7-13-2022

# A Distributed Trust Model Simulator for Energy Grid of Things Distributed Energy Resource Management System

Abdullah Barghouti
*Portland State University*



Recommended Citation
Barghouti, Abdullah, "A Distributed Trust Model Simulator for Energy Grid of Things Distributed Energy Resource Management System" (2022). *Dissertations and Theses.* Paper 6100.
https://doi.org/10.15760/etd.7960

# A Distributed Trust Model Simulator for Energy Grid of Things Distributed Energy Resource Management System

by

Abdullah Barghouti

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Thesis Committee:
John M Acken, Chair
Robert Bass
Yuchen Huang

Portland State University
2022

## Abstract

The evolution of networks into more distributed, self-reliant nodes has mitigated single-point failures that plagued traditional centralized networks. Applied to power grids, distributed systems can increase the integrity and availability of grid services while also offering a power management solution. However, while distributed networks provide scalability, security, and sustainability compared to centralized networks, their distributed nature makes them harder for anomaly detection and prevention. Incorporating a Distributed Trust Model (DTM) System into an Energy Grid of Things Distributed Energy Resource Management System (EGOT DERMS) allows grid participants to be characterized and their communication to be analyzed for possible attacks. A Trust Model simulator is needed to evaluate and improve the DTM System.

Trustworthiness is calculated using a Trust Model. While many trust models exist, most only consider 2-3 matrices to evaluate trust. The TM proposed in this thesis uses a Metric Vector of Trust (MVoT) monitoring 17 parameters when assessing trust. Moreover, unlike standard trust models, the proposed trust model establishes a method to test the trust between various actors within the network and probe the trust model itself. Using a Trust Model Simulator, MVoT calaculations, initial values, and parameters are fine-tuned to achieve high-confidence message classifications and minimize false positives. The DTM System and Trust Mode Simulation Suite allow for distributed trust evaluation with a real-

time classification of EGOT DERMS actors, providing additional security for distributed systems.

## Dedication

For my siblings, parents, and grandparents who showed their unconditional love and support.

## Acknowledgements

This work would not have been possible without the support, guidance, and mentorship of my advisor, Dr. John M Acken. I am forever grateful to have had the opportunity to work along side you. Thank you for believing in me and assisting me through this entire journey. You have been invaluable in shaping me as an engineer and instilling work ethics that I will continue to cherish for the remainder of my career.

Special thanks go to Dr. Robert B. Bass for seeing my potential and allowing me to take part in the Power Engineering Group. I appreciate your willingness to offer guidance and advice. Thank you for always welcoming questions and encouraging technical discussions. You are a great leader, and the EGOT DERMS project is a testament.

My appreciation goes to Professor Yuchen Huang for teaching me computer logic and guiding me through understanding advanced computer architecture. You genuinely care about your students, and it shows. Thank you for entrusting me to be a student mentor for your ECE101 class and supporting me throughout my bachelor's and master's degrees.

Thank you, Mohammed Alsaid and Midrar Adham, for recommending me for this project and for providing constant feedback and encouragement. Together we were able to build great tools and design the Distributed Trust Model. Its been an honor working alongside you.

I want to express my gratitude to Sonali Fernando for taking the lead in designing the MVoT and conducting hypothesis analysis and testing. I appreciate all your feedback and contributions on how I can improve the TMDG and TMS.

Thank you to my fellow Power Engineering Group researchers for your endless help and support. Thank you, Tylor Slay and Blue Spitzer, for helping me navigate the interworkings of the EGOT DERMS project.

Finally, my deepest and most sincere appreciation goes to my family and friends. Without your endless love and support, I wouldn't be where I am.

## Contents

## List of Tables

## List of Figures

## Acronyms

**CDTA** Central Distributed Trust Aggregator

**CISA** Cybersecurity and Infrastructure Security Agency

**CSV** Comma Separated Values

**DCM** Distributed Control Module

**DER** Distributed Energy Resource

**DOS** Denial of Service

**DTM System** Distributed Trust Model System

**DTMC** Distributed Trust Model Client

**EGoT DERMS** Energy Grid of Things Distributed Energy Resource Management System

**ESI** Energy Service Interface

**GO** Grid Operator

**GSP** Grid Service Provider

**HMI** Human Machine Interface

**IoT** Internet of Things

**MCI** Modular Communications Interface

**MITM** Man-in-The-Middle

**MVoT** Metric Vector of Trust

**P2P** Peer-to-Peer

**PLC** Programmable Logic Controllere

**RFC** Relative Factor of Certainty

**SCADA** Supervisory Control and Data Acquisition

**SDTT** Standard Deviation of Transit Time

**SPC** Service Provisioning Customer

**TM** Trust Model

**TMDG** Trust Model Data Generator

**TMS** Trust Model Simulator

**UTM** Unified Trust Model

**WSN** Wireless Sensor Networks

# 1 Introduction

With the exponential increase in computing performance, simulators and simulation frameworks have been the go-to avenue for cost-effective and time-efficient testing methods for calculations and parameters. While a plethora of simulators promise simulation on trust between entities, none have demonstrated the ability to simulate the trust model itself [1],[2]. Current implementations aim at showcasing the result of introducing a trust model to an existing network but shy away from discussing the integrity and rigidity of the trust model itself [3]. Moreover, traditional trust model simulation research often ignores a key importance of simulators, which is highlighting flaws in calculations, design, or implementation. A problem arises when discussing how one can evaluate a trust model and associated equations for correction and improvement. Therefore, a trust model simulator that allows for the trust model itself to be simulated while also displaying the potential issues in its design is valuable.

Similar to the concept of trust between humans, networks are starting to utilize trust-based interactions for data handling. While the concept of trust has been extensively studied in the context of social science and philosophy, in computing, trust remains a relatively new concept [4]. When discussing trust in the digital realm, the definition is not as forthright. Digital trust has various parameters and changes with respect to context, content, and reputation [5]. Whether a Wireless Sensor Networks (WSN), Peer-to-Peer

(P2P) networks, overlay networks, or ad-hoc networks, data transaction between nodes relies on trust to distinguish between benevolent and malicious actors. Moreover, access control and privileges are directly affected by the trust values associated with an actor; thus, good behavior is rewarded while bad behavior is punished throughout the network.

As the concept of distributed and ad-hoc networks gains popularity, there have been numerous attempts at modeling the trust between actors [6],[7]. While these simulators do a great job simulating the overall network interactions between the various actors, they lack in quantifying trust evolution through the network. Our MVoT-based distributed TMS allows for the interworking of the network to be simulated while also evaluating trust for all the participating actors.

Simulation models can be designed to closely mimic real-world environments, guiding engineers to alternative methods that would be better suited for their desired application. Said simulation models can use various parameters, variables, equations, and initial values to make a simulation abide by any set of desired constraints. Some of these parameters include registration time and time step, while some of these variables include time since last communication, communication frequency, trust score, and certainty. Our goal with the aforementioned parameters and variables is to arrive at a numerical way of representing the concept of trust between computers and actors and to provide a method of benchmarking the trust model itself.

While the proposed trust model simulator can be applied to various fields and applications, our current implementation tests the Trust Model (TM) within an EGoT DERMS.

The current implementation is incorporated in the , a compulsory module within the overall system. The main goal of the  is to augment existing security within an EGoT DERMS by monitoring exchanged messages and energy requests between the various system actors. Calculations, initial values, and alert thresholds were tested by leveraging the TMS within an EGoT DERMS.

The structure for the remainder of this paper is as follows: Chapter 2 is a background of simulators and Trust Models. Chapter 3 discusses the Energy Grid pertaining to Distributed Energy Resource Management Systems. Chapter 4 is the software decisions and implementations. Chapter 5 is the experimental setup for DTM Simulation Suit. Chapter 6 discusses the results of the DTM Simulation Suite, and Chapter 7 is the conclusion.

## 2 Background

While trust is easily defined when applied to humans and their interactions, the definition is not as clear when discussing devices and machines on a network. Trust in computing is very field-dependent. Whether in data provenance, semantic web, cloud, or mobile computing, the notion of trust is different [4]. Due to the wide variance of areas relying on trust, it is common to find field-specific simulators aiming to simulate the trust between the active devices within a network or organization.

With the explosion of Internet of Things (IoT) devices and their applications, IoT devices have become very lucrative targets for attackers. Due to the relatively large attack surface and inherently weak built-in security of IoT networks, security researchers have shed light on the threats and vulnerabilities associated with IoT networks [8, 9, 10, 11]. One mechanism to improve the overall security of an IoT network is by adding a trust element and an evaluating authority. By incorporating trust, the pre-existing security can be heightened and augmented to provide the end-users and the infrastructure more security and privacy.

P2P networks has been simulated to showcase the effect malicious nodes has on performance. A popular unstructured P2P network simulator is GIAnduia (GIA). GIA is a simulation environment based on OverSim, an open-source framework based on OMNeT++. GIA uses three main matrices to evaluate performance: Satisfaction Level, Hop Count, and Search

Bandwidth Consumption. By incorporating satisfaction level in the evaluation process, GIA assigns a satisfaction value to each participating peer that ranges between 0 and 1, with values close to 0 representing an unsatisfactory peer and values close to 1 representing a satisfactory peer. Based on the satisfaction level, the simulation aims at reconfiguring peers to connect with peers who have the highest satisfaction level. Regarding hop count, the simulation measures the distance (or the number of hops) between the source and destination nodes. Finally, the inclusion of search bandwidth consumption allows the simulator to measure the amount of traffic generated by a peer's request. Leveraging the three matrices, GIA restricts malicious nodes from participating in a P2P network and increases the overall network performance [12]. Although the three matrices used by GIA may be considered sufficient at showcasing overlay network improvements, the matrices remain limited in scope. While it was enough to show a performance increase after including a trust-based mechanism, its evaluation is not dynamic enough to provide a real-time assessment of the trust between the peers within the observed network.

WSN have also been simulated using TRMSim-WSN to understand better the evolution of trust and reputation for Wireless Sensor Networks. TRMSim-WSN leverages the Unified Trust Model (UTM) [13] to perform its trust evaluation. UTM consists of five main layers. It starts by gaining information about the participating sensors, what they offer and what they can request. Based on the information it acquires, the model creates a ranked table. The table is ranked in terms of sensor trustworthiness. TRMSim-WSN then monitors the interaction being conducted between the various components and either punishes or rewards

the components based on the feedback they receive. When evaluating the trustworthiness of a sensor, UTM incorporates the following metrics: history, recommendation, context, and platform properties. Through their experimental results, the authors concluded that the UTM-enabled settings had higher levels of accuracy but had longer path lengths [14]. While TRMSim-WSN provides its users with a way of testing and comparing different models [15, 16, 17, 18], it lacks the ability to test the trust model itself and focuses on the trust relationship between wireless sensors within a network.

DTMSim-IoT is a .NET-based simulator that calculates trust based on each observed interaction. DTMSim-IoT consists of seven sub-modules: Network Configuration, Network Initialization, Service Request, Service Evaluation, Trust Computation, Simulator Logs, and Result Data Export. The Network Configuration stage allows the simulator users to specify basic configurations such as the total number of nodes, the trust model selection, and time delay. The Trust Computation Module follows a Reward and Punishment mechanism. For each successful service provision, the actor's reward is calculated by one multiplied by the service weight (Ws), while negative two multiplied by Ws calculate failed service transactions. The overall trust value considers both the reward and punishment values and is a value that ranges from -1 to 1. Close to 1 represents a cooperating node, while trust values near -1 indicate a malicious node [19].

While there are a plethora of trust simulators, most aim to simulate the relationship and trust between participating devices or components. The trust model simulator is a field-specific simulator that aims to showcase trust between actors within an EGoT DERMS network. In

addition, it provides a way to test the trust model itself while simultaneously verifying and testing the calculations being developed.

# 3 Energy Grid of Things Distributed Energy Resource Management System

## 3.1 Overview

Traditional power grids are viewed as a single-stream service, i.e., an energy service user requesting power from an energy service provider. A newer and smarter energy grid system is proposed. This more technologically advanced and distributed energy grid is an EGoT DERMS. EGoT DERMS offers a method for exchanging information and resources between participating actors to provide essential reliability services, which support large-scale deployment of renewable generation and electrification of loads. At the heart of the EGoT DERMS is the Energy Service Interface (ESI). The ESI governs the information exchange between GSP and SPC through a set of policies. Moreover, the ESI uses the information exchanged between GSP and SPC to dispatch essential reliability services through large-scale aggregation of Distributed Energy Resource (DER). The ESI defines a bi-directional, service-oriented, logical interface that defines system security, privacy, and trust. The DTM System is incorporated within the EGoT DERMS to augment the system's existing security measures. The DTM System serves as a detective component within the EGoT DERMS to detect system anomalies. Figure 3.1 showcases the overall EGoT DERMS. EGoT DERMS actors are shown in blue, while the DTM System components are shown in red. The DTM System consists of multiple Distributed Trust Model Clients

(DTMCs) and a CDTA. The various components of the EGoT DERMS will be discussed in further detail in the upcoming sub-sections [20, 21, 22].

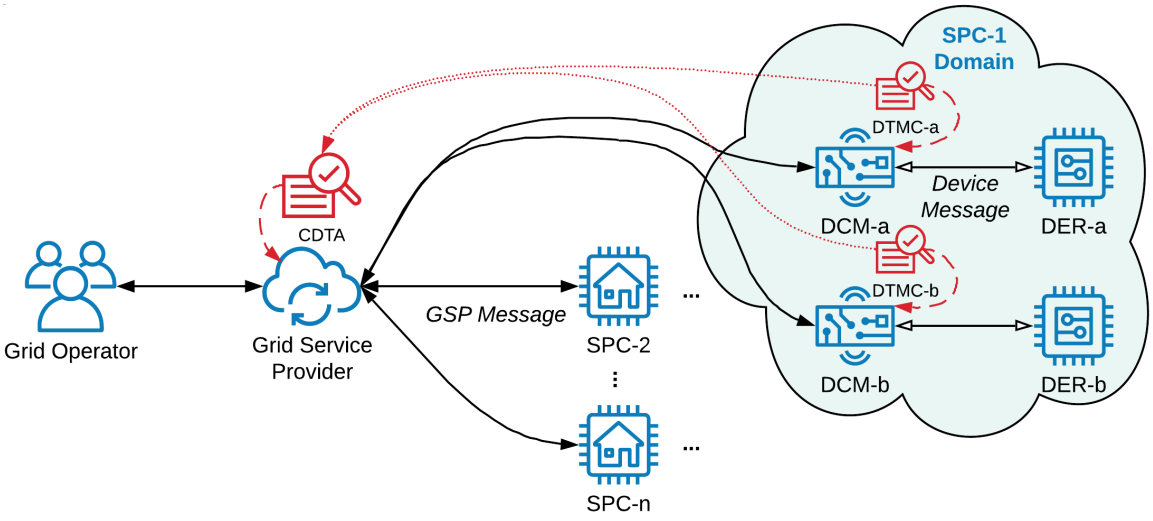


Figure 3.1: An overview of EGoT DERMS architecture.

## 3.2 EGoT Infrastructure

As briefly discussed in the earlier section, an EGoT DERMS is comprised of multiple actors working together to facilitate the transfer of information and energy services throughout the system. In this section, the responsibilities and functionalities of each of the participating actors within an EGoT DERMS will be further discussed. Figure 3.1 demonstrates the overall infrastructure of an EGoT DERMS and the connections between the actors.

### 3.2.1 Grid Operator

The Grid Operator (GO) communicates with the GSP to acquire grid services to achieve operational objectives. Operational objectives include maintaining operations within pre-defined constraints to prevent grid component and/or equipment damage.

### 3.2.2 Grid Service Provider

A GSP provides grid services to a GO through the dispatch of DER that have subscribed to respond to a GO program or mandate. Grid services are the means by which a GO achieves operational objectives.

### 3.2.3 Service Provisioning Customer

The SPC is a residential electric utility customer who owns one or moreDERs. The SPC is interested in providing those DERs to a GSP through an aggregation program.

### 3.2.4 Distributed Trust Model System

Figure 3.1 shows the DTM System components in red. As mentioned earlier, the DTM System consists of multiple DTM Systems and a single CDTA. Each of the DTMCs is located at an SPC and provides trust evaluations to the various actors connected to that SPC. The DTMCs aggregate the MVoT parameters for each actor, then sends the MVoTs to the CDTA. The CDTA is located at the GSP and stores the MVoTs of all the actors within the DTM System.

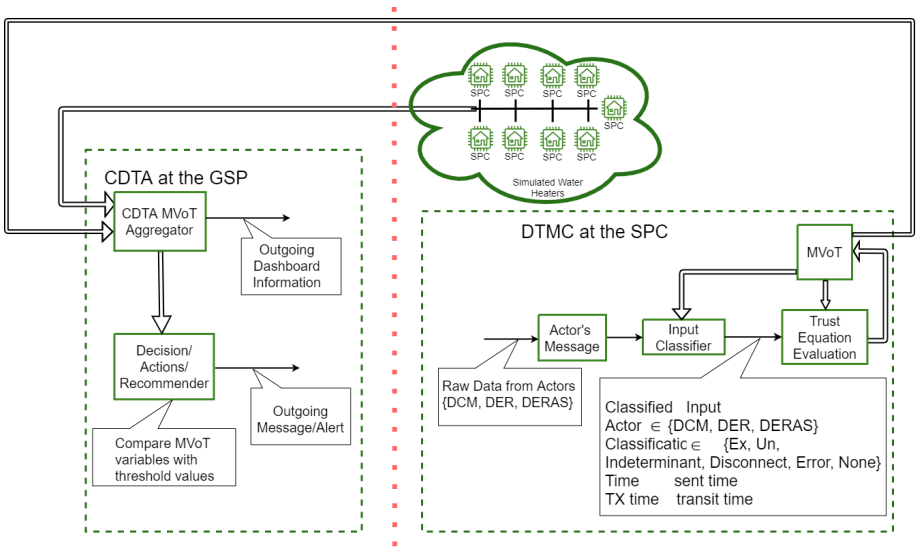


Figure 3.2: An overview of DTM System components with the DTMC at each SPC (right side) and CDTA at the GSP (left side).

Figure 3.2 displays the mechanics of the DTM System in more detail. On the left side of the dotted red line are the inter-workings of the DTM System at the SPC. The DTMC classifies the incoming actor messages to arrive at a message evaluation category for each of the messages. The MVoT is updated based on the message evaluation category. On the right side of the figure is the CDTA at the GSP. The CDTA responsibilities include providing recommendations to the GSP and providing the data needed to create dashboards for the GO.

## 3.3 Information Exchange

### 3.3.1 System Protocols

The EGoT DERMS primarily relies on two protocols to facilitate the transfer of information and energy resources: IEEE 2030.5 and CTA-2045. The IEEE2030.5 standard is the

Smart Energy Profile protocol. The IEEE 2030.5 standard defines a protocol for smart grid communication to facilitate smart metering, demand/ response automation, and load control [23].

CTA-2045 is a standard that defines a Modular Communications Interface (MCI) to allow for energy management functionality in residential devices. Through the MCI, CTA-2045 provides a standard interface for communicating with user-end devices. [24]

### 3.3.2 Actor Infomration Exchange

To facilitate the communication between the various actors within an EGoT DERMS, the EGoT DERMS architecture specifies three main information exchange phases. The first phase is the GO to GSP information exchange. GO to GSP communications are defined according to the GO's requirements. Given the scope of this paper, we will not be discussing GO to GSP communications in greater detail. The second phase is GSP to Distributed Control Module (DCM) information exchange. GSP to DCM communications strictly adheres to the IEEE 2030.5 protocol for information exchange and service requests. The final stage is DCM to DER information exchange. DCM to DER communications can follow several DER-specific protocols. Common DER protocols include CTA-2045 [24], SunSpec Modbus [25], and J3072 [26].

## 3.4 Critical Infrastructure Attacks

Critical infrastructure attacks are cyberattacks that target systems or assets vital to a nation. According to the Cybersecurity and Infrastructure Security Agency (CISA), there are 16 crit-

ical infrastructure sectors that include: Defence Industrial Base Sector, Emergency Service Sector, Energy Sector, Financial Services Sector, Healthcare, and Public Health Sector, and Water and Wastewater System Sector. An attack to any of the critical infrastructure sectors would have a drastic impact on a nation's security [27].

### 3.4.1 Attack Methodology

Attacks on a nation-state often target specific components and devices. Whether Supervisory Control and Data Acquisition (SCADA), Human Machine Interface (HMI), or Programmable Logic Controllere (PLC), the components within the system can be used as a launch pad for attackers. Like the US military kill chain, cyber attacks follow a set of stages and steps. The phases of the cyber attack chain include:

- Reconnaissance - A research phase on the target to identify possible vulnerabilities.
- Weaponization - Malware development to exploit the vulnerability found during the reconnaissance stage.
- Delivery - Transmission of the developed malware onto the target's machine/ system
- Exploitation - Triggering of the malware to exploit the known vulnerability
- Installation - The malware installs backdoors to allow for persistent access to the victim's machine/ system
- Command & Control - The attack gains persistent access via an outside server

- Action on Objective - The attack works toward their end goal (data theft, exfiltration, data destruction)

### 3.4.2 Threat Models

Multiple risk assessment methods use threat and vulnerability modeling schemes to better understand the impact of threats and provide an approach for mitigating them. The goal of using a threat model differs depending on the field and environment. For smart power grids, integrity and availability are the highest priority. One of the standard methods of ensuring integrity and availability comes with Anomaly Detection. By constructing a model for what "normal" and "expected" traffic look like, a normative model can assist in identifying abnormal cases [28].

One prominent and effective attack on power grids is a Denial of Service (DOS) attack. DOS attacks target grid availability rather than the integrity of the service requests propagating through the system. To create a threat model for a DOS attack, it is important to understand the nature and goal of such an attack. The goal of a DOS attack is to drown the target with a large amount of requests so that legitimate requests are prevented from accessing the system services [29]. In the context of EGoT DERMS, a DOS attack aims at negating the authorized requests of EGoT DERMS actors, effectively bringing the system to a halt. Developing a threat model for EGoT DERMS DOS attacks consists of understanding the data source, the request type, and network bandwidth consumption in the form of communication frequency.

Another possible attack on power grids is a Man-in-The-Middle (MITM) Attack. Unlike a DOS attack, a MITM attack aims at violating the integrity of requests within the system. A MITM attack can take the form of legitimate communication between two EGoT DERMS actors being intercepted and propagated by an actor; this is referred to as traffic leading. Another form of MITM attack can be done by impersonating one or more system actors. This is referred to as identify spoofing [30]. Regarding the EGoT DERMS, a MITM attack aims to intercept legitimate EGoT DERMS messages and modify the messages before delivering them to the expected receiver. Developing a threat model for the EGoT DERMS MITM attack consists of understanding the communication standards throughout the systems and the registration process for new actors.

Developing the various components that make up the EGoT DERMS is no small feat. The system is complex and follows multiple protocols that intertwine and work together. At any given moment, multiple actors could exchange hundreds of messages and service requests. To ensure the stability and availability of the infrastructure of EGoT DERMS, the DTM System is incorporated. The goal of the DTM System is to add to the existing system security. The DTM System and the tools used to develop it are described in greater detail in the upcoming chapter.

# 4 Software Description

The DTM System is a component within the overarching EGoT DERMS project. The sole purpose of the DTM System is to augment existing security measures by monitoring the communication between the various participating EGoT DERMS actors, as shown in Figure 4.1.

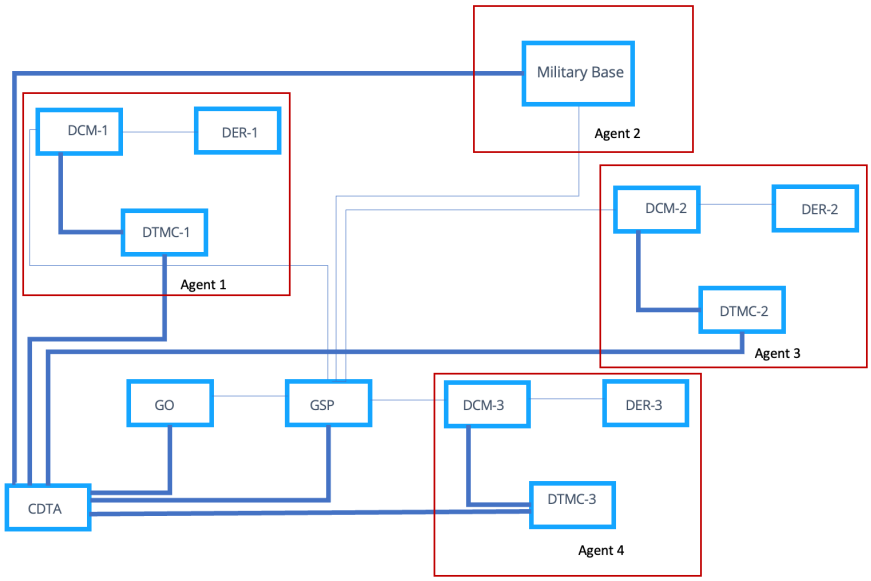


Figure 4.1: A DTM System example with multiple DTMCs aggregating agent data into a CDTA.

The DTM System plays a detective rather than a preventative role within the overall system and can provide recommendations to the service provider. DTM Systems can be distributed across the system to monitor subgroups and aggregate their findings to the CDTA. The DTM System uses a MVoT to represent the evaluations of each participating actor within

EGoT DERMS [31]. The MVoT consists of 17 parameters that include Trust Score (TS), Distrust Score (DS), Certainty (C). The complete list of MVoT variables and descriptions can be found in Table 4.1. The rest of the participating actors include GO, GSP, DCM, and DER.

| **Abbreviation** | **Variable** | **Description** |
|---|---|---|
| TS | Trust Score | Overall trust score for each actor |
| DS | Distrust Score | Distrust score for each actor |
| C | Certainty | How certain is the DTM System for each evaluation |
| CExMsg | Count of Expected Messages | Total count of messages that are expected for each actor |
| CUxMsg | Count of Unexpected Messages | Total count of messages that are unexpected for each actor |
| TotMsg | Total Number of Messages | Count of Total messages |
| Time_Stmp | Time of the Last Message Received | Time of the most recent message received from the actor |
| Regstr_Time | Registration Date | Time of the first message received from an actor |
| ComFreq | Registration Date | How often an actor communicates |
| TX_Time | Measured Transit Time | Time difference for message to travel from the source to destination |
| Avg_TX_Time | Average Transaction Time | Expected transaction time is average transaction time |
| TSLC | Time Since Last Communication | Time delta of the last message is received |
| SDTT | Standard Deviation of TX Time | Extent of deviation for Transit time as a whole |
| RFC | Relative Factor of Certainty | Certainty indicator of lean toward or against TS or DS |
| T_Out | Count of Timeouts | Total count of timeouts for each actor |
| C_Alrt | Count of Alerts | Total count of alerts sent out to each actor |
| C_Other | Count of Other Actions | Total count of disconnects and additional actions sent out to actors |

Table 4.1: MVoT variables and descriptions.

Figure 4.2 shows the overall DTM software suite. The process can be started either

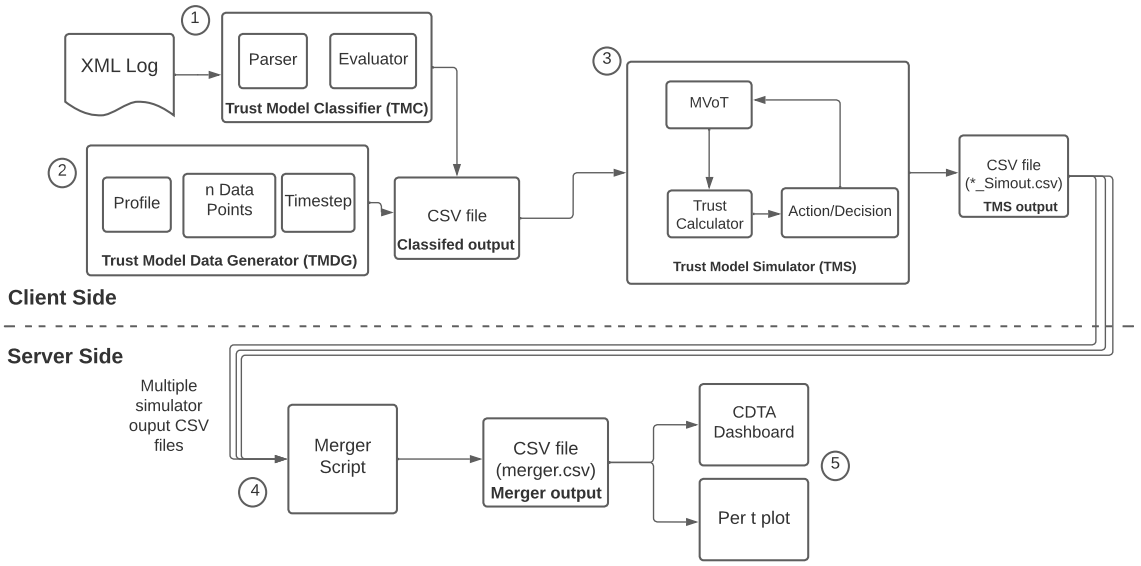


Figure 4.2: An overview of the 5 stages of the DTM Simulation Suite with the components and interconnections

by reading and classifying an XML log, as shown in stage 1, or by generating a data set using the TMDG, as shown in stage 2. Regardless of which of the two options is used to get started, the output will be a Comma Separated Values (CSV) file with the classified output. The classified CSV file can then be read into the TMS to run the trust model and update the MVoT based on the trust calculation per read message, as shown in stage 3. Once the simulator runs through the entire dataset, a CSV file with the suffix simout.csv is created with each row in the file presenting the actor, evaluation category, and an updated MVoT. Finally, to simulate the CDTA, multiple iterations of the simulator must be run to produce two or more simout.csv files. All the TMS output CSV files are then fed into the merger script, which orders the incoming messages based on their timestamp, as shown in stage 4. This process is used to mimic the functionality of the CDTA since it will be observing all the aggregated messages and storing them in the order in which it receives them. The output of the merger script is a single CSV file that includes every MVoT row from each of the

funneled in simout.csv files. Using the merger.csv file, plots and dashboards are produced using either the perTplot script or the CDTA Dash script, as shown in stage 5. Both scripts provide the user with a list of supported MVoT parameters and time increments. The scripts produce a CSV file that includes the x,y data points to generate either a plot or dashboard. Stages 1-3 simulate a DTMC at the client, while stages 4 and 5 simulate the CDTA at the server side.

To aid in the development of the DTM System, both a TMS and TMDG were developed. The goal of both the TMS and TMDG is to readily generate and simulate messages and requests from the various EGoT DERMS actors. The TMDG allows the user to generate data for the simulator run. The generator provides the user with a way to set the number of data points and profiles that influence the nature of the generated data. Some of these profiles can generate all expected messages, while others can allow the user to manually set the evaluation category of a specific message. The output of the TMDG is a CSV file that is the input to the TMS. The TMS provides the means to test and verify MVoT calculations, initial values, and various thresholds. The TMS takes in the CSV file generated by the TMDG and runs through the various messages while updating the MVoT of each of the participating actors. Once completed, the values of each of the MVoTs for each actor are then written out to a CSV file. The TMS showcases the trust relationship between the EGoT DERMS actors over time while also providing accuracy and performance metrics to understand the trust algorithm better.

Various components facilitate test data creation and simulation. The TMDG is in place

to replace real-time data, while the TMS is used to test and tweak the TM component in the DTM System. The DTM System system consists of five main components: classifier, MVoT calculator, CDTA, recommender, and dashboard. The role of the classifier is to characterize incoming messages and produce an evaluation for each message. The MVoT calculator updates each MVoT parameter based on the evaluation category of the incoming message. The CDTA receives MVoT values from the DTMCs and calculates periodic normalized values. The recommender sends alerts and recommendations to GSP based on the set thresholds. Finally, the dashboard presents the aggregated MVoT data to the GSP.

The Trust Model testing suite starts with the TMDG, where a user can generate CSV files to reflect a specific messaging pattern. The generated CSV file includes the actor name, the message evaluation category, current time, and transit time. The TMS then reads in the CSV file as an input and then runs through the rows of the CSV file. Based on the message evaluation of each actor, the MVoT entry for each actor is updated appropriately. Once the simulator completes reading the entire data set, a new CSV file is generated. The new CSV file includes the same information as the CSV file produced by the TMDG plus the updated MVoT of each of the actors. A merger script mimics the functionality of the CDTA. The goal of the merger script is to aggregate multiple TMS outputs into a single CSV file. The output of the merger script is a CSV file that includes all the actors and their MVoT ranked based on their time of arrival at the CDTA. The merger CSV file then generates plots per some user-defined time increment or generate the CDTA plots used in the dashboard.

# 5 Experimental Setup

The DTM System is composed of many different blocks and stages. Figure 5.1 showcases the relationship between the TMDG and the TMS. Moreover, Figure 5.1 presents the different blocks and stages within the DTM System. First, a CSV file is read within the Trust Model Simulator; the CSV file is produced by either the generator or the classifier. Next, its content is passed on to the Trust Calculation block. If the incoming message is the first from the actor, a new trust vector is created for the actor. If the incoming message is not the first message, the existing trust vector for the actor is updated to include a new entry. Once the trust calculations are completed, the updated MVoT parameters are then stored in the MVoT block. The trust calculation output is also passed to the Threshold block. Within the Threshold block, the incoming trust results are compared against various limits. The Decision block then processes the output of the threshold comparison to arrive at the appropriate action based on the provided threshold information. Finally, a recommendation is passed to the Action block, where the relevant authorities receive a recommendation or alert based on the information gathered throughout the TMS process. The type of the alert is also stored in the MVoT for each actor to influence future recommendations.

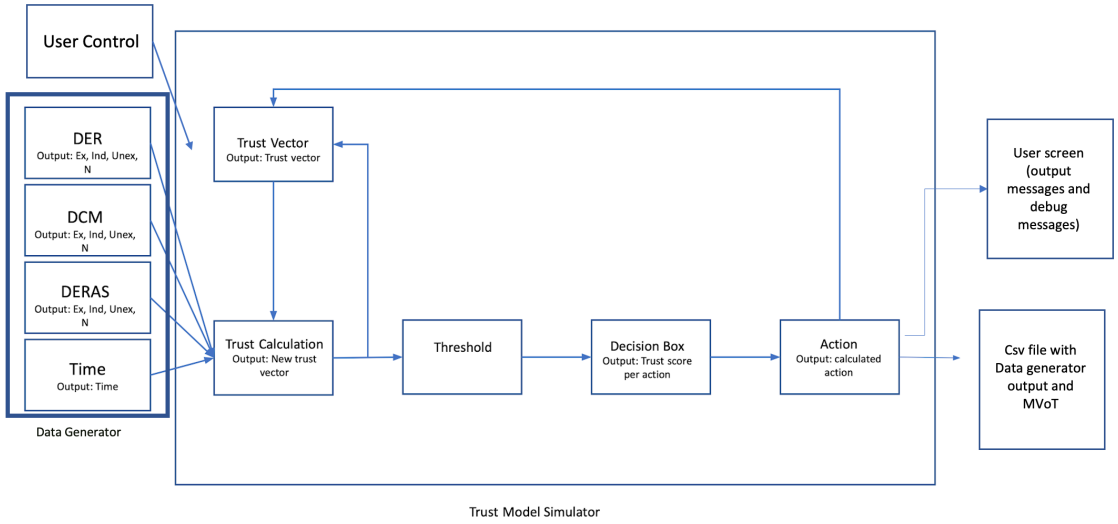


Figure 5.1: An overview of TMDG and TMS interconnection and component blocks.

The MVoT-based TMS can be run using two different input streams. The first input is by reading and parsing GSP, DCM, and DER logs. Once the logs of each of the participating actors are parsed, an evaluation category is associated with each of the messages, and then the trust calculations can begin. A list of evaluation categories and their descriptions is presented in Table 5.1.

| **Evaluation** | **Description** |
|---|---|
| Expected (Ex) | All message contents are presented, valid, and follow the correct order. |
| Unexpected (Ux) | The message is repetitive, out-of-order, or contains extreme values. |
| Indeterminant (Ind) | The DTM System is unable to classify the message. |
| Discconect (Dis) | The device is nonresponsive. |
| Error (Err) | The message content is incorrect or missing. |
| None (N) | The message does not fit any classification. |

Table 5.1: Evaluation categories and descriptions.

The second method to run the TMS uses the TMDG to create a CSV file that includes a list of actors, message evaluation categories, time stamps, and transit time. The TMS can read in this CSV file, and the simulator will go through the messages and run through

the process until it arrives at a recommendation. While the two processes appear similar, they mimic and test different parts of the DTM System. The first implementation tests the classification portion of the system. This part of the system is considered more realistic since the DTM is conducting the classification on legitimate logs. Here, the DTM reads actual XML logs and parses them to extract the necessary information to provide an evaluation. The second implementation rapidly tests the MVoT, thresholds, and recommendation messages. The generator gives users the ability to mass-produce data sets aimed at testing various aspects of the DTM System while still being time-efficient. The input format of the TMS can differ depending on the path used to run the simulator; however, the information required to run the simulator is consistent regardless of the input stream. Whether an XML log or a generated CSV file, the necessary information to get the simulation started is the same. The simulator input must contain the actor, the target or evaluation category, and a time stamp. Other information such as transit time and HTML hrefs can provide a better assessment but are not necessary.

As a part of the simulator process, and depending on the message evaluation catergory associated with each incoming message, the MVoT parameters are updated. A description of all 17 MVoT variables are briefly described below.

## 5.1 Trust Score (TS)

The Trust Score calculation, presented in Equation 5.1, shows how the overall trust of an actor is derived. The TS calculation takes into account multiple factors and parameters. Some obvious factors are the number of Expected Messages (ExMsg) and a weighted ($\alpha$) number of Unexpected Messages (UnMsg). $\alpha$ is used to determine the impact negative messages have on the overall TS score. Certainty (C) is incorporated into the TS calculation to represent the DTM confidence value regarding the actor in question. The confidence calculation is made up of multiple parameters and will be discussed in further detail in Equation 5.3.

$$TS(i) = [CExMSG(i) - (\alpha \times CUnMSG(i))] \times C(i) \tag{5.1}$$

## 5.2 Distrust Score (DS)

Many existing trust models arrive at a single trust score. This creates a potential problem because an indication of an anomaly due to one factor can be hidden or overwhelmed by the effects of other variables. The attacker can take advantage of this concept by loading some false messages to influence the overall score. To help avoid this, we keep a separate distrust score (DS) to flag or track suspicious activity. Unlike Equation 5.1, Distrust Score quantifies how untrustworthy an actor is. DS considers the number of Unexpected Messages (UnMsg) and Certainty (C).

$$DS(i) = CUnMSG(i) \times C(i) \tag{5.2}$$

### 5.3 Certainty (C)

Certainty plays a critical role in evaluating the overall trust. The DTM System represents the confidence of each actor within the DTM System using a Certainty score. Certainty is a function of an actor's Relative Factor of Certainty, $1 - e^{(-\gamma \times TotMsg)}$, normalized Communication Frequency (ComFreq), and normalized Time Since Last Communication (TSLC). Gamma ($\gamma$) in $1 - e^{(-\gamma \times TotMsg)}$ is a weighting value used to dictate the influence the Total Message Count (TotMsg) has on Certainty (C).

$$C(i) = (RFC \times (1 - e^{(-\gamma \times TotMsg)} \times \frac{ComFreq}{max\_ComFreq})) \times \frac{min\_TSLC}{TSLC} \tag{5.3}$$

### 5.4 Count of Expected Messages (CExMsg

Count of Expected Messages is used by multiple MVoT parameters to gauge an actor's expected communication history. The DTM System increments CExMsg based on its evaluation of the incoming messages being expected.

$$CExMsg(i) = CExMsg(i-1) + 1 \tag{5.4}$$

### 5.5 Count of Unexpected Messages (CUxMsg)

Count of Unexpected Messages is used by multiple MVoT parameters to gauge an actor's unexpected communication history. The DTM System increments CUxMsg based on its evaluation of the incoming messages being unexpected. A relatively high number of Unexpected Messages could signal to the DTM System that an actor is acting maliciously

resulting in the actor being flagged for suspicious activity.

$$CUxMsg(i) = CUxMsg(i-1) + 1 \tag{5.5}$$

### 5.6 Count of Total Messages (TotMsg)

Count of Total Messages is used by the DTM to keep a count of all incoming messages regardless of their evaluation category.

$$TotMsg(i) = TotMsg(i-1) + 1 \tag{5.6}$$

### 5.7 Time Stamp (Time_Stmp)

As messages traverse through EGoT, the DTM updates the Time_Stmp parameter for each actor's message to keep track of the time in which the message was received.

$$Time\_Stmp(i) = Current\ time\ in\ Unix\ time \tag{5.7}$$

### 5.8 Registration Time (Regstr_Time)

Registration Time is the Time_Stmp for when an actor sends their first message. The use for Regstr_Time is to quantify how long an actor has been participating in grid services

$$Time\_Stmp(i) = Time\_Stmp(0) \tag{5.8}$$

### 5.9 Communication Frequency (ComFreq)

The Communication Frequency calculation, in Equation 5.9, provides the frequency at which an actor is requesting grid services. ComFreq is used to identify drastic shifts in an

actor's communication rate.

$$ComFreq = \frac{TotMsg}{Current\,Time - Regstr\_Time} \tag{5.9}$$

### 5.10 Transit Time (TX_Time) and Average Transit Time (Avg_TX_Time)

Transit Time is used by the DTM System to understand the time it take for a message to propagate from the sending actor to the receiving actor. Equation 5.10 looks at the Average Transit Time, which represents the mean value of message transaction time. Based on the calculated values, the DTM can evaluate whether or not the message's transit time is expected or not.

$$\mu_n = \frac{1}{n}\sum_{i=1}^{n}(x_i) \tag{5.10}$$

### 5.11 Time Since Last Communication (TSLC)

Equation 5.10 describes the time between an actor's messages. TSLC is used to understand how active/unactive a user is and identify any anomalies associated with their communication patterns.

$$TSLC = \Delta time\ of\ last\ message\ received \tag{5.11}$$

### 5.12 Standard Deviation of Transit Time (SDTT)

Equation 5.12 is for the Standard Deviation of Transit Time (SDTT). SDTT is updated with every new incoming message from an actor.

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_n)^2} \tag{5.12}$$

### 5.13 Relative Factor of Cerainty RFC

Equation 5.13 refers to the Relative Factor for Certainty (RFC). RFC measures the balance between expected and unexpected messages. The relative certainty of an evaluation (for example, whether to send a message or not) is influenced by how heavily the data leans towards either expected or unexpected. The $\beta$ sets the maximum value of what RFC can be. For example, if $\beta$ = 2, then the maximum RFC value is 1.0.

$$RFC = |\frac{CExMsg}{CExMsg + CUnMsg} - 0.5| \times \beta \tag{5.13}$$

### 5.14 Count Of Timeouts (C_Out)

Equation 5.14 refers to the count of timeouts associated with an actor. Timeouts can be caused by an ignored heartbeat signal or an actor unexpectedly disconnecting from the system.

$$T\_Out(i) = T\_Out(i-1) + 1 \tag{5.14}$$

### 5.15 Count Of Alerts (C_Alert)

Equation 5.15 refers to the count of alerts associated with an actor. Alerts are generated for a wide range of reasons. A list of alerts can be found in the CDTA discussion in Chapter 4.

$$C\_Alrt(i) = C\_Alrt(i-1) + 1 \tag{5.15}$$

### 5.16 Count Of Other Actions (C_Other)

Equation 5.16 refers to the count of other alerts associated with an actor. Count Of Other Actions is reserved for additional actions that may be introduced in the future.

$$C\_Other(i) = C\_Other(i-1) + 1 \tag{5.16}$$

| Parameter | Description |
|---|---|
| $\alpha$ | The weighting factor of trust and distrust |
| $\gamma$ | Sets the maximum value of what RFC can be |
| $\beta$ | The weighting factor dictating the influence messaging count has on certainty |

Table 5.2: MVoT parameters and descriptions.

Table 5.2 presents the parameters used in some of the MVoT equations. One of the purposes of the TMS is to evaluate the values associated with the parameters. Moreover, the TMS was used to fine-tune parameter effects on MVoT variables.

Once the simulation is completed, the CDTA assesses and evaluates the MVoT values for each DTMC. The MVoTs for each of the actors are then compared against thresholds and other actors' MVoTs to arrive at a recommendation. The DTM System uses the recommendations to alert the appropriate personnel of abnormal activities detected by the CDTA.

Alerts and recommendations include:

- "Excessive time since last communication from DER"
- "Trust is low for GSP/DCM/DER"

- "Communication rate is low from GSP/DCM/DER"

Due to the limited data available and the need for readily available data sets, the Trust Model Data Generator was developed. The TMDG supports multiple profiles and a user-specified number of data points and time increments.

Developed profiles include:

- All expected (Ex): This profile generates only expected data
- All unexpected (Ux): This profile generates only unexpected data
- Almost good (Ex and Ux): This profile generates expected messages until a user-specified threshold is reached, in which case it switches to unexpected messages.
- Almost bad (Ex and Ux): This profile generates unexpected messages until a user-specified threshold is reached, in which case it switches to expected messages.
- Random (Ex, Ux, Ind, Err, Dis, and None) : This profile assigns a random message evaluation category to a random actor.
- Mixed (Ex, Ux, Ind, Err, Dis, and None): This profile allows the user to specify two or more of the supported profiles.
- User specified (Ex, Ux, Ind, Err, Dis, and None): This profile allows the user to manually set the message evaluation category for an actor.

The output of the TMDG is a CSV file that includes actor name, message evaluation category, transit time, and the current time. The TMDG CSV file is then used as the input for the Trust

Model Simulator and to populate the MVoTs associated with each of the participating actors. There are various calculations for calculating MVoT values rather than a single value. The MVoT and output CSV files from the TMDG and their role throughout the DTM System are discussed in the upcoming chapter.

# 6 Experimental Results

Throughout this thesis, the TMDG, TMS, CDTA plotting tools, and DTM Classifier were used to help design and fine-tune the DTM System. As mentioned in the previous chapters, the limited "real" data available required a tool that would readily create testing data leading to the development of the TMDG and DTM Classifier. Moreover, the unique design and functionality of the DTM System by using an MVoT required the TMS to verify the calculations, initial values, and thresholds. The DTM System and CDTA demonstrate the aggregation of data and the sending of recommendation messages/ alerts. Moreover, the CDTA plotting tools provide the required data points to create the CDTA dashboard.

## 6.1 Trust Model Data Generator

The Trust Model Data generator is a Python program that takes in user input as parameters and generates a data set based on the user input. The output of the generator is a CSV file with the contents shown in Table 6.1.

| Actor | Message Eval Catagory | Current Time | Transit Time | Attack Status |
|---|---|---|---|---|
| DER | Ex | 1646183942 | 0.22 | No Attack |
| DCM | Err | 1646183952 | 0.24 | No Attack |
| DERAS | Ux | 1646183962 | 0.41 | Attack |
| DTM | Dis | 1646183972 | 0.45 | No Attack |
| DER | Er | 1646183982 | 0.24 | No Attack |
| DCM | Ex | 1646183992 | 0.19 | No Attack |
| DERAS | Ex | 1646184002 | 0.35 | No Attack |
| DTM | Ind | 1646184012 | 0.38 | Attack |
| DER | Ex | 1646184022 | 0.32 | No Attack |

Table 6.1: An example of a TMDG output showing a random dataset with actors communicating every 10 seconds.

Table 6.1 shows the output of the TMDG. The first column lists participating actors; this list includes the DER, DCM, DERAS, and DTM. The second column is the message evaluation category associated with each message. The possible categories are Expected (Ex), Unexpected (Ux), Indeterminate (Ind), Disconnected (Dis), Error (Err), and None (N). The third column is when the message was sent in Unix Epoch time (Seconds since January 1st, 1970 UTC). The fourth column is the transit time for the message. Given that this is an artificially generated dataset, the transit time is a random time between 150 ms and 450 ms. Finally, the last column is the attack status. The generator can generate a data set with messages pre-labeled as attacks for testing purposes.

## 6.2 Trust Model Simulator

The Trust Model Simulator reads in a CSV file containing EGoT DERMS actors and their message evaluation categories. The TMS is built on the same Trust Model engine in the DTM System. As a result, the TMS uses the same initial values, thresholds, and MVoT

calculations. As each message is received, the TMS updates all the MVoT parameters. Once the simulation is completed, a CSV file is created showing the input to the simulator and the MVoT values for all the parameters per message. An example of the simulator output CSV file can be seen in Table 6.1.

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Actor | Message Eval Catagory | Current Time | Transit Time | trust score | distrust score | certainty | relative factor of certainty | expected message count | unexpected message count | total message count | time stamp | reg date | comm. frequency | message transit time | average transit time | time since last comm. | timeout count | alert count | other action count | SD transit time | Max CommFreq | Min TSLC | DTM response |
| DER | Ex | 1648428392 | 0.21 | 0.1 | 0 | 0.1 | 1 | 1 | 0 | 1 | 1.6E+09 | 1.6E+09 | 0.001 | 0.21 | 0.105 | 0 | 0 | 0 | 0 | 0.246779 | 0 | 100000 | Do nothing |
| DCM | Ex | 1648428992 | 0.16 | 0.1 | 0 | 0.1 | 1 | 1 | 0 | 1 | 1.6E+09 | 1.6E+09 | 0.001 | 0.16 | 0.053333 | 0 | 0 | 0 | 0 | 0.117757 | 0 | 100000 | Do nothing |
| DERAS | Ex | 1648429592 | 0.21 | 0.1 | 0 | 0.1 | 1 | 1 | 0 | 1 | 1.6E+09 | 1.6E+09 | 0.001 | 0.21 | 0.0525 | 0 | 0 | 0 | 0 | 0.052915 | 0 | 100000 | Do nothing |
| DTM | Ex | 1648430192 | 0.39 | 0.1 | 0 | 0.1 | 1 | 1 | 0 | 1 | 1.6E+09 | 1.6E+09 | 0.001 | 0.39 | 0.078 | 0 | 0 | 0 | 0 | 0.136107 | 0 | 100000 | Do nothing |
| DER | Ex | 1648430792 | 0.42 | 0.2 | 0 | 0.0793 | 1 | 2 | 0 | 2 | 1.6E+09 | 1.6E+09 | 0.0008333 | 0.42 | 0.1575 | 2400.00002 | 0 | 0 | 0 | 0.070993 | 0.001 | 2400 | Do nothing |
| DCM | Ex | 1648431392 | 0.39 | 0.2 | 0 | 0.0793 | 1 | 2 | 0 | 2 | 1.6E+09 | 1.6E+09 | 0.0008333 | 0.39 | 0.101429 | 2400.00001 | 0 | 0 | 0 | 0.070559 | 0.001 | 2400 | Do nothing |
| DERAS | Ex | 1648431992 | 0.22 | 0.2 | 0 | 0.0793 | 1 | 2 | 0 | 2 | 1.6E+09 | 1.6E+09 | 0.0008333 | 0.22 | 0.073438 | 2400.00001 | 0 | 0 | 0 | 0.072297 | 0.001 | 2400 | Do nothing |
| DTM | Ex | 1648432592 | 0.24 | 0.2 | 0 | 0.0793 | 1 | 2 | 0 | 2 | 1.6E+09 | 1.6E+09 | 0.0008333 | 0.24 | 0.096 | 2400.00001 | 0 | 0 | 0 | 0.084499 | 0.001 | 2400 | Do nothing |
| DER | Ex | 1648433192 | 0.37 | 0.23791 | 0 | 0.08706 | 1 | 3 | 0 | 3 | 1.6E+09 | 1.6E+09 | 0.000625 | 0.37 | 0.17875 | 2400.00001 | 0 | 0 | 0 | 0.105475 | 0.001 | 2400 | Do nothing |
| DCM | Ex | 1648433792 | 0.37 | 0.23791 | 0 | 0.08706 | 1 | 3 | 0 | 3 | 1.6E+09 | 1.6E+09 | 0.000625 | 0.37 | 0.125844 | 2400.00001 | 0 | 0 | 0 | 0.051452 | 0.001 | 2400 | Do nothing |
| DERAS | Ex | 1648434392 | 0.33 | 0.23791 | 0 | 0.08706 | 1 | 3 | 0 | 3 | 1.6E+09 | 1.6E+09 | 0.000625 | 0.33 | 0.094818 | 2400.00001 | 0 | 0 | 0 | 0.03038 | 0.001 | 2400 | Do nothing |
| DTM | Ex | 1648434992 | 0.18 | 0.23791 | 0 | 0.08706 | 1 | 3 | 0 | 3 | 1.6E+09 | 1.6E+09 | 0.000625 | 0.18 | 0.102462 | 2400.00001 | 0 | 0 | 0 | 0.085314 | 0.001 | 2400 | Do nothing |
| DER | Ex | 1648435592 | 0.44 | 0.34823 | 0 | 0.10071 | 1 | 4 | 0 | 4 | 1.6E+09 | 1.6E+09 | 0.0005556 | 0.44 | 0.197411 | 2400.00001 | 0 | 0 | 0 | 0.079644 | 0.001 | 2400 | Do nothing |
| DCM | Ex | 1648436192 | 0.31 | 0.34823 | 0 | 0.10071 | 1 | 4 | 0 | 4 | 1.6E+09 | 1.6E+09 | 0.0005556 | 0.31 | 0.138121 | 2400.00001 | 0 | 0 | 0 | 0.076079 | 0.001 | 2400 | Do nothing |
| DERAS | Ex | 1648436792 | 0.4 | 0.34823 | 0 | 0.10071 | 1 | 4 | 0 | 4 | 1.6E+09 | 1.6E+09 | 0.0005556 | 0.4 | 0.113892 | 2400.00001 | 0 | 0 | 0 | 0.023354 | 0.001 | 2400 | Do nothing |
| DTM | Ex | 1648437392 | 0.45 | 0.34823 | 0 | 0.10071 | 1 | 4 | 0 | 4 | 1.6E+09 | 1.6E+09 | 0.0005556 | 0.45 | 0.122905 | 2400.00001 | 0 | 0 | 0 | 0.045749 | 0.001 | 2400 | Do nothing |
| DER | Ex | 1648437992 | 0.32 | 0.50353 | 0 | 0.11521 | 1 | 5 | 0 | 5 | 1.6E+09 | 1.6E+09 | 0.0005208 | 0.32 | 0.204221 | 2400.00001 | 0 | 0 | 0 | 0.098516 | 0.001 | 2400 | Do nothing |
| DCM | Ex | 1648438592 | 0.3 | 0.50353 | 0 | 0.11521 | 1 | 5 | 0 | 5 | 1.6E+09 | 1.6E+09 | 0.0005208 | 0.3 | 0.146641 | 2400.00001 | 0 | 0 | 0 | 0.073816 | 0.001 | 2400 | Do nothing |
| DERAS | Ex | 1648439192 | 0.17 | 0.50353 | 0 | 0.11521 | 1 | 5 | 0 | 5 | 1.6E+09 | 1.6E+09 | 0.0005208 | 0.17 | 0.116697 | 2400.00001 | 0 | 0 | 0 | 0.075099 | 0.001 | 2400 | Do nothing |
| DTM | Ex | 1648439792 | 0.39 | 0.50353 | 0 | 0.11521 | 1 | 5 | 0 | 5 | 1.6E+09 | 1.6E+09 | 0.0005208 | 0.39 | 0.135624 | 2400.00001 | 0 | 0 | 0 | 0.039659 | 0.001 | 2400 | Do nothing |
| DER | Ex | 1648440392 | 0.44 | 0.69125 | 0 | 0.12959 | 1 | 6 | 0 | 6 | 1.6E+09 | 1.6E+09 | 0.0005 | 0.44 | 0.214938 | 2400.00001 | 0 | 0 | 0 | 0.072767 | 0.001 | 2400 | Do nothing |
| DCM | Ex | 1648440992 | 0.31 | 0.69125 | 0 | 0.12959 | 1 | 6 | 0 | 6 | 1.6E+09 | 1.6E+09 | 0.0005 | 0.31 | 0.153744 | 2400.00001 | 0 | 0 | 0 | 0.068458 | 0.001 | 2400 | Do nothing |

Figure 6.1: An example of a TMS output showing the MVoT variables for each actor.

Referring to Figure 6.1, the first four columns of the output CSV file are the input to the simulator. As mentioned earlier, the input is a CSV file generated by the TMDG or based on a classified XML log. The column represents the actor name (column 1/A) , message evaluation category (column 2/B), the current time (column 3/C), and transit time (column 4/D). The remainder of the columns are values calculated by the simulator based on the input. The remainder of the columns are:

| Column | Description |
|---|---|
| 5/E | Trust score value |
| 6/F | Reputation value |
| 7/G | Distrust score value |
| 8/H | How certain is the DTM for each evaluation |
| 9/I | Certainty indicator of lean toward or against trust score or distrust score |
| 10/J | Total count of messages that are expected for each actor |
| 11/K | Total count of messages that are unexpected for each actor |
| 12/L | Number of total messages |
| 13/M | Message time stamp |
| 14/N | This can be the first time a message is received from an actor |
| 15/O | How often an actor communicates |
| 16/P | Message transit time |
| 17/Q | Average transit time |
| 18/R | Time since last communication |
| 19/S | Total count of timeouts for each actor |
| 20/T | Total count of alerts sent out to each actors |
| 21/U | Other action count |
| 22/V | SD transit time |
| 23/W | Max CommFreq |
| 24/X | Min TSLC |
| 25/Y | DTM response |

Table 6.2: An overview of the columns in a TMS output csv file.

The simulator output CSV file allows researchers to see how an actor's MVoT parameters change as more communication occurs. The CDTA dashboard also uses it to create plots and diagrams for the GO to view DTM System status.

### 6.3 DTM Classifier

Revisiting figure 4.2 in the software Description section, we can better understand the DTM workflow and the various components that create the DTM System. Referring to Figure 6.2, the operation starts at step 1. In this section, an XML log is provided to the DTM. The DTM

is tasked with classifying the message to arrive at a message evaluation category. Incoming messages are broken down into their individual XML tags and are parsed and compared against a schema to ensure that they are the correct format, type, and within the expected range for that target. Based on the various checks, a message evaluation is produced and is written to a CSV file along with the 'from' and 'to' actors, current time, and a pseudo-real transit time. Alternatively, the same outcome can be achieved by running the Trust Model Generator at step 2. However, unlike the workflow at step 1, step 2 mimicks the classifier outcome. No messages are being classified, but a message evaluation category is associated with the messages based on their user-specified arguments when running the generator.

A classified output is located at step 3. As mentioned and seen in Figure 6.2, a user can arrive at this file by either running the Trust Model Generator or classifying an incoming XML log. This CSV file contains the actor name, a message evaluation category, time stamp, and transit time.

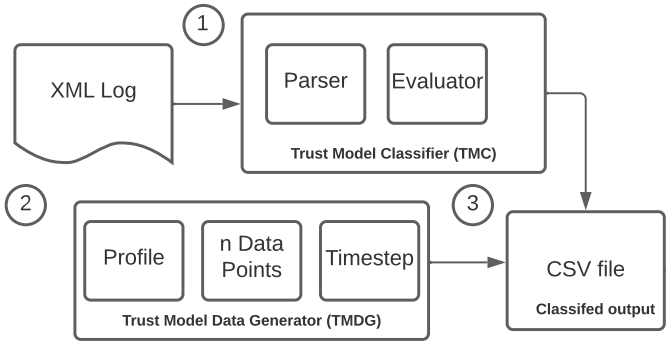


Figure 6.2: Trust Model System stages 1-3

Referring to Figure 6.3, we find that the Trust Model Simulator is located at step 4. As mentioned in great detail earlier in this chapter, the TMS reads in the CSV file and writes

out a CSV file with the tag "_Simout.csv", as shown in step 5.

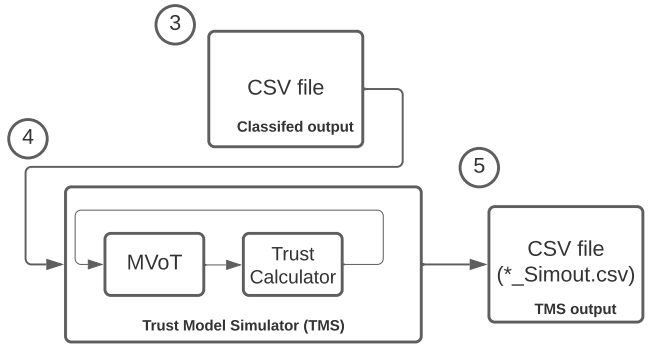


Figure 6.3: Trust Model System stages 3-5

As shown in Figure 6.4, to run the merger script in step 6, a dataset of 2 or more TMS output CSV files (*_Simout.csv) is required. The script reads the user-specified CSV files and creates a single CSV file with all the entries from the TMS output sorted in order of which they would have been received at the CDTA.

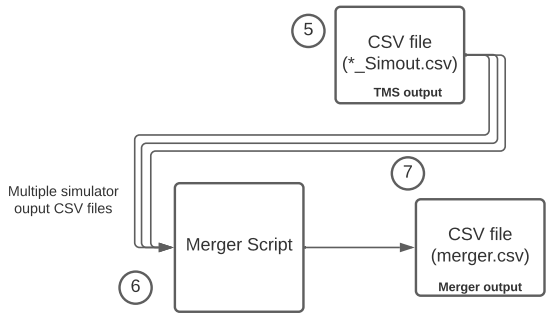


Figure 6.4: Trust Model System stages 5-7

## 6.4 Plots and Dashboards

As the messages get aggregated by the merger script, we arrive at a single CSV file that contains all the messages and MVoT for all the participating actors. The data in the CSV file is parsed to provide valuable information about the overall system state. Referring to

Figure 6.5, once the merger script is completed, it generates a single CSV file, as shown in step 7. This CSV file can then be used to generate plots and CDTA dashboards, as shown in steps 8 and 9, respectively.

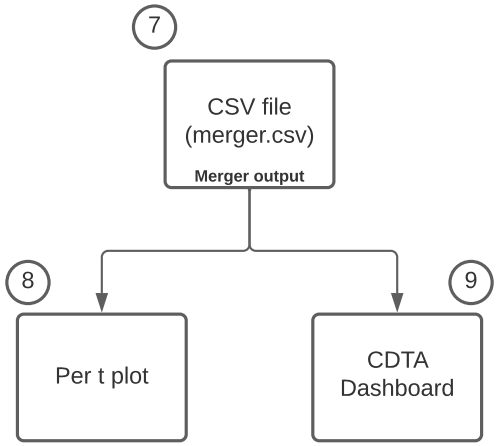


Figure 6.5: Trust Model System stages 7-9

By supporting plots and dashboards in the DTM System, an operator can monitor the overall system status. The DTM System Dashboard also provides critical information at a glance. The DTM Dashboard and plotting program support a wide range of MVoTs and time increments. The plotting program facilitates extracting the data associated with any of the MVoT and retrieves information on a per-minute basis. The dashboard is a dynamic and powerful tool that is valuable to the researchers working on the project and Grid Operators looking to understand how the system behaves. The remainder of this chapter showcases and discusses some of the plots and dashboards supported by the DTM System.

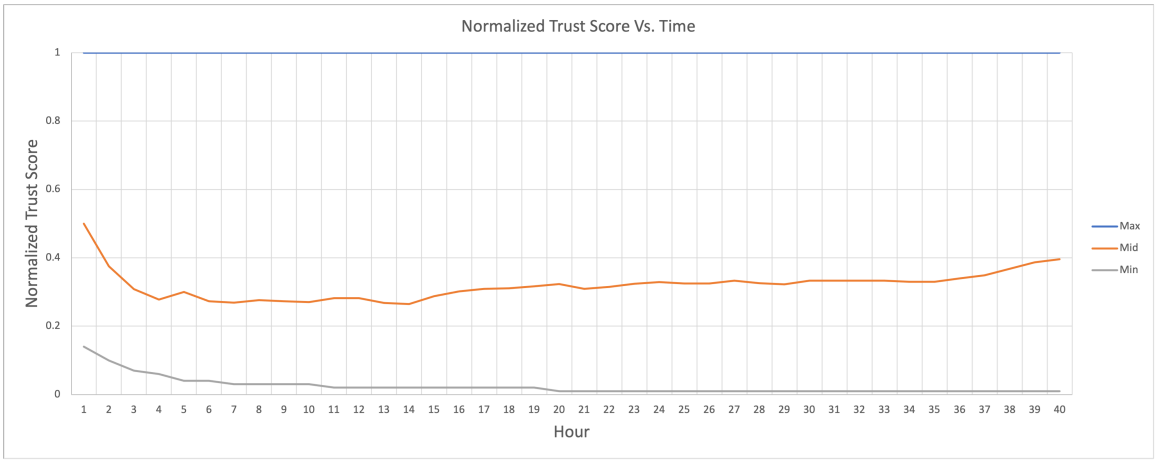


Figure 6.6: Normalized Trust Score Vs. Time for an Ideal Dataset

Figure 6.6 show the evolution of the Normalized Trust Score over time for an ideal dataset (i.e. all messages are evaluated to be expected). For this plot, there are 10 SPCs, each has a DER, DCM, and GSP, giving us a total of 30 actors resulting in 30 different MVoTs that the CDTA is aggregating. This plot describes the first 40 hours since 30 actors register with the DTM system.

As shown, the Normalized trust score decreases slightly during the first few hours. This is expected as we initialize the certainty value to 10 percent, and the DTM needs time before the various parameters adjust. As time goes on, the Normalized trust score starts to increase. The longer and more frequent an actor communicates expectedly, the higher their trust score becomes.

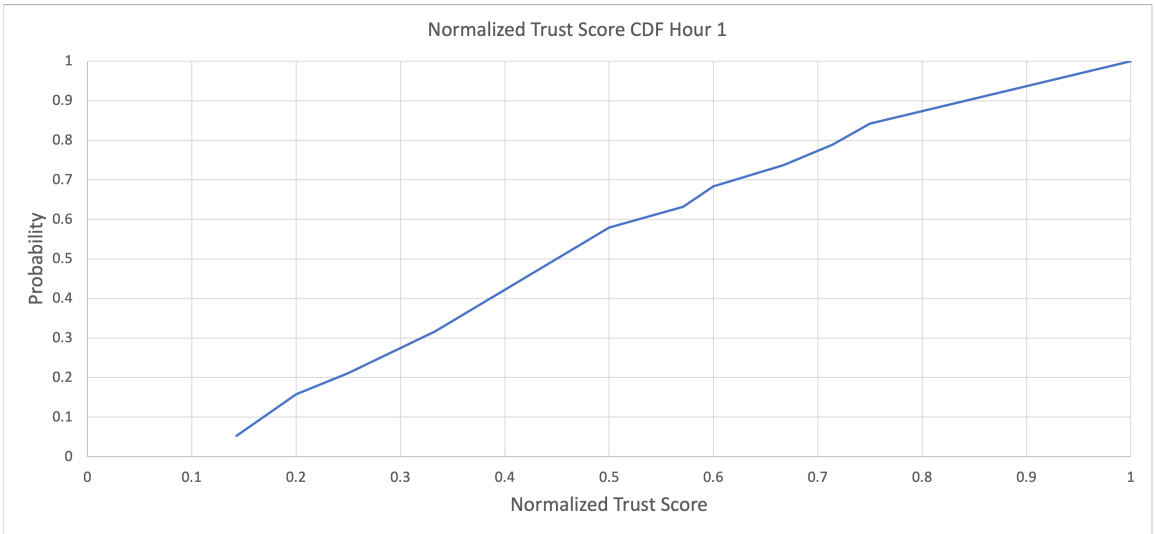


Figure 6.7: Normalized Trust Score vs. Cumulative Distribution Function: Hour 1

Figure 6.7 shows the Normalized Trust Score vs. Cumulative Distribution Function plot for the first hour. Since this is only the first hour, there is not much variation even though some actors communicate more frequently than others. This is because the actors have not communicated enough messages yet. As we discuss the normalized CDF of later hours, there will be more variance.

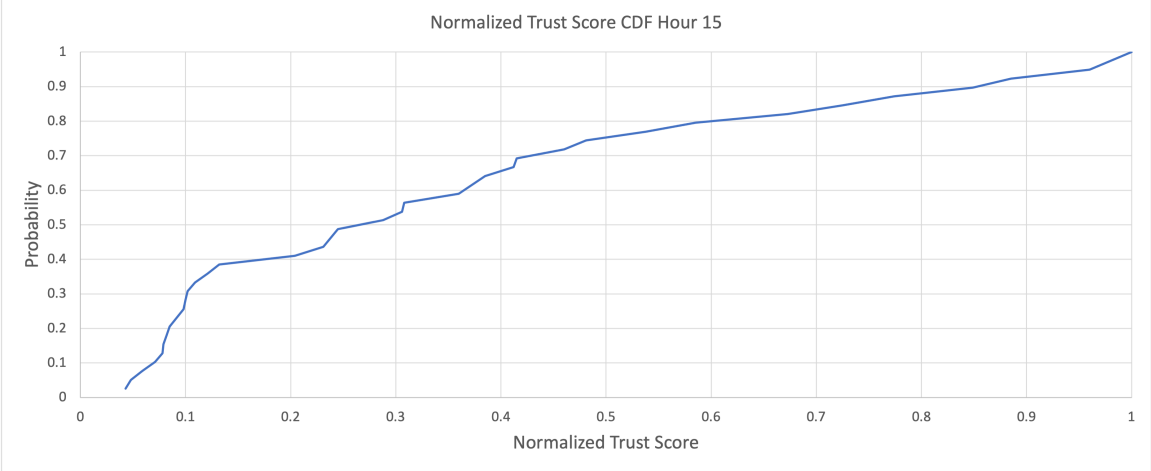


Figure 6.8: Normalized Trust Score vs. Cumulative Distribution Function: Hour 15

A plot of the normalized trust score vs CDF for the 15th hour is shown in Figure 6.8. There is more variation in the data. Specifically,there are four sharp increases in probability. This is attributed to the fact that our dataset is made up of actors communicating at four different communication frequencies. Some actors communicate every 5 minutes, some every 10 minutes, some every 15 minutes, and the majority of actors communicating every hour.

Referring to the Figure 6.8, most actors lag behind in terms of their normalized trust score. This is because the majority of the actors are communicating every hour, so they would have fewer expected messages, resulting in a lower normalized trust score.

The actors communicating every 15 minutes, which make up 20 percent of the data set, have a normalized trust score that ranges from 0.15 to 0.36. The actors communicating every 10 minutes, which make up another 20 percent of the data set, are spread out from around the 0.36 mark to around 0.60. A greater spread in Normalized Trust Score for these actors because certainty, incorporated in the trust score calculation, is non-linear. Finally, the most frequently communicating actors have a normalized trust score ranging from 0.60 to 1.

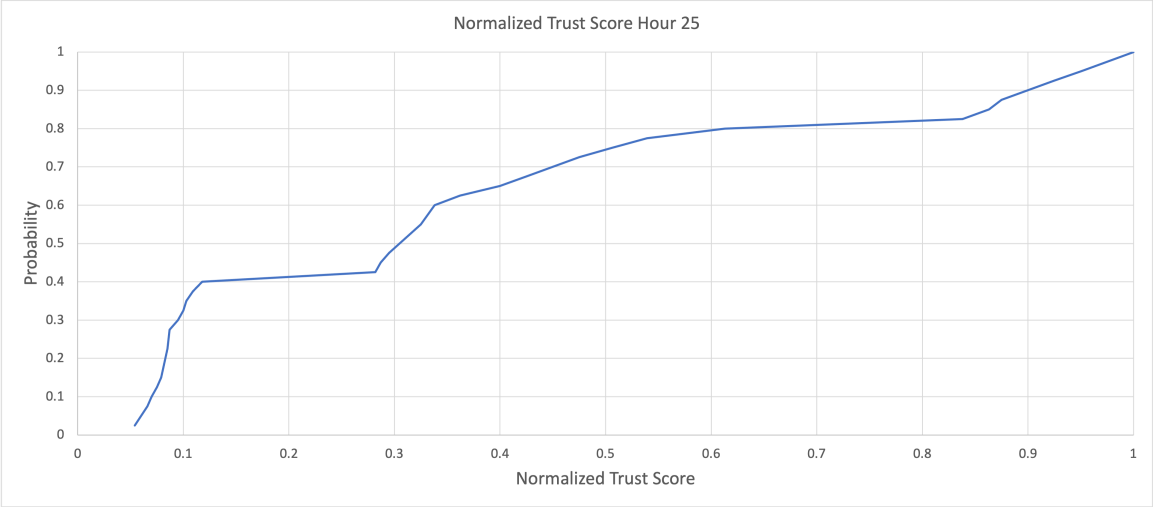


Figure 6.9: Normalized Trust Score vs. Cumulative Distribution Function: Hour 25

Referring to Figure 6.9, the majority of actors are still lagging in terms of their normalized trust score. This is once again because most of the actors communicate every hour.

Most of the actors communicating every 15 minutes have their normalized trust score range between 0.27 to 0.36 instead of the 0.15 to 0.36 range from the earlier plot in figure 6.8. The actors communicating every 10 minutes had their normalized trust score range spread out from the 0.34 mark to around 0.65, instead of the 0.36 to 0.60 range from the earlier plot in Figure 6.8. Finally, the most frequently communicating actors had their normalized trust score range from 0.65 to 1, instead of 0.60 to 1 from the previous plot in figure 6.8.

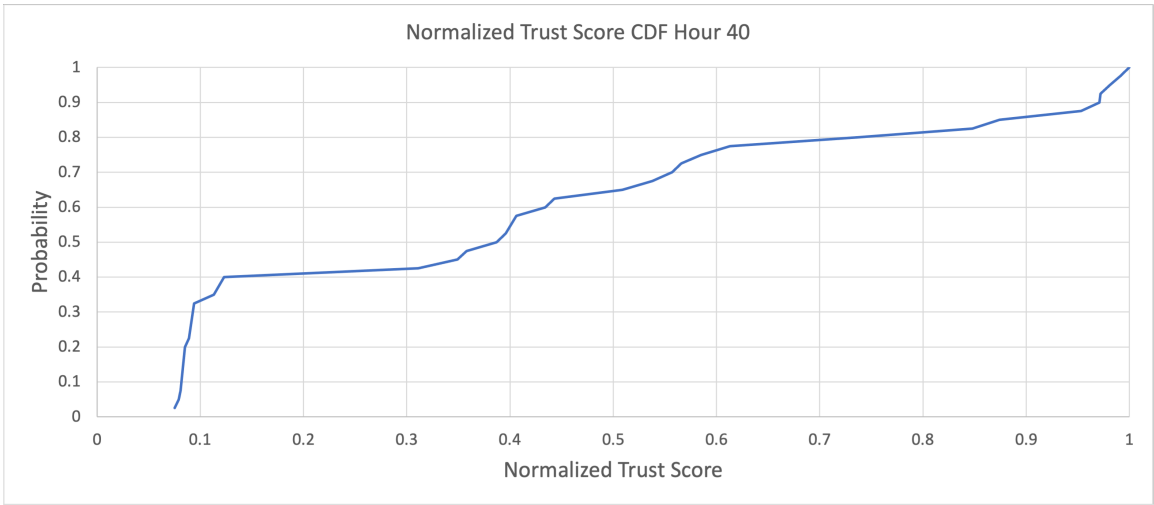


Figure 6.10: Normalized Trust Score vs. Cumulative Distribution Function: Hour 40

Referring to Figure 6.10, while the actors communicating every hour still have a normalized trust score less than 0.1, all the actors are increasing in trust score and slowly approaching 0.1

The actors communicating every 15 minutes have increased to where most of the actors have a normalized trust score that ranges from 0.35 to 0.43, instead of the 0.27 to 0.36 in the previous plot in figure 6.9. Likewise, the actors communicating every 5 minutes now range from 0.7 to 1 instead of the 0.65 to 1 range.

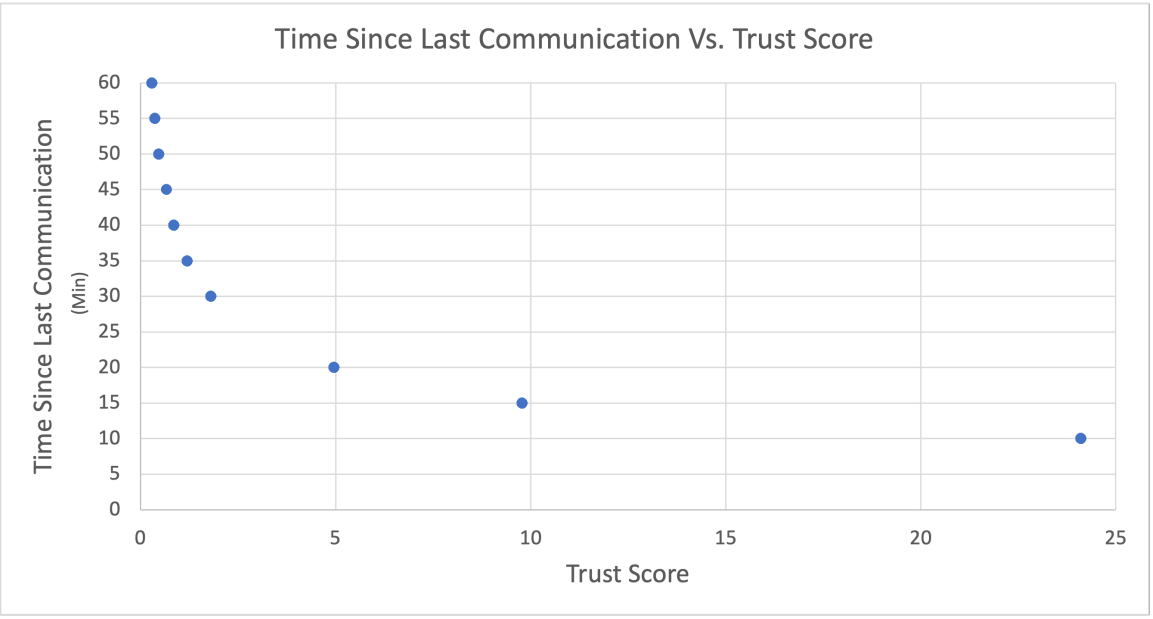


Figure 6.11: TSLC vs.Trust Score for 40 hours with varying communication frequencies

Figure 6.11 shows the relationship between TSLC and TS. The plot showcases the TS collected from 100 SPC actors. The DTMCs of each SPC aggregated the MVoT values to the CDTA. All 100 SPC actors communicated through a 40-hour time frame with varying communication frequencies. The range of communication frequencies is as follows:

- 10 SPC actors communicated every 10 minutes (600 seconds)
- 10 SPC actors communicating every 15 minutes (900 seconds)
- 10 SPC actors communicating every 20 minutes (1200 seconds)
- 15 SPC actors communicating every 30 minutes (1800 seconds)
- 15 SPC actors communicating every 35 minutes (2100 seconds)
- 15 SPC actors communicating every 40 minutes (2400 seconds)
- 10 SPC actors communicating every 45 minutes (2700 seconds)
- 5 SPC actors communicating every 50 minutes (3000 seconds)

- 5 SPC actors communicating every 55 minutes (3300 seconds)

- 5 SPC actors communicating every 60 minutes (3600 seconds)

Looking at the plots in Figures 6.7 to 6.11, as more expected messages are communicated, the actor's overall normalized trust score increases, and the variation in the plotted data becomes more apparent.

The DTM Simulation Suite provides powerful tools to modify, enhance, and test the DTM. The TMDG allows for datasets with varying characteristics to be readily generated. The TMS provides the MVoT calculations, parameters, initial values, and thresholds to be fine-tuned to give the DTM the optimum classification and detection level. Finally, the CDTA plotting tools provides a GO with an interface for EGOT DERMS.

# 7 Conclusion

The work described in this thesis aims to showcase the DTM System and the DTM Simulation Suite and their positive effect on EGoT DERMS. Furthermore, the DTM System enhanced and modified by the DTM Simulation Suite augments the existing security of DERMS as defined by the IEEE 2030.5 protocol.

The DTM Simulation Suite consists of the Trust Model Data Generator, Trust Mode Simulator, and CDTA plotting tools. The DTM Simulation Suite was developed to provide EGOT DERMS with a method for classifying messages between participating system actors, detecting system anomalies, and notifying the appropriate parties in the event of suspicious activity. The DTM Simulation Suite also honed in on the DTM's Trust Model to evaluate it and further improve it.

The Trust Model Data Generator was developed to generate datasets that depict real actor communications readily. The Trust Model Simulator incorporated the DTM Systems Trust Model to fine-tune it for correctness and accuracy. The Trust Model Simulator ran on hundreds of datasets generated by the Trust Model Generator to identify errors and shortcomings in MVoT equations. In addition, the CDTA plotting tools facilitated the creation of dashboard plots from aggregated data acquired throughout the DTM System, allowing Grid Operators to view the status of EGoT DERMS at a glance.

The DTM classifier can be further improved with machine learning to improve anomaly detection and data generation. Using a Trust Model Data Generator, an ML model can be trained to allow for more variation in the datasets than currently offered by the TMDG. Furthermore, providing real-time weather reports to the ML model can enable the DTM Classifier to consider weather-induced changes in communication patterns. The suggested improvement would improve detection rates and minimize false positives and negatives.

Security is an arms race, and attackers are a constant threat. The approach and solution discussed in this thesis allow for additional security in smart grids. The DTM System and its Simulation Suite complements grid security and adds extra layers of security and detection, making the energy sector of a nation's critical infrastructure more reliable, sustainable, and secure.

## Bibliography


[1] Félix Gómez Mármol and Gregorio Martínez Pérez. Providing trust in wireless sensor networks using a bio-inspired technique. *Telecommunication Systems*, 46:163–180, 2011.

[2] A. Boukerch, L. Xu, and K. EL-Khatib. Trust-based security for wireless ad hoc and sensor networks. *Computer Communications*, 30(11-12):2413–2427, 2007.

[3] Sonja Buchegger and Jean-Yves Le Boudec. A robust reputation system for P2P and mobile ad-hoc networks. 2004.

[4] Raja Naeem Akram and Ryan K.L. Ko. Digital trust - trusted computing and beyond: A position paper. In *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 884–892, 2014.

[5] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[6] Stephen Naicken, Anirban Basu, Barnaby Livingston, and Sethalat Rodhetbhai. A survey of peer-to-peer network simulators. In *Proceedings of The Seventh Annual Postgraduate Symposium, Liverpool, UK*, volume 2. Citeseer, 2006.

[7] Ben L Titzer, Daniel K Lee, and Jens Palsberg. Avrora: Scalable sensor network simulation with precise timing. In *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pages 477–482. IEEE, 2005.

[8] Pawani Porambage, Mika Ylianttila, Corinna Schmitt, Pardeep Kumar, Andrei Gurtov, and Athanasios V Vasilakos. The quest for privacy in the Internet of Things. *IEEE Cloud Computing*, 3(2):36–45, 2016.

[9] Ghada Glissa and Aref Meddeb. 6lowpan multi-layered security protocol based on IEEE 802.15. 4 security features. In *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 264–269. IEEE, 2017.

[10] Jorge Granjal, Edmundo Monteiro, and Jorge Sá Silva. Security for the internet of things: a survey of existing protocols and open research issues. *IEEE Communications Surveys & Tutorials*, 17(3):1294–1312, 2015.

[11] David Airehrour, Jairo Gutierrez, and Sayan Kumar Ray. Secure routing for internet of things: A survey. *Journal of Network and Computer Applications*, 66:198–213, 2016.

[12] Shoieb Arshad et al. Performance improvement of overlay networks for p2p systems using trust-based mechanisms. In *2012 Third International Conference on Intelligent Systems Modelling and Simulation*, pages 550–553. IEEE, 2012.

[13] Hamed Khiabani, Zailani Mohamed Sidek, and Jamalul-Lail Ab Manan. Towards a unified trust model in pervasive systems. In *2010 IEEE 24th International Conference*

*on Advanced Information Networking and Applications Workshops*, pages 831–835. IEEE, 2010.

[14] Hamed Khiabani, Norbik Bashah Idris, and Jamalul-Lail Ab Manan. Leveraging remote attestation to enhance the unified trust model for wsns. In *Proceedings Title: 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*, pages 139–143. IEEE, 2012.

[15] Xiangqian Chen, Kia Makki, Kang Yen, and Niki Pissinou. Sensor network security: a survey. *IEEE Communications surveys & tutorials*, 11(2):52–73, 2009.

[16] Alexander Becher, Zinaida Benenson, and Maximillian Dornseif. Tampering with motes: Real-world physical attacks on wireless sensor networks. In *International Conference on Security in Pervasive Computing*, pages 104–118. Springer, 2006.

[17] Arvind Seshadri, Adrian Perrig, Leendert Van Doorn, and Pradeep Khosla. Swatt: Software-based attestation for embedded devices. In *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, pages 272–282. IEEE, 2004.

[18] Mark Shaneck, Karthikeyan Mahadevan, Vishal Kher, and Yongdae Kim. Remote software-based attestation for wireless sensors. In *European Workshop on Security in Ad-hoc and Sensor Networks*, pages 27–41. Springer, 2005.

[19] Syed Wasif Abbas Hamdani, Abdul Waheed Khan, Naima Iltaf, and Waseem Iqbal. DTMSim-IoT: A distributed trust management simulator for iot networks. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Com-*

*puting, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 491–498. IEEE, 2020.

[20] T. Slay and R. Bass. An energy service interface for distributed energy resources. In *IEEE Conference on Technologies for Sustainability*, 2021.

[21] T. Slay, J. M. Acken, and R. B. Bass. Incentivizing distributed energy resource participation in grid services. In *9th IEEE Conference on Technologies for Sustainability (accepted)*, 2022.

[22] T. Slay, G. Spitzer, and R. B. Bass. Proposed application for an entity component system in an energy services interface. In *9th IEEE Conference on Technologies for Sustainability (accepted)*, 2022.

[23] Ieee standard for smart energy profile application protocol. *IEEE Std 2030.5-2018 (Revision of IEEE Std 2030.5-2013)*, pages 1–361, 2018.

[24] *CTA-2045-B: Modular Communications Interface for Energy Management*. Consumer Technology Association, November 2020.

[25] *SunSpec Information Model Specification: SunSpec Alliance Interoperability Specification Version 1.9*. Sun-Spec Alliance, April 2015.

[26] *Interconnection Requirements for Onboard, Grid Support Inverter Systems J3072*$_2$02103. $SAE, March$2021.

[27] Abhishek Narain Singh, MP Gupta, and Amitabh Ojha. Identifying critical infrastructure sectors and their dependencies: An Indian scenario. *International Journal of Critical Infrastructure Protection*, 7(2):71–85, 2014.

[28] Partha Datta Ray, Rajgopal Harnoor, and Mariana Hentea. Smart power grid security: A unified risk management approach. In *44th Annual 2010 IEEE international Carnahan conference on security technology*, pages 276–285. IEEE, 2010.

[29] Xiuzhen Chen, Shenghong Li, Jin Ma, and Jianhua Li. Quantitative threat assessment of denial of service attacks on service availability. In *2011 IEEE International Conference on Computer Science and Automation Engineering*, volume 1, pages 220–224. IEEE, 2011.

[30] Zhe Chen, Shize Guo, Kangfeng Zheng, and Yixian Yang. Modeling of man-in-the-middle attack in the wireless networks. In *2007 International Conference on Wireless Communications, Networking and Mobile Computing*, pages 2255–2258. IEEE, 2007.

[31] N. S. Fernando, J. M. Acken, and R. B. Bass. Developing a distributed trust model for distributed energy resources. In *IEEE Conference on Technologies for Sustainability*, 2021.

# Appendix A: Running the Software

To run the DTM System Simulation Suite, the system must be supported and all the dependencies must be installed.

## A.1 Dependencies

### A.1.1 System Dependencies

Both the trust model simulator and data generator have been tested on Linux and macOS. This how-to guide mimics the process on a Linux machine running Ubuntu 20.04.1 LTS. This is a Python-based program tested on Python 3.8.5. Other versions of Python3 should work but haven't been tested.

### A.1.2 Library Dependencies

To run the Simulation Suite, the Pandas library must be installed. To install Pandas, start by installing pip3

```
sudo apt install python3-pip
```

Once pip3 is installed, install Pandas

```
pip3 install pandas
```

## A.2 Running the Software

### A.2.1 Running the Trust Mode Data Generator

```
python3 TMsimDataGenerator.py -f [FILENAME] -p [PROFILE]
-n [NUMBER OF DATA POITNS] -i [TIMESTEP] -pa {optional}
[PROFILE ADDONS] -ma {optional} [MIXED ARGS]
-db {optional} [DER BIAS]
```

FILENAME - The name of the file you are generating. This must be a csv file (.csv)

PROFILE - The profile you want the generated data to be. Pick one of the profiles available below.

- ideal - This profile only generates expected data

- random - This randomly assigns a message evaluation to a random actor

- flawed - This profile generates no expected messages

- almost_good - This profile generates expected messages until the last set

- almost_bad - This profile generates Ux, Ind, Dis, N messages until the last set

N DATA POINTS - The number of messages to be generated per actor

TIMESTEP - The increment between each message

PROFILE ADDONS - Allows for special funcitonality in some profiles

- Random profile - if PROFILE ADDONS is 1, each data point will have a random timestamp

- almost_good - if PROFILE ADDONS is not zero, PROFILE ADDONS value will be where

a UX message is added

- almost_bad - if PROFILE ADDONS is not zero, PROFILE ADDONS value will be where a EX message is added

MIXED ARGS - Allows the user to specify the profiles to be used in the mixed profile

DER BIAS - Alows the user to specify how many more DERs should be generated

### A.2.2 Trust Mode Data Generator Example

```
python3 TMsimDataGenerator.py -f test.csv -p ideal -n 100 -i 1
```

This example generates a file called test.csv, the message evaluation for all the actors will be expected. There will be 100 messages per actor (400 messages total) and there is 1-sec step in the time

### A.2.3 Running the Trust Mode Simulator

```
python3 TMsim.py -f [FILENAME] -e [EQUATION] -d [DEBUG]
```

FILENAME - The name of the file you generated. This must be a csv file (.csv)

EQUATION - The equation version to be used. This is to allow for multiple equation versions. For now, there is only one version of equations so use "v1" when running.

DEBUG - This parameter allows for the simulator to run one message at a time. It will require the user to press a button to move to the next message. Options:

y - run simulator in debug mode

n - run simulator automatically

### A.2.4 Trust Mode Simulator Example

```
python3 TMsim.py -f test.csv -e v1 -d n
```

This example would provide test.csv as an input to the simulator. The simulator will run using the first version of the MVoT equations and will be running in automatic mode.

### A.2.5 Running CDTA Merger Script

```
python3 merger.py -f [FILE 1] [FILE 2] ... [FILE N]
```

All thats needed to run the CDTA merger script is to list all the csv files that you would like to merger in the current directory. The script will run on all the csv files entered in the command line same directory that it is in. If there are csv files with a different format, it might cause issues.

### A.2.6 Running CDTA Per Time Script

```
python3 perTPlots.py -f [FILE] -t [TIME] -i [INCREMENT]
```

FILENAME - The name of the file to read from (csv)

TIME - The sample size of the data in seconds. i.e. using 60 will give the last value per minute

INCREMENT - The number of hours you want within the data set

## Appendix B: DTM Classifer visual

The DTM System Classifier reads an XML log containing incoming messages and breaks the log into individual XML tags. Those XML tags are compared against a schema to ensure that they are in the correct format, type, and within the expected range for that target.Figure B.1 shows a visualization of the Trust Model Classifier input XML log (left side) and the classifier output CSV file (right side).


Figure B.1: XML to CSV conversion executed by the Trust Model Classifer

## Appendix C: DTM Simulator Code

```
# Abdullah Barghouti

# DTM Tool - Trust Model Simulator

# Portland State University Fall 2020


import sys

import csv

import argparse

import pathlib

import time

import math


import pandas as pd

from datetime import date


#take in user arguments

ap = argparse.ArgumentParser()

ap.add_argument('-f', '--file', required=True, metavar='',

help = 'name of the file that will be generated')

ap.add_argument('-e', '--equation', required=True, metavar='',

help = 'the equation to be used to calculate trust')

ap.add_argument('-d', '--debug', required=False, metavar='',
```

```
help = 'runs simulator in debug mode. Prints with user input')
args = ap.parse_args()

fileName = args.file
sfn = fileName.split(".csv")
simOutFileName = sfn[0] + "_SimOut.csv"

#Weights
ALPHA = 10
BETA = 2
DELTA = 1
GAMMA = 0.05

#Inital variables
n_SDTT = 0
n_AvgTX = 0
MAX_COMMFREQ = 0.0
ACTOR_MAX_COMMFREQ = {'DCM': 0.0, 'DTM': 0.0,
'DERAS': 0.0, 'DER': 0.0}
MIN_TSLC = 100
ACTOR_MIN_TSLC = {'DCM': 100000.0, 'DTM': 100000.0,
'DERAS': 100000.0, 'DER': 100000.0}

#load in threshold information into dictionaries
threshDict_over = {
    "distrust score" : 50,
```

```
    "timeout count": 2,
    "alert count" : 2,
    "time since last communication": 700
}

threshDict_under = {
    "trust score" : .1,
    "certainty" : .1,
    "relative factor of certainty": .5,
    "communication frequency": 1,
}

# response block thresholds
R_THRESH_TS = 4
R_THRESH_DS = 1.6
R_THRESH_C = 0.8
R_THRESH_TotMsg = 10
R_THRESH_CommFrq = 3.3
R_THRESH_CommFrq_HIGH = 6
R_THRESH_TxT = 40
R_THRESH_TSLC = 900
R_THRESH_RFC = 0.5

# response list
DTM_response = []
#increase to increase message freq
```

```
CFC = 0


#time stamp increment

TIMESTAMP = 1


TEMP = 0


#MVoT Parameters

param = {"trust score" : 0,

"reputation" : 0,

"distrust score" : 0,

"certainty" : 0.1,

"relative factor of certainty": 0,

"expected message count" : 0,

"unexpected message count" : 0,

"total message count" : 0,

"time stamp" : 0,

"registration date" : 0,

"communication frequency" : 0,

"message transit time" : 0,

"average transit time" : 0,

"time since last communication" : 0,

"timeout count" : 0,

"alert count" : 0,

"other action count" : 0,

"SD transit time" : 0,
```

```
"Max CommFreq" : 0.0,
"Min TSLC" : 100000.0,
#"DTM response:" : 0
}

TIME = time.time()

#Current MVoT Dict
currentTrustVector = {'DCM': param,
'DTM': param.copy(), 'DERAS': param.copy(),
'DER': param.copy()}

#Prev MVoT Dict
oldTrustVector = {'DCM': param.copy(),
'DTM': param.copy(), 'DERAS': param.copy(),
'DER': param.copy()}
#                0   1       2           3
actorList = {'DCM', 'DTM', 'DERAS' , 'DER'}

initalTime = {0,0,0,0}



#read from data (csv) file
'''
    this function is in charge of updating
    message counts and takes in a full row from the csv file
```

```
    generated by the trust model data generator
'''
def messageCount(current_row):
    actor = current_row[0]

    #update old vector
    oldTrustVector[actor]['unexpected message count'] =
    currentTrustVector[actor]['unexpected message count']

    oldTrustVector[actor]['expected message count'] =
    currentTrustVector[actor]['expected message count']

    oldTrustVector[actor]['total message count'] =
    currentTrustVector[actor]['total message count']

    print(" oldTrustVector[actor]['unexpected message count']",
    oldTrustVector[actor]['unexpected message count'])
    if (row[1] == 'Ex'):
        #update ex count
        currentTrustVector[actor]['expected message count'] += 1
    else:
        #update unex count
        currentTrustVector[actor]["unexpected message count"] += 1
        print("currentTrustVector[actor]['unexpected message count']"
        ,currentTrustVector[actor]['unexpected message count'])
```

```
    #update total message
    currentTrustVector[actor]["total message count"] =
    (currentTrustVector[actor]['expected message count'] +
    currentTrustVector[actor]["unexpected message count"])


'''
    this function is in charge of updating time
    related MVoT variables and takes in a full row
    from the csv file generated by the trust model
    data generator
'''
def timeKeeping(current_row):
    #get actor name
    actor = current_row[0]
    print(actor)


    #time stamp
    #set time stamp from 3rd col of generated data file
    print("old time stamp  = ", oldTrustVector[actor]["time stamp"])


    currentTrustVector[actor]["time stamp"] = float(current_row[2])


    #normal operations, update time stamp and TSLC
    if (currentTrustVector[actor]["total message count"] > 1):
        currentTrustVector[actor]["time since last communication"] =
        (currentTrustVector[actor]["time stamp"]
```

```
 - oldTrustVector[actor]["time stamp"])
    #normalize TSLC
    global MIN_TSLC
    if (currentTrustVector[actor]
    ["time since last communication"] <
    ACTOR_MIN_TSLC[actor]):
        ACTOR_MIN_TSLC[actor] = currentTrustVector[actor]
        ["time since last communication"]
        currentTrustVector[actor]["Min TSLC"] =
        ACTOR_MIN_TSLC[actor]

    #calculate normalized comm freq
    if (currentTrustVector[actor]["communication frequency"] >
    ACTOR_MAX_COMMFREQ[actor]):
        ACTOR_MAX_COMMFREQ[actor] = currentTrustVector[actor]
        ["communication frequency"]
        currentTrustVector[actor]["Max CommFreq"] =
        ACTOR_MAX_COMMFREQ[actor]

#first incounter. Set registration to be
#current time (time stamp) and TSLC to zero
if (currentTrustVector[actor]["registration date"] == 0):
    currentTrustVector[actor]["registration date"] =
    currentTrustVector[actor]["time stamp"]
    currentTrustVector[actor]
    ["time since last communication"] = 0
```

```
            oldTrustVector[actor]["time stamp"] = 0


        #update timestamp
        oldTrustVector[actor]["time stamp"] =
        float(currentTrustVector[actor]["time stamp"])


        #message transit time
        currentTrustVector[actor]["message transit time"] =
        float(current_row[3])


        print("Done timekeeping")
        print()

    '''
        this function is in charge of ... and takes in a
        full row from the csv file
        generated by the trust model data generator
    '''



def response(current_row):
    actor = current_row[0]
    response_flag = 0
    print("----RESPONSE SECTION----")
    #to see where we are
    print(currentTrustVector[actor]["total message count"])
```

```
#check for Excessive TSLC
if (currentTrustVector[actor]
["time since last communication"] >
threshDict_over["time since last communication"]):
    message = "Excessive time since
    last communication from " + actor
    DTM_response.append(message)
    response_flag = 1
    print(DTM_response)

#check for low trust score
if (currentTrustVector[actor]["trust score"] <
threshDict_under["trust score"] and
currentTrustVector[actor]["certainty"] >
threshDict_under["trust score"] and
currentTrustVector[actor]
["relative factor of certainty"] >
threshDict_under["relative factor of certainty"] and
currentTrustVector[actor]["distrust score"] >
threshDict_over["distrust score"]):
message = "Trust is low for " + actor
DTM_response.append(message)
response_flag = 1
print(DTM_response)
```

```
    if (not response_flag):
        message = "Do nothing"
        DTM_response.append(message)

    currentTrustVector[actor]["DTM response"] =
    ' '.join(DTM_response)
    print("DTM response", currentTrustVector[actor]
    ["DTM response"])
    DTM_response.clear()



'''
    this function prints the contents of the the MVoT to a
    userspecified CSV file
'''
def printToCSV(row,x):

    d = {'Actor' : row[0],
    'Message Eval Catagory' : row[1],
    'Current Time' : row[2],
    'Transit Time' : row[3]
    }
    actor = row[0]
    df2 = pd.DataFrame(row)
    print("df2", df2)
```

```
#df1 = pd.DataFrame(d,index=range(len(d)))
df1 = pd.DataFrame(d,index=range(len(d)-3))
#df2 = df1.join(df2)
print("df2 join", df2)
df = pd.DataFrame(currentTrustVector[actor],
index=range(int(len(currentTrustVector[actor].keys())/
len(currentTrustVector[actor].values()))))
print("df")
print(df)
df1 = df1.join(df)
#df2 = df2.join(df)
print("df1")
print(df1.columns)


if (x == 0):
    today = date.today()
    td = today.strftime("%d/%m/%Y")
    info = {'Name':args.file,
    'Data':td}
    #csv.writer('test1.csv', 'w', info)


    inf = pd.DataFrame(info, index = range(1))
    inf.to_csv(simOutFileName, index = False, mode = "a",
    header= False)
    df1.to_csv(simOutFileName, index = False, mode = "a")
    x += 1
```

```
    else:
        df1.to_csv(simOutFileName, index = False, mode = "a",
        header= False)


    temp = currentTrustVector[actor]
    k = temp.keys()
    #for k in temp:
    print(", ".join(k))
    print()
    val = []
    for k, v in temp.items():
        val.append(str(v))
    print(", ".join(val))



    data = currentTrustVector[actor]
    key = []
    val = []
    for k, v in data.items():
        key.append(k)
        pad = (len(k)- len(str(v)) ) * ' '
        val.append(pad + str(v))
    print(','.join(key))
    print(','.join(val))
#    for element in printActor:
```

```
def printClean():
    for K,V in currentTrustVector.items():
        print()
        #prints actor
        print(K)

        #prints dicts associated with actor
        for k,v in V.items():
            print(k,":",v)

def Time():
    global TIME
    TIME = TIME + TIMESTAMP
    return float(TIME)



'''
    this function is in charge of calculating the core MVoT
    values. It includes TS, C, DS, CommFreq, TSLC, TX time
    and avg_TX
'''

def calculation(current_row):
    actor = current_row[0]

    global n_AvgTX
```

```
n_AvgTX+= 1
global n_SDTT
n_SDTT+= 1
print("n_AvgTX", n_AvgTX)
print(actor)

#update old trust vector
oldTrustVector[actor]['certainty'] =
currentTrustVector[actor]['certainty']

oldTrustVector[actor]["trust score"] =
currentTrustVector[actor]["trust score"]

oldTrustVector[actor]["unexpected message count"] =
currentTrustVector[actor]["unexpected message count"]

oldTrustVector[actor]["average transit time"] =
currentTrustVector[actor]["average transit time"]

#calculate trust score
msgRatio = currentTrustVector[actor]
["expected message count"] -
(ALPHA * currentTrustVector[actor]
["unexpected message count"])
print("old certainty")
print(ALPHA, "ALPHA")
```

```
print(oldTrustVector[actor]["certainty"])
currentTrustVector[actor]["trust score"] =
msgRatio * currentTrustVector[actor]["certainty"]

#calculate communication frequency

print(currentTrustVector[actor]["registration date"],
"currentTrustVector[actor]['registration date']")
we = time.time()

if((float(current_row[2]) - currentTrustVector[actor]
["registration date"]) == 0):
    currentTrustVector[actor]["communication frequency"] =
    0.001
else:
    currentTrustVector[actor]["communication frequency"] =
    ((currentTrustVector[actor]["total message count"])/
    ((currentTrustVector[actor]["time stamp"] -
    currentTrustVector[actor]["registration date"])))
print("comm freq", currentTrustVector[actor]
["communication frequency"])


#calculate relative factor of certainty  (RFC)
print("expected message count", currentTrustVector[actor]
["expected message count"])
```

```
print("total message count",currentTrustVector[actor]
["total message count"])
currentTrustVector[actor]["relative factor of certainty"] =
abs(
    (currentTrustVector[actor]["expected message count"] /
    currentTrustVector[actor]["total message count"]) - .5 )
    * BETA

#calculate certainty
if (not currentTrustVector[actor]
["time since last communication"] <= 0):
    currentTrustVector[actor]["certainty"] = (
        currentTrustVector[actor]["relative factor of certainty"]
        * (1 - math.exp(-GAMMA *
        currentTrustVector[actor]["total message count"])) *
        (currentTrustVector[actor]["communication
        frequency"]/ACTOR_MAX_COMMFREQ[actor]) *
        (ACTOR_MIN_TSLC[actor]/
        (currentTrustVector[actor]
        ["time since last communication"])))
    if (currentTrustVector[actor]["certainty"] == 0):
        currentTrustVector[actor]["certainty"] = 0.1

    print(actor)
    print((1 - math.exp(-GAMMA *
    currentTrustVector[actor]["total message count"])),
```

```
    "Gamma")


#calculate distrust
currentTrustVector[actor]["distrust score"] =
oldTrustVector[actor]["unexpected message count"] *
currentTrustVector[actor]["certainty"]

print(currentTrustVector[actor]["distrust score"],
"currentTrustVector[actor]['distrust score']")

#average transit time
currentTrustVector[actor]["average transit time"] =
((currentTrustVector[actor]["message transit time"] +
n_AvgTX * oldTrustVector[actor]["average transit time"])/
(n_AvgTX + 1))

#standard deviation transit time
SD = ((n_SDTT * oldTrustVector[actor]
["SD transit time"] ** 2 +
(currentTrustVector[actor]["message transit time"] -
oldTrustVector[actor]["average transit time"]) *

currentTrustVector[actor]["message transit time"] -
currentTrustVector[actor]["average transit time"]) /
n_SDTT)
```

```
        print(currentTrustVector[actor]["total message count"], "MSG CNT")
        print(oldTrustVector[actor]["SD transit time"] ** 2, "SDTT ^2")
        print(currentTrustVector[actor]["message transit time"], "MSG TX")
        print(currentTrustVector[actor]["average transit time"], "AVG TX")
        print(SD, " SD")
        SD = abs(SD)
        currentTrustVector[actor]["SD transit time"] = math.sqrt(SD)
        #calculate relative factor of transit time
        '''
        if (abs(currentTrustVector[actor]
        ["average tranist time"] - currentTrustVector[actor]
        ["message tranist time"]) >
        abs(DELTA * currentTrustVector[actor]
        ["SD transit time"])):
        print("BAD")
        '''



if (pathlib.Path(args.file).exists() == True):
    with open(args.file, mode='r') as csvFile:
        fileReader = csv.reader(csvFile)
        count = 0
        x = 0
        for row in fileReader:
            if (count > 0):
                #update message counters
```

```
                messageCount(row)
                #timekeeping new message
                #print(oldTrustVector)
                timeKeeping(row)

                if (args.equation == "v1"):
                    #calculate MVoT
                    calculation(row)
                    #generate response
                    response(row)
                    #write to output csv file
                    printToCSV(row,x)
                    x += 1
                #response(row)

            if (args.debug == 'y'):
                input("Press Enter to continue...")
            count += 1
        print("count", count)
        printClean()
        print()
        print(currentTrustVector)

else:
    print("File does not exist")
```