCount +1 ;

increment lowCount, highCount, or targetCount based on t if t is below, above, or just right;

if $totalCount \geq MinTimes$ **then**

if $lowCount \geq highCount * 2$ **then**
declare x is too low and repeat

end

if $highCount \geq lowCount * 2$ **then**
declare x is too high and repeat

end

if $targetCount \geq abs(lowCount - highCount)$ **then**
declare x is just right and exit out

end

if *new x and old x don't change enough* **then**
declare x is just right and exit out

end

end

end

end

Algorithm 1: Learning Reputation Based CPT Values

network	WBR	BR	R	AR	WAR
1st	0.375	0.496094	0.5003	0.496094	0.375
2nd	0.375	0.496033	0.5003	0.496033	0.375
3rd	0.451172	0.496033	0.5003	0.496033	0.451172
4th	0.451875	0.496094	0.5003	0.496094	0.451875
5th	0.484375	0.496094	0.5003	0.496094	0.484375

Table 5.1: Informed CPT row with FPR equal to 60m

Pct Go	min	1st quarter	median	mean	3rd quarter	max
0.50	2m	16m	50m	66m	102m	281m
0.55	2k	4k	5k	6k	7k	12k
0.60	447	889	1k	1k	1k	3k
0.65	114	114	628	593	682	922
0.70	114	114	340	337	472	626
0.75	114	114	153	176	211	389

Table 5.2: First detect of first network trained targeting 60m FPR

5.1 Experiment

We trained a network based off the algorithm described in Algorithm 1 targeting a first detect false positive of 60 million samples. Table 5.1 shows the various CPTs that were trained.

Repeating our earlier experiment of finding the first detect, the duration and the second detect shows that our average first false positive is around 60 million as seen in Table 5.2 at the cost of being slightly slower (by at least 14 samples) when the dice is not fair. However, the duration of how long we declare a positive is far improved as seen in Table 5.3.

Finally we were curious how the trained network would compare against our original simplified networks they were based off of when placed in our Wi-

Pct GO	min	1st quarter	median	mean	3rd quarter	max
0.50	3	123	475	1225	2183	5612
0.55	5	290	11k	∞	78k	∞
0.60	15	∞	∞	∞	∞	∞
0.65	∞	∞	∞	∞	∞	∞
0.70	∞	∞	∞	∞	∞	∞

Table 5.3: Duration of first network trained targeting 60 FPR

Fi Direct simulator. We reran each of the percent disconnect graphs for both TBB and premature quitting which had non-zero values to see how the reputation network would compare with the simplified network. In each comparison graph, the original network refers to the standard Bayesian network whereas the reputation network describes the treatment described here. Even though the reputation network is no longer dealing with probabilities, we treated it as if was still a Bayesian network and its results can be seen in Figure 5.3 and in Figure 5.6. In all cases, the reputation network detects soon than the original network which can be seen in Figure 5.1, Figure 5.2, Figure 5.3, Figure 5.4, Figure 5.5, and Figure 5.6. One word of cause when combining all three reputation networks as if they were Bayesian networks, the false positives increase enough to be noticable. This is likely caused by being unlucky and one participant getting slight more time as GO, which will cause both iGo and tGo to be high simultaneously causing the devices reputation to drop faster than looking at a single variable.

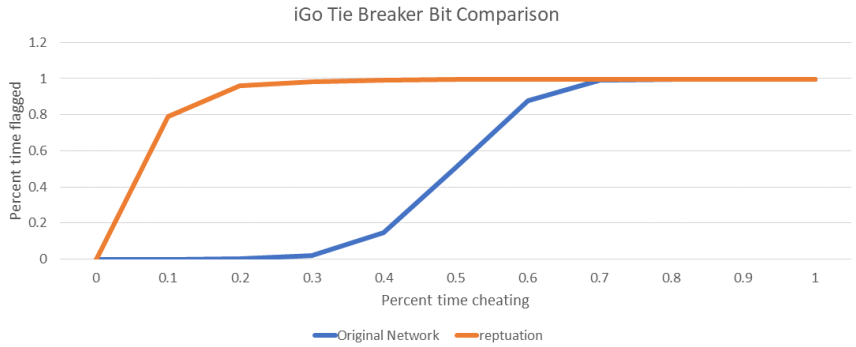

Figure 5.1: iGo TBB comparison between simplified network and reputation network

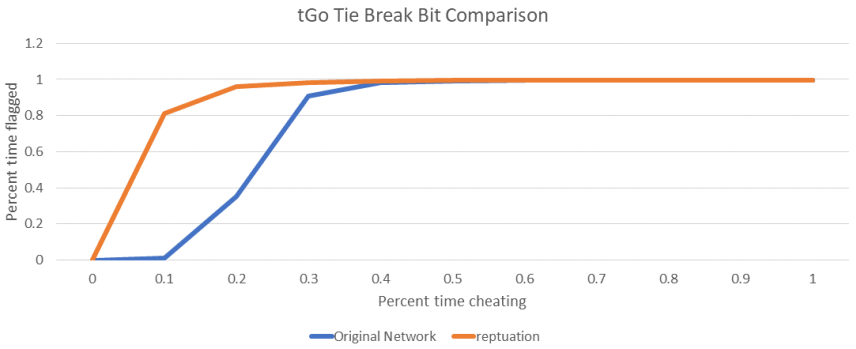

Figure 5.2: tGo TBB comparison between simplified network and reputation network

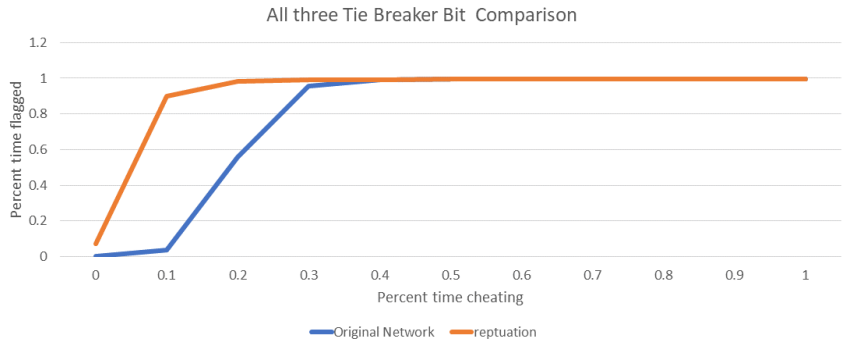

Figure 5.3: All three variable TBB comparison between simplified network and reputation network

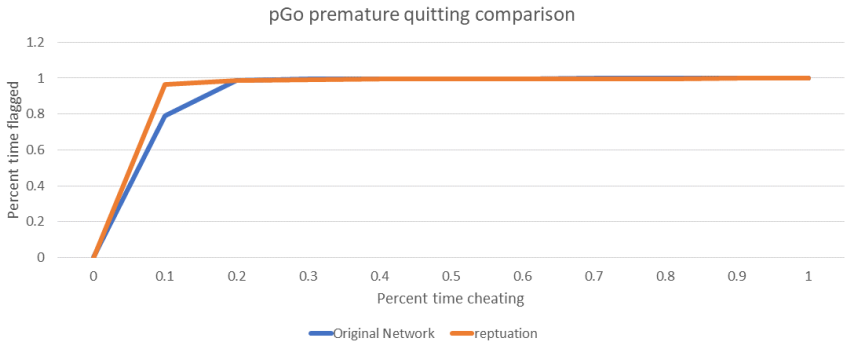

Figure 5.4: pGo disconnect comparison between simplified network and reputation network

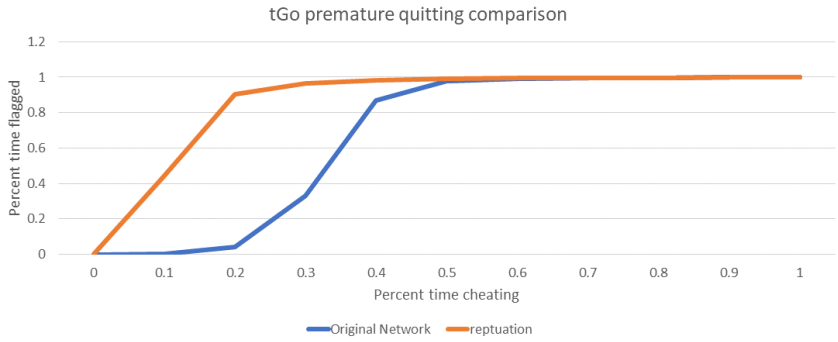

Figure 5.5: tGo disconnect comparison between simplified network and reputation network

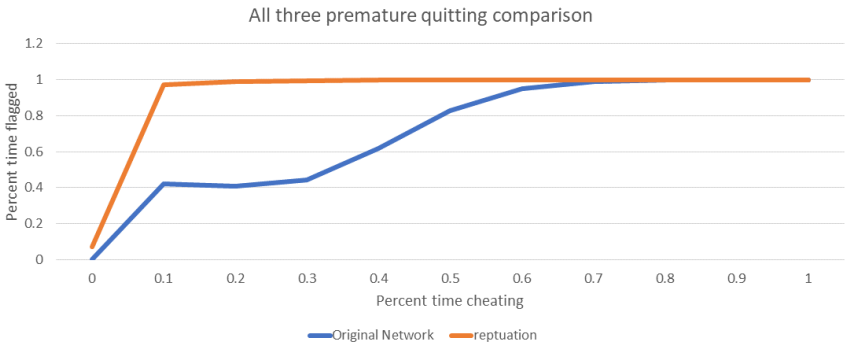

Figure 5.6: All three variable disconnect comparison between simplified network and reputation network

Chapter 6

Conclusion

A new methodology in testing focused on approximate models using Bayesian Networks is shown to enable detection of issues ranging from security attacks to usability of user interfaces even when modeling the expected behavior of only a single variable.

The complexity and dynamism of software development generally preclude exact modeling of all requirements and testing of behavior leaving defects in software which can account for up to 75% of the company's operational budget [35]. Providing the ability for a company to see in real time that their users are not following their expectations can help reduce that cost by allowing the company to be proactive.

The False Friend Battery Depletion (FFBD) attack is identified as a potential hazard against devices in smart homes, where battery-powered victims are coaxed into volunteering to support attacker devices, as intensive power consuming group owners, reducing their own availability to end-users and damaging their brand reputation. We address FFBD attacks based on manipulating TBBs and on premature quitting. A set of solutions is proposed for addressing FFBD

attacks, using secure bit commitments, learning statistical profiles of peers, and classifying them using Bayesian Networks. The solutions based on secure bit commitments proved to be very robust, but would require changes in the Wi-Fi Direct standard group formation protocols, changes which are unlikely to occur soon given that many devices already support the current standard. The results based solely on learning statistical profiles and detection using Bayesian Networks were shown to also work very well, being a valid alternative where attacks are detected and automatically denied.

Reputation Bayesian networks are shown to help address some of the short falls of our Bayesian networks CPTs allowing an engineer heuristics to better set or learn the parameters and prevent the toggling of a positive alarm when an anomaly is ongoing.

Finally, applying our technique to a domain completely different from embedded wireless technology shows the technique is general purpose and not limited to just embedded devices allowing all industries to benefit from this methodology.

References

[1] Deniz Akdur, Vahid Garousi, and Onur Demirörs. A survey on modeling and model-driven engineering practices in the embedded software industry. *Journal of Systems Architecture*, 91:62–82, 2018.

[2] W Alliance. Wifi peer-to-peer (p2p) technical specification version 1.7. *Wi-Fi Alliance Technical Committee P2P Task Group*, 7:1–201, 2016.

[3] Wi-Fi Alliance. Wi-fi simple configuration technical specification version 2.0.6, 4 2018.

[4] Ala Altaweel, Radu Stoleru, and Guofei Gu. Evildirect: A new wi-fi direct hijacking attack and countermeasures. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–11. IEEE, 2017.

[5] Rabéa Ameur-Boulifa, Florian Lugou, and Ludovic Apvrille. Sysml model transformation for safety and security analysis. In *Security and Safety Interplay of Intelligent Software Systems*, pages 35–49. Springer, 2018.

[6] Parmar Amish and VB Vaghela. Detection and prevention of wormhole attack in wireless sensor network using aomdv protocol. *Procedia computer science*, 79:700–707, 2016.

[7] Arash Asadi and Vincenzo Mancuso. Wifi direct and lte d2d in action. In *Wireless Days (WD), 2013 IFIP*, pages 1–8. IEEE, 2013.

[8] National Defense Industry Association et al. Final report of the model based engineering (mbe) subcommittee, 2011.

[9] R. Atat, L. Liu, N. Mastronarde, and Y. Yi. Energy harvesting-based d2d-assisted machine-type communications. *IEEE Transactions on Communications*, 65(3):1289–1302, March 2017.

[10] Timothy Atkinson and Marius-Calin Silaghi. An efficient way to access an array at a secret index. *IACR Cryptol. ePrint Arch.*, 2006:157, 2006.

[11] Batteries+Bulbs. Sli24mdc group 24 12 volt deep cycle marine, boat and rv battery, 2020 (accessed August 21, 2020).

[12] M.C Bech. Wi-fi direct vs bluetooth vs zigbee, 2 2019.

[13] Youcef Begriche, Rida Khatoun, Ali Rachini, and Lyes Khoukhi. A reputation system using a bayesian statistical filter in vehicular networks. In *2020 Sixth International Conference on Mobile And Secure Services (MobiSecServ)*, pages 1–7. IEEE, 2020.

[14] Victor Pazmino Betancourt, Thomas Glock, Aleksei Kharitonov, Matthias Kern, Bo Liu, Eric Sax, and Jürgen Becker. Linking intrusion detection system information and system model to redesign security architecture. In *2020 IEEE International Systems Conference (SysCon)*, pages 1–7. IEEE.

[15] David Blackman and Sebastiano Vigna. Scrambled linear pseudorandom number generators. *arXiv preprint arXiv:1805.01407*, 2018.

[16] Bruno Blanchet. Symbolic and computational mechanized verification of the ARINC823 avionic protocols. In *30th IEEE Computer Security Foundations Symposium (CSF'17)*, pages 68–82, Santa Barbara, CA, USA, August 2017. IEEE.

[17] Xianghui Cao, Devu Manikantan Shila, Yu Cheng, Zequ Yang, Yang Zhou, and Jiming Chen. Ghost-in-zigbee: Energy depletion attack on zigbee-based wireless networks. *IEEE Internet of Things Journal*, 3(5):816–829, 2016.

[18] Aaron Carroll, Gernot Heiser, et al. An analysis of power consumption in a smartphone. In *USENIX annual technical conference*, volume 14, pages 21–21. Boston, MA, 2010.

[19] Massimo Condoluci, Leonardo Militano, Antonino Orsino, Jesus Alonso-Zarate, and Giuseppe Araniti. Lte-direct vs. wifi-direct for machine-type communications over lte-a systems. In *2015 IEEE 26th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 2298–2302. IEEE, 2015.

[20] Jonathan Corley, Brian P Eddy, Eugene Syriani, and Jeff Gray. Efficient and scalable omniscient debugging for model transformations. *Software Quality Journal*, 25(1):7–48, 2017.

[21] Jarbele CS Coutinho, Wilkerson L Andrade, and Patrícia DL Machado. Requirements engineering and software testing in agile methodologies: a systematic mapping. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, pages 322–331, 2019.

[22] Rodrigues da Silva Alberto. Model-driven engineering: A survey supported by the unified conceptual model. 43:139–155, 2015.

[23] Fangwei Ding, Feng Xia, Wei Zhang, Xuhai Zhao, and Chengchuan Ma. Monitoring energy consumption of smartphones. In *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, pages 610–613. IEEE, 2011.

[24] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. Configuration knowledge representation using uml/ocl. In *International Conference on the Unified Modeling Language*, pages 49–62. Springer, 2002.

[25] Shyamnath Gollakota, Nabeel Ahmed, Nickolai Zeldovich, and Dina Katabi. Secure in-band wireless pairing. In *USENIX security symposium*, pages 1–16. San Francisco, CA, USA, 2011.

[26] John Grundy, Hourieh Khalajzadeh, and Jennifer Mcintosh. Towards human-centric model-driven software engineering. In *ENASE*, pages 229–238, 2020.

[27] Matthew Hause et al. The sysml modelling language. In *Fifteenth European Systems Engineering Conference*, volume 9, pages 1–12. Citeseer, 2006.

[28] IEEE. Standard 802.11 part 11. page 466, 2012.

[29] Roslan Ismail. The beta reputation system.

[30] Daniel Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(2):256–290, 2002.

[31] Camenisch Jan. The wonderful world of global random oracles. pages 280–312, 2018.

[32] Yu Jiang, Han Liu, Houbing Song, Hui Kong, Rui Wang, Yong Guan, and Lui Sha. Safety-assured model-driven design of the multifunction vehicle bus controller. *IEEE Transactions on Intelligent Transportation Systems*, 19(10):3320–3333, 2018.

[33] Meihua Jin, Ji-Young Jung, and Jung-Ryun Lee. Dynamic power-saving method for wi-fi direct based iot networks considering variable-bit-rate video traffic. *Sensors*, 16(10):1680, 2016.

[34] Paul C Jorgensen. *Software testing: a craftsman's approach*. CRC press, 2018.

[35] Herb Krasner. The cost of poor quality software in the us: A 2018 report. *Consortium for IT Software Quality, Tech. Rep*, 10, 2018.

[36] Arianit Maraj, Timothy Atkinson, and Marius C Silaghi. Learning strategies for resisting power attacks on wi-fi direct group formation. In *FLAIRS*, 2019.

[37] T. Martin, M. Hsiao, Dong Ha, and J. Krishnaswami. Denial-of-service attacks on battery-powered mobile computers. In *Second IEEE Annual Conference on Pervasive Computing and Communications, 2004. Proceedings of the*, pages 309–318, March 2004.

[38] Arthur Henrique de Andrade Melani and Gilberto Francisco Martha de Souza. Mapping sysml diagrams into bayesian networks: A systems engineering approach for fault diagnosis. *ASCE-ASME J Risk and Uncert in Engrg Sys Part B Mech Engrg*, 6(3), 2020.

[39] Liao Chih-Chiang Shin-Ming Cheng Domb Memachem. On designing energy efficient wi-fi p2p connections for internet of things. IEEE, June 2017.

[40] Naor Moni. Bit commitment using pseudo-randomness, 2001.

[41] None. Information technology – security techniques – hash-functions, 2004.

[42] None. Unified modeling language, 1 2019.

[43] Judea Pearl. Bayesian netwcrks: A model cf self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine, CA, USA*, pages 15–17, 1985.

[44] Judea Pearl and Dana Mackenzie. *The book of why: the new science of cause and effect*. Basic books, 2018.

[45] D. R. Raymond and S. F. Midkiff. Denial-of-service in wireless sensor networks: Attacks and defenses. *IEEE Pervasive Computing*, 7(1):74–81, Jan 2008.

[46] Shahid Raza, Prasant Misra, Zhitao He, and Thiemo Voigt. Building the internet of things with bluetooth smart. *Ad Hoc Networks*, 57:19–31, 2017.

[47] Hamilton Kim Miles Russ. *Learning UML 2.0*. O'Reilly Media, Inc, 2008.

[48] Nash D. C.. Martin T. L. Ha D. S. Hsiao M. S. Towards an intrusion detection system for battery exhaustion attacks on mobile computing devices. pages 141–145. IEEE, 2005.

[49] Glenn Shafer. *A mathematical theory of evidence*, volume 42. Princeton university press, 1976.

[50] Vladimir Shakhov and Insoo Koo. Depletion-of-battery attack: Specificity, modelling and analysis. *Sensors*, 18(6):1849, 2018.

[51] Sparkfun. Teensy 4.1, 2020 (accessed August 10, 2020).

[52] Internet Live Stats. Internet live stats, 2019.

[53] Md Tauhiduzzaman and Mea Wang. Fighting pollution attacks in p2p streaming. *Computer Networks*, 79:39–52, 2015.

[54] Mostafa Uddin and Tamer Nadeem. A2psm: audio assisted wi-fi power saving mechanism for smart devices. In *Mobile Computing Systems and Applications*, page 4. ACM, 2013.

[55] Burak Uzun and Bedir Tekinerdogan. Model-driven architecture based testing: A systematic literature review. *Information and Software Technology*, 102:30–48, 2018.

[56] S. Viehbck. Wi-fi protected setup online pin brute force vulnerability, cert vulnerability note vu 723755. 2011. `https://www.kb.cert.org/vuls/id/723755`, 2018.

[57] Yufei Yin, Yiqun Xu, Weikai Miao, and Yixiang Chen. An automated test case generation approach based on activity diagrams of sysml. *International Journal of Performability Engineering*, 13(6), 2017.

[58] H. Yoo, S. Kim, S. Lee, J. Hwang, and D. Kim. Traffic-aware parameter tuning for wi-fi direct power saving. In *2014 Sixth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 479–480, July 2014.

[59] Umut Çabuk, Georgios Kanakis, and Feriştah Dalkılıç. Lte direct as a device-to-device network technology: Use cases and security. *International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE)*, 5:401, 07 2016.

Appendix A

Background

The goal of software testing and requirement engineering is to develop products that meet the customers' expectations [21].

Stated another way, the product should not only match the specified requirements, but also the expected behavior. Unfortunately, exhaustively testing of the specified requirements is generally not practical due to the large number of possible input sequences [55]. This does not count the potential "bugs" from not satisfying expected behavior.

Testing is typically broken up into test cases, a set of inputs with a set of expected outputs with an identity. Typically test cases are discovered by one of two methods: through the specification or through the source code. There is a debate as to which approach is better, as each method has its strengths and weaknesses and many argue a good test plan should use both [34]. Regardless of the method used, these tests only have value the first time they are run against the target as they either help reduce risk of potential defects by either revealing a problem or fail to detect a problem thereby giving confidence the application is working as intended.

A.1 Models

Modeling is usually defined for communication, analysis, or guiding the production process as such models are diverse in nature and quality [55]. There are many definitions for what a model is; however, there seem to be two points that all the definitions agree with: a model defines a system under study, and the model is a system in its own right [22]. A model is "a system that helps define and give answers of the system under study without the need to consider it directly" [22].

The Stachowiak process calls for three criteria to determine if an artifact can be considered a model rather than something else: pragmatism, reduction, and mapping. The artifact must be functional and capable of replacing the original for specific purposes. It must be a simplified version of the original so that not all aspects are depicted. It must be possible to identify the original phenomenon in the model [22].

Software modeling is meant to ensure functionality is complete and correct by meeting end-user needs and ensuring the program design supports its requirements [42]. The Unified Modeling Language (UML) initial version came out of combining the top three methods from the method wars: Object-Oriented Analysis and Design, Object-Oriented Software Engineering approach, and Object Modeling Technique [47]. After 10 to 15 years of use, the UML standard was reworked to clean up the previous decade worth of additions and simplify the specification.

There are two other methods that models can be used: static analysis through consistency and completeness checks and generating dynamic tests [55].

Our focus is neither on consistency checks or generating dynamic tests but rather using models to passively listen to user inputs and decide if they are expected given the design.

A.2 Alternative Peer-to-Peer Specifications

Wi-Fi simple configuration is an optional protocol for standard IEEE.11 infrastructure mode whose purpose is to reduce the number of support calls, equipment returns due to complex initial setup and properly configure security for home configuration [3]. There are three roles in Simple Configuration: Access Point (AP), registrar, and enrollee. The AP has the ultimate responsibility of performing link-layer access control and routing traffic to the WLAN should one be connected. The registrar, which can be the same as the AP, allows new enrollees to join the network. Finally, the enrollee is the new device being added.

There are many different ways that a manufacturer can choose to implement Wi-Fi Simple configuration. They support both in-band, using the Wi-Fi itself for setup; out-of-band media, like an NFC tag; or a hybrid of technologies. Typically, the configuration requires entering a PIN or a password into the enrollee. Unfortunately, according to [56] the PIN method is vulnerable to brute force attacks. An alternative, for users who wish to sacrifice security for convenience, there is a push-button method (without a pin) where a user can press a button on both the registrar and the enrollee at the same time–which is vulnerable to man-in-the-middle attacks [25]. By default, after registration, the enrollee will be configured using WPA2-Personal.

Long Term Evolution (LTE) Direct is a competing technology for device-to-device communication that uses licensed cellphone frequencies instead of unlicensed spectrum. One of the primary differences between LTE Direct and other device-to-device communication is an operator has to grant spectrum to devices wishing to pair. Several of the use cases for LTE Direct focus on advertising, causing Wi-Fi direct to be more popular for device-to-device communication [59].

The energy efficiency of LTE Direct and Wi-Fi direct for uploading data to the internet was measured by [19]. They found that LTE Direct was more efficient with large number of users while Wi-Fi Direct was more efficient for small packets. An aggregation protocol was proposed by [7] for aggregating Wi-Fi direct onto LTE Direct.

Bluetooth is the classic device-to-device wireless protocol targeting low-powered devices [46]. However, unlike Wi-Fi, Bluetooth Low power is a closed standard, only supports client/server communication, lacks multi-hop paradigms, and does not allow streaming of data [12].

Zigbee is a low data rate IoT communication protocol with a data rate far lower than either Bluetooth or Wi-Fi [12]. It has a range between 30 and 40 meters and is used primarily in sensor networks [17].

A.3 Cryptographic solutions

One method to protect an application is to obfuscate all data associated with the application. An extreme case of obfuscation is splitting the data into shares such that multiple people must work together in order to get work done [10]. The

issue with these techniques is that while they maybe mathematically hard to break if implemented correctly; it can be tricky for even a seasoned professional to get it right.

One interesting use of cryptography is to provide proof of a secret and then use that proof in interesting way. An example of such a technique is the simple bit commitment protocol in the random oracle model [31]. Assume an agent A wants to commit to a bit with value b. A generates a sufficiently large random number R, and then publishes the value $c = H(R, b)$ where H is a secure message digest function such as SHA-2 or Whirlpool [41]. The commitment is *binding*, in the sense that it is difficult for A to find other values for R and b that generate the same commitment c. If R is large enough, the commitments are also *hiding* in the sense that an attacker would not be able to efficiently try sufficiently many values of R and b in order to find one that matches c. Bit commitment techniques have also been proposed based of pseudo-random number generators as well as on other security concepts [40].

Coin flipping over the network by two agents A and B can be performed by having A commit to a bit b_A before B publicly discloses a bit b_B. When A receives b_B then it also reveals b_A. The result of the coin flip is the exclusive OR of the bits, $b_A \otimes b_B$.

A.4 Product Configurators

Product configurators are a set of tools that look at constraints for product selection. Two solutions are presented in [24] and [30] as they both use the class diagram, a standard software engineering modeling language, and use constraint

satisfaction problems to solve their particular problem. Felfernig et al. proposed using Object Constraint Language (OCL) binary expressions as their language of choice [24] while Jackson and Daniel created a special purpose language to do the same thing [30]. Both techniques can use any constraint satisfaction solver and were focused on picking products.