Trust Modelling and Management for Collaborative and Composite Applications in the Internet of Things

Anuoluwapo Amarachukwu Adewuyi

A Thesis Submitted in Partial Fulfilment of the Requirements of Liverpool John Moores University for the Degree of Doctor of Philosophy

October 2021

SUPERVISORS' CERTIFICATION

We certify that the thesis entitled "**Trust Modelling and Management for Collaborative and Composite Applications in the Internet of Things**" was prepared under our supervision at the Department of Computer Science and Mathematics, Liverpool John Moores University, in partial fulfilment of the requirements of Liverpool John Moores University for the degree of Doctor of Philosophy.

Name	**Prof. Qi Shi**
Title	Director of Study
Address	James Parsons Building, Byrom St, Liverpool, L3 3AF, UK

Name	**Dr. Hui Cheng**
Title	Second Supervisor
Address	Department of Computer Science, University of Hertfordshire, Hatfield, Hertfordshire, UK

Signed (Director of Studies): [illegible]

Date: 11/10/2021

ABSTRACT

A future Internet of Things (IoT) will feature a service-oriented architecture consisting of lightweight computing platforms offering individual, loosely coupled microservices. Often, an end-user will request a bespoke service that will require a composition of two or more microservices offered by different service providers. This architecture offers several advantages that are key to the realisation of the IoT vision, such as modularity, increased reliability and technology heterogeneity and interoperability. As a result, the adoption of this architecture in the IoT is being extensively researched. However, the underlying complexities of service compositions and the increased security risks inherent in such a massively decentralised and distributed architecture remain key problems. The use of trust management to secure the IoT remains a current and interesting topic; its potential as a basis for service compositions has not been thoroughly researched, however.

Security through trust presents a viable solution for threat management in the IoT. Currently, a well-defined trust management framework for collaborative and composite applications on an IoT platform does not exist. In this thesis, a collaborative application refers to the one that enables collaboration among its users to jointly complete certain tasks, whereas a composite application is the one composed of multiple existing services to deliver integrated functionalities. To estimate reliably the trust values of nodes within a system, the trust should be measured by suitable parameters that are based on the nodes' functional properties in the application context. Existing models do not clearly outline the parametrisation of trust. Also, trust decay is inadequately modelled in many current models. In addition, trust recommendations are usually inaccurately weighted with respect to previous trust, thereby increasing the effect of bad recommendations.

This thesis focuses on providing solutions to the twin issues of trust-based security and trust-based compositions for the IoT. First, a new model, CTRUST, is proposed to resolve the above stated shortcomings of previous trust models. In CTRUST, trust is accurately parametrised while recommendations are evaluated through belief functions. The effects of trust decay and maturity on the trust evaluation process were studied. Each trust component is neatly modelled by appropriate mathematical functions. CTRUST was implemented in a collaborative download application and its performance was evaluated based on the utility derived and its trust accuracy, convergence, and resiliency. The results indicate that IoT collaborative applications based on CTRUST gain a significant improvement in performance, in terms of efficiency and security.

In a second study, the trust properties of service compositions in the IoT, along with the effect of the service architecture on the security and performance of the composed service, are investigated. Novel approaches are considered in relation to trust decomposition and composition, respectively. Relevant trust evaluation functions are derived to guide the compositions, which are used to extend CTRUST into a new trust model, SC-TRUST. SC-TRUST is implemented in a suitable simulation and the results are evaluated. The model reliably guides service compositions while ensuring utility to the end-user. Overall, the analyses and evaluations support the conclusion that the trust models are effective in terms of performance gain and security. The models are scalable and lightweight such that they could be deployed to secure applications and drive meaningful services and collaborations in the envisaged IoT and Web 3.0 sphere.

ACKNOWLEDGMENTS

This thesis is dedicated to my God and Saviour Jesus Christ, Who gives me life and strength - and without Whom no endeavour is meaningful or possible – for helping me to begin and finish this project.

My Director of Study, Prof. Qi Shi, has my immense gratitude and appreciation, now and always, for his support, patience, insights, and the overall kindness and understanding he showed, all which helped bring this work to fruition. My second supervisor, Dr. Hui Cheng, was exceptionally invaluable to my progress and development throughout the programme.

I am grateful for the support I received from the Department of Computer Science, both financially and administratively. I must specially appreciate Tricia and her team for supporting my research and responding to my queries (which would be well over a thousand) over the years.

My indescribably wonderful parents, Pastor Dr. and Pastor Mrs. Adewumi and Maryjones Adewuyi, who encouraged me to undertake this programme, sponsored and coached me, and gave and did everything I required and beyond, deserve my eternal gratitude. My darling wife, loyal partner, faithful best friend, and inspirational muse, Osekafore, you are the best! I love you! My lovely siblings, Onaopemipo and Boluwarin, have my eternal love and gratitude for their unconditional and unwavering support, without which I would have given up long ago. Their constant affirmation pushed me through to the finish line.

Finally, I appreciate Christ the Cornerstone Bible Church for providing me with spiritual and emotional guidance, and loving friends.

PUBLICATIONS

Journal Papers:

1. A. A. Adewuyi, H. Cheng, Q. Shi, J. Cao, A. MacDermott, and X. Wang, "CTRUST: A dynamic trust model for collaborative applications in the internet of things," IEEE Internet of Things Journal, vol. 6, no. 3, pp. 5432–5445, Feb. 2019, doi: 10.1109/JIOT.2019.2902022.

2. A. A. Adewuyi, H. Cheng, Q. Shi, J. Cao, X. Wang, and B. Zhou, "SC-TRUST: A Dynamic Model for Trustworthy Service Composition in the Internet of Things," IEEE Internet of Things Journal, 2021, doi: 10.1109/JIOT.2021.3097980.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AHP	Analytical Hierarchical Process
BPMN	Business Process Model and Notation
CBA	Cumulative Bandwidth Average
CD	Collaborative Downloading
CPS	Cyber-Physical Systems
ER	Evidential Reasoning
IoT	Internet of Things
IRI	Inverse Risk Index
JSON	JavaScript Object Notation
MCDA	Multiple Criteria Decision Analysis
P2P	Peer-to-Peer
QoS	Quality of Service
SCR	Successful Completion Rate
SDN	Software-Defined Networking
SIoT	Social Internet of Things
SOA	Service Oriented Architecture
SP	Service Provider
SR	Service Requester
TaaD	Trust-as-a-Decision
TMS	Trust Management System
WSNs	Wireless Sensor Network

LIST OF FIGURES

List of Tables

CHAPTER 1 INTRODUCTION

1.1 Research Motivation

The increasing development and pervasiveness of the IoT have facilitated the proliferation of a new generation of smart objects [1], [2]. A smart object is a physical thing or device with a unique identifier and equipped with some computing and networking capabilities that enable it to connect and communicate with similar objects and human users [3]–[5]. A major consequence of this rapid adoption is that human users can connect to their environment, data, and services at an unprecedented scale for various novel, multi-contextual, dynamic, and on-demand applications. The potential benefits and use cases of the IoT apply to virtually every domain of social life, including healthcare, transportation and agriculture [6]–[8]. The IoT paradigm is built upon other research areas such as cyber-physical systems (CPS), wireless sensor networks (WSNs), big data, machine learning, adhoc networks, mobile computing, and ubiquitous computing, and requires collaborations and interoperability between various devices and networks on a massive scale.

Generally, the future IoT will be used to provision collaborative applications and service compositions. In collaborative IoT applications, several users or devices come together and pool their resources to execute a task or provide a service. The resources could be bandwidth, network routes and access, processing power or storage space. The motive behind the collaboration could be an increase in the speed of the task execution, as is the case in the collaborative download of a file. It could also serve to achieve redundancy and therefore increase reliability, as is the case in collaborative storage, routing, and streaming applications. These are a few examples of use cases where a collaboration is beneficial. Most of these collaborations will be formed “on the go”; that is, the collaborating peers will be unknown to each other.

Another principal and necessary component of the future IoT will be the provision of bespoke services on the fly to satisfy dynamic user requirements. This will require the cooperation and collaboration of individual smart devices offering unique microservices which can be transparently composed, as required, to provide a service offering that is guaranteed to fulfil the user’s service requests [9]–[11]. The process by which this is done is called a service composition. In this context, *transparency* means that the inner working of the composition is abstracted from the users or entities interacting with the composed service. Specifically, performance, access and location transparencies [12] are implied. The composed service appears as a single service to the user requesting the service, as the composition should be performed in an agnostic manner; the existence and details of the underlying microservices and any middleware should be abstracted from the user [13], [14].

In a service composition, the IoT middleware consists of a service-oriented architecture (SOA) where each connected device is a service provider (SP) or a service requester (SR) [7], [15]. The underlying microservices are hosted and provided by SPs, while the middleware layer accepts service requests from end-users, performs service discovery and aggregation, SP selection, network routing functions, and security and trust management, and delivers the results of the composed service to the requesting user [16]–[19]. The middleware itself may be hosted in the cloud or by another device which provides the composition platform. In this scenario, the devices providing the underlying services can be referred to as wholesalers of such services, while the middleware conglomerates such services and retails them to the end-user. A device may provide multiple services, e.g. temperature and humidity sensing. However, each service usually belongs to a single service class, which is an abstraction for all services of the same type [10]. Depending on the type of composition, the output of one service may be passed as input to another. For example, the geographical coordinates from a GPS sensor (geolocation service class) may be passed to a weather forecast service and the forecast readings are returned to the user. This network of interactions between users, smart objects, and the services they offer forms the foundation of the concept referred as the Social Internet of Things (SIoT) [17], [20]–[23].

The IoT requires the extensive interoperability of heterogeneous devices, networks, and technologies, for which the traditional internet is ill-suited. Therefore, security is of critical importance but is more complex to manage because this heterogeneity expands the threat landscape [24]. Also, due to the different IoT technologies being used, and the fact that most IoT devices have limited computing power, employ a distributed architecture and use less conventional networking methods, traditional security management used in the current Internet cannot be directly applied here [24], [25]. New security countermeasures are required that are lightweight, intelligent and operable in real-time [24]–[26]. While this remains an ongoing research challenge, an interesting candidate solution is security through trust. Trust is an important component of computer security [27]. Given that the IoT consists of services and devices provided by different actors who may be unknown to end users, the necessity of trust evaluation is higher than it is for the traditional Internet [24], [28].

The need for secure and trustworthy collaborations and service compositions is evident. Given that collaborating parties are largely anonymous to one another (at least, initially and in most use cases), the security risks involved are high and include privacy and data loss, which may be used as vectors for more sophisticated attacks. The introduction of the notion of trust among peers is one way to minimise these threats [24], [26], [29]. For service compositions, there may be multiple SPs offering the same functionality within a service class. In composing a user-requested service, it would be necessary to select the most reliable and least malicious SPs in each required service class. This is particularly important given the transparent nature of the composition, as the end user may have no knowledge or relationship of the providers of the underlying services. Therefore, some method must exist to identify malicious nodes and preclude their service offering from the composition [30]. Also, the composed service itself must be guaranteed to fulfil the request of the user, while maintaining security and privacy. In other words, the middleware must consider the compatibility of the service components to be mixed and matched, while ensuring that the results are reliable and reasonably satisfy the requester's utility [16], [31], [32].

The difficulty in finding universal solutions suitable for the IoT context establishes trust as, perhaps, the most important security metric in SOA-based IoT systems. In fact, it has been posited that trust management is wider in scope than traditional security management [26], because while the latter is largely corrective, the former is more predictive and preventive, and also addresses the quality of service (QoS) provided and reliability of SPs [17], [26], [32]. Thus, it can be used to determine which nodes should be selected for interactions and service provision, while excluding malicious nodes from the service context. However, though trust is an essential and common social concept, it is difficult to define due to its abstract and multi-faceted nature. It is also either purely or mostly subjective, and its meaning depends on the context in which it is used [33], [34]. Even though there is no agreed definition in literature, a large volume of research on trust shows it is a very important concept [24]. For example, a 'trusting intention' is given in [33] as "the extent to which one party is willing to depend on the other party in a given situation with a feeling of relative security, even though negative consequences are possible". This definition correctly identifies trust as a decision taken by the trustor. Another issue is that several definitions of trust do not provide measurable indices that may be used to evaluate trust.

The concept of computational trust is, thus, introduced. Computational trust is the adaptation of the social notion of trust to the digital world, so that it can be represented and evaluated by mathematical models [28], known as trust models. These determine how trust is computed in a specific context. A trust management system (TMS) provides methods and mechanisms to evaluate the trustworthiness of interacting peers, based on a trust model. This work focuses on the design of trust models for two major IoT contexts: collaborative applications and social compositions.

1.2 Research Problems and Justification

Several TMSs have been proposed and widely studied in literature. However, there are comparatively fewer studies on the management of trust in IoT contexts [15], [26], [30], [32]. Many of these models focus mainly (or only) on recommendations based on an assumed social relationship among interacting nodes. Little or no account is taken of other aspects of trust management, such as trust decay, trust parameter selection and the weighting of trust parameters. However, given that collaborative applications are task-based systems, the trust score of a trustee node should indicate the degree to which a trustor believes that the trustee is both competent and willing to execute required task(s) reliably.

Even less work has been done on the management of trust in SOA-IoT contexts [15], [32]. Furthermore, these models do not consider transparent service compositions, where an SR can neither provide direct trust ratings on nor receive recommendations on SPs of the underlying services. In addition, they do not consider the trustworthiness of the composed service separately from the trust ratings of the SPs. Also, there is an implicit assumption that, due to the social nature of SOA-based IoT, a social relationship exists between the owners of the participating IoT devices. Based on this assumption, such social relationships must factor into the trust estimation. However, in a true service-based IoT where the primary or only incentive for interactions is to provide or request a service, no other relationship may exist among SPs and SRs outside the given service context. The implication of this is that these models may exclude trustworthy nodes capable of delivering reliable services and include nodes which are owned by SPs who have an external social relationship with the SR. Finally, these models do not respect privacy, as the middleware composing the service must be made aware of these relationships among the SPs and SR.

In summary, there is a need for reliable and well-defined trust models to for both collaborative applications and compositions of IoT services. Such models must express trust as a performance metric that is based on the functional properties of the utilised IoT context. This work addresses these research gaps by providing a comprehensive approach to trust evaluation management in IoT applications. The aims and objectives of this research are enumerated in the next section.

1.3 Research Aims and Objectives

1.3.1 Aims

This research work aims to study existing TMSs, investigate their limitations with respect to IoT contexts, and to design and implement ideal trust models suitable for collaborative applications and service compositions in the IoT. By modelling trust as a performance metric measured through assessable criteria that are derived from functional properties of nodes in the context, the trust models developed will ensure security and a reliable quality of service in IoT applications. The realisation of this aim will increase the adoption of IoT offerings by enabling the provisioning of dynamic applications built from resilient microservices with low-risk thresholds.

This aim proposes to answer some research questions namely, how to:

- **Define** the notion of Trust among nodes in a collaborative or composite IoT application.
- **Detect** and measure this trust computationally while modelling it like social trust among humans.
- **Deduce** the effects of recommendations and decay (temporal degradation) on trust values.
- **Decide** how to aggregate and apply computed trust values to select "partner/friendly" devices in various contexts in the IoT.

1.3.2 Objectives

To accomplish the aim of this project, the following objectives must be achieved:

1. A comprehensive review of the existing literature on TMSs, considering trust derivation, evaluation and aggregation employed in these trust models, and their suitability and limitations for IoT contexts.

2. Derivation of the necessary properties for an ideal functional trust model for IoT contexts, specifically for collaborative applications and service composition.

3. A formal definition of the concept of functional trust, and proposal of methods for accurate trust estimation, aggregation, evaluation, decay, and its application in the IoT.

4. Design and evaluation of a dynamic trust model for collaborative applications in the IoT.

5. Design and evaluation a dynamic model for trustworthy service composition in the IoT, based on the model in (4) above.

The methodology deployed to achieve these goals is discussed in the next section.

1.4 Research Methodology and Scope

1.4.1 Methodology

This research was divided into four major phases, namely:

1. **Literature review**: a thorough review of the existing body of work regarding trust modelling and management, collaborative tasks, and service composition in the IoT was conducted to identify research gaps and generate insights into practical solutions. Thus, in this phase, Objective 1 is fulfilled.

2. **Requirements Analysis and specification**: following on from (1), the current state of the art was critically analysed to elicit the necessary properties of ideal trust models for collaborative applications and service compositions in the IoT. To ensure that these models are viable, the architecture and limitations of IoT networks and devices were considered. In this phase, Objective 2 was completed, and Objective 3 was partly achieved.

3. **Model Design**: Based on the results of the analysis in (2), initial parameters for the proposed models were determined. The design phase was divided into two sub-phases: (i) design of a trust model for collaborative applications and (ii) extension of the previous design to a new model for trustworthy service composition in the IoT. This phase completely fulfils Objective 3 and satisfies the design goal of Objectives 4-5.

4. **Implementation and evaluation**: In this phase, the designed models were implemented by simulation of appropriate contexts, based on the methods developed in the design phase. Rigorous testing was carried out to ensure that the models meet the specified criteria derived during the requirements analysis phase. Furthermore, the performance of the implemented models was evaluated with respect to relevant existing work, to show its distinction and significance in the field of study. This, together with Phase (3), completely satisfies Objectives 4-5.

1.4.2 Scope

There are several issues involved in the design of any IoT solution due to the heterogenous nature of the IoT, in terms of devices, technologies and networks. The range of IoT devices includes sensors, mobile devices, RFID tags, smart bands, and vehicles, and these may utilise a vast array of networks such as Wi-Fi, 5G, or IEEE 802.15.4 networks (such as ZigBee). This research focuses on broadly on trust derivation and evaluation for the IoT in general, especially the trust-related aspects of collaboration of service composition, and largely abstracts technical details regarding the type or topology of devices and networks. However, the following assumptions are made concerning IoT device capabilities in this project:

1. The devices have some memory and processing abilities to perform lightweight processes.
2. The devices have some form of persistent storage, even if small.
3. The devices have some form of unique identifiers.
4. Given that the IoT contexts studied in this research are largely peer-to-peer (P2P) networks, it is assumed that the designed trust models will be implemented in a decentralised network architecture.

Furthermore, it may not always be possible to directly compare the results obtained directly with other models in literature, due to different IoT implementation environments. It will not be feasible to adapt each one of these other algorithms to be implemented in our chosen contexts. In these situations, an indirect evaluation approach is adopted instead. This involves comparing the performance of the models to two baseline cases:

(i) to ground truth (that is, given perfect knowledge of the behaviour of the collaborating devices) and (ii) to results obtained from a random selection of peers in the same context. The results can then be indirectly compared to other trust models in literature that have been evaluated against either or both baseline cases. Statistical tests will be performed to determine whether any differences found are significant.

1.5 Novel Contributions

This research proposes the following additions to the existing body of work:

1. The use of weighted trust parameters (criteria) that can be specified at runtime to adapt the model to different contexts. This means that trust parameters, in contrast to recommendations, form the basic building block for trust computation. In most trust models in literature, the trust computation is built almost entirely on recommendations. This approach does not consider trust as a performance metric, and thus weakens the trustor's decision to trust. To the proposer's knowledge, this is the first work to formally define trust as a decision-making process and utilise decision analysis techniques for trust elicitation, aggregation, and evaluation, thereby establishing its novelty.

2. The study of the effects of trust degradation over time as a distinct component of the trust computation. Previous trust models use parameters to weight past trust to current experience. This is done recursively with every new interaction and therefore does not consider the time that has elapsed since a previous trust assessment was made. To resolve this, the proposed models include a novel trust decay function

with a dynamic component to accommodate different degrees of nodes' willingness to trust. This ensures trust degrades in a consistent manner.

3. The implementation of an improved recommendation function with the addition of a novel belief degree function, utilising established evidential reasoning (ER) techniques. Other trust models only consider the recommender's trust scores in accepting recommendation. The designs developed in this work identified other criteria that determine the degree to which a recommendation is accepted in social contexts and applied them to model the belief degree, which is then used to weight recommendations received from other nodes.

4. The design of a parameter to model trust maturity or equilibrium between two nodes, the point at which trust can be computed using direct interactions alone. This implies that it is possible to determine trusted nodes solely by empirical methods, which is a novel contribution.

5. Formulation of methods for the derivation and aggregation of trust values based on the relevant trust parameters of SPs from different required service classes, based on a study of requirements identified for a suitable trust model for service compositions in the IoT context. As this is the first formal study to comprehensively investigate trust-based service composition in the IoT, it is a novel contribution.

6. The design of a method to reliably estimate the trust score of the composed service based on the indirect trust scores computed for the service providers in a service composition in accordance with the service requester's requirements, thus providing a novel privacy-preserving solution for transparent trust composition in the IoT.

7. Finally, the service composition model receives trust ratings from the SR based on the satisfaction received from the composed service and provides a method to indirectly update the trust scores of the underlying services. This solves the problem of transparent trust decomposition and is therefore a novel contribution.

These contributions are set out in detail in subsequent chapters of the thesis, which are set out below.

1.6 Organisation of The Thesis

Chapter 2 introduces related concepts and provides a comprehensive overview of trust modelling and service composition techniques, highlights research gaps, and specifies requirements for the proposed IoT trust models.

Chapters 3 and 4 specify, analyse, implement, and evaluate the designs of the two proposed trust models, for collaborative applications and service compositions in the IoT, respectively.

Chapter 5 summarises and concludes the thesis. In addition, it points out some directions for further research extending from this work.

CHAPTER 2 BACKGROUND STUDY

This chapter reviews previous trust-related work in the IoT, with focus on two broad categories: (i) trust modelling and management, discussed in Section 2.1 and (ii) trust-based service composition and provisioning, discussed in Section 2.2. In section 2.3, the desirable properties of an ideal IoT-centric trust model are enumerated.

2.1 Trust Modelling and Management

The concept of trust modelling and management in the IoT is a rapidly evolving research area. A system model and a holistic trust management framework are given in [26]. The overall objective is to ensure that trust models offer a balanced approach to the realisation of functional and non-functional requirements of the IoT contexts wherein they are utilised. The appropriate design of a trust management system is critical, therefore, and must be evaluated based on verifiable trust properties which may be elicited based on an empirical study of the existing body of work. Thus, the following subsections give a review of existing trust models for IoT and generic reputation models, with emphasis on their perceived strengths and shortcomings.

2.1.1 Existing IoT-Centric Trust Models

The peer-to-peer (P2P) nature of collaborative IoT applications means that there is no central authority. Modelling and evaluating trust in such contexts is usually difficult [29]. Collaborating peers will often be strangers to one another, having no shared history between them. Centralised TMSs should therefore not be considered in a collaborative IoT context. While a centralised approach to trust management is chosen in [30], the use of multiple trust management servers in different geographical locations is assumed. This means that all nodes must be registered under one of these servers and that the servers themselves are owned by a single entity. It does not take into consideration of collaborative applications that may be performed without access to the Internet. Indeed, the work focuses on service provisioning rather than collaborative situations, and only proposes a framework for different services without specifying solutions for individual contexts.

The concept of social IoT trust is utilised in [15], [32], [35], [36], where it is argued that existing social relationships between owners must be taken into account in trust management. This usually involves sharing some confidential information, such as user identities, locations, and other relationships. This opens the door to personal, malicious attacks from bad peers [37]–[39]. While trust is a human concept, it also depends on the context in which it is used. In the IoT context, the trust is between the interacting nodes, which may sometimes be required to exchange some information for identification and trust computation. However, this should be done in a transparent and non-intrusive manner that maintains the privacy of non-relevant information [26].

In [40], a trust management model for IoT based on fuzzy reputation is proposed. However, the model is specific to WSNs and only evaluates objective properties of packet forwarding/delivery ratios and energy consumptions [30], [35]. Thus, the model cannot be applied to collaborative IoT scenarios without some extensions. Furthermore, it neither properly models trust as a decision of the trustor nor considers the subjective properties of the trustee. The trust model in [7] is designed specifically for health IoT systems and cannot be applied to collaborative IoT contexts.

A detailed trust model for social IoT systems is presented in [15]. However, the model lacks a distinct trust decay function. Instead, two parameters are introduced; one weights past experiences versus direct assessments and the other weights recommendations versus past experiences. This introduces several problems in the trust computation problem. First, every direct assessment that is followed by a recommendation reduces the importance of past trust because it is weighted twice in both interactions. This does not allow for graceful degradation of trust. Moreover, if the trustor receives several consecutive recommendations on the same node, the impact of the past trust score and the trustor’s direct assessment on that node rapidly declines with each recommendation, as is the case in [41]. This is the case even if the trustor’s direct assessment were made about the same time as the recommendation. Thus, malicious nodes can come together to influence the trust rating of one node with another node. Also, the model does not consider the time value of trust in that the recommendation made, even if genuine, could be based on an interaction further in time than the trustor’s last direct assessment of the node on which a recommendation is being received.

Boa and Chen proposed a dynamic TMS and extended it to trust-based service composition in the IoT [35], [42]. The work considers three parameters to derive a trust value: honesty, cooperativeness as a service provider, and the community-interest of the nodes. The model includes a weighting factor to determine the relative importance of recommendations based on the trust level of node providing the recommendation. This factor can be dynamically increased to improve the resilience of the system with respect to the proportion of good peers and malicious peers. This however subjects the system to opportunistic service and on-off attacks [43], where a malicious node can provide good service but bad recommendations about other nodes. This is a consequence of evaluating a node’s trust score entirely on the subjective opinions of some other nodes, as will be shown in the next subsection.

There is also the need to consider the temporal nature of trust. It is usually useful to store trust scores from past interactions and utilize them in making trust decisions in the future, thereby building a trust history. This idea is widely employed in trust models to aggregate trust values over time. However, as is the case with recommendations, this notion may be abused by malicious nodes. If trust is to be a reliable assessment of the performance of nodes on functional properties, then it is necessary to track the behaviour of the nodes with respect to such properties and to detect and respond to changes over time. There is, therefore, the need for a trust decay function such that previous trust values degrade gracefully over time. This is also required to prevent on-off and opportunistic service attacks. In most existing trust models, however, trust decay is not considered.

Boa and Chen extended their previous work to service oriented architecture (SOA) based IoT systems and service management in social IoT in [32] and [15] respectively. The new model focuses on social trust based on the parameters of friendship, social contact, and community of interests. This model is not feasible for use in collaborative IoT contexts, as previously argued. Moreover, as it is based on [35] and [42], it inherits the limitations of subjective opinions discussed above. The nomadic, adhoc nature of IoT collaborations implies that collaborators may have no previous transactions with one another. In these cases, the use of a TMS solely based on reputation, such as EigenTrust [44] or PeerTrust [45], is not a good solution for several reasons, as will be discussed in the next subsection.

2.1.2 Traditional Reputation Models in IoT Contexts

While reputation is an integral part of trust, the two are not equivalent. The reputation of a person or device usually depends on the subjective views of others. In a largely decentralised architecture such as collaborative IoT, there is no standard way to determine whether the present reputation score of the device was not bought or given by a group of malicious peers. Since there may be no central database to keep track of reputation ratings, it is not always feasible to find out which peer contributed a ranking to the present overall score. A reputation-based system works in large P2P networks, social networks such as Facebook, and e-commerce applications such as eBay because there is a centralised trust authority and database [29]. This makes it possible to track the consistency of the rankings of every peer in the system. In the case of social networks, the nodes or peers are linked based on social trust.

It is necessary to distinguish between conventional social trust and the notion of functional trust considered in this research. The social trust between nodes usually refers to the degree or strength of the connection between them [46], [47]. Consequently, factors such as similarity, colocation, friendliness, and honesty are primary determinants or parameters of the trust score of the relationship. Also, the trust relationship tends to be symmetric and the trust score between the nodes is approximately equal in both directions [48], [49]; that is, if node A trusts node B, then it often implies that node B trusts node A, and the trust score of A on B is approximately equal to that of B on A. This is demonstrated in many social networks, where a "connection" or "friendship" between two parties on the network must be mutual.

Furthermore, the social trust network is usually transitive; thus, if A trusts B and B trust C, A is quite likely to trust C, even if there is no direct trust link between A and C presently. This indirect trust link is a based on the recommendation about C, from B to A. Recommendations are also known as *referral trusts* and are used to propagate or extend an existing trust network or to cold start a new one [50]–[52]. It is important to note that recommendation is not a trust parameter in itself; however, it is used to augment the trust computation process where the required information to directly compute a reliable trust score is either unavailable or incomplete [52]–[55]. In the hypothetical example above for instance, A cannot directly compute a trust score on C as there is no direct trust relationship between them. However, based on the recommendation from B, a trust score can then be assigned by A to C, which will be updated once a direct trust link between A and C is formed. A real-world example of this in social networks is the "friend-of-friend" feature, which is used to recommend new connections and content (e.g., posts or tweets) to a user.

Functional trust, on the other hand, refers to the degree to which a trustor believes that the trustee is both competent and willing to execute required task(s) reliably in a specific functional context, which defines the trust scope [56], [57]. It is dependent on the trustee's ability to perform certain functions and its historical performance as measured directly by the trustor [58]. It is immediately evident that, unlike conventional social trust, functional trust is asymmetric [59]. For example, the fact that A trusts B to perform a certain task reliably does not imply that B trusts A to perform the same task to the same degree of reliability, or even at all. Hence, functional trust is also non-mutual [60], and usually non-transitive [61], [62]. However, recommendations can be made, just as in social networks but it is not explicitly used to extend the trust network until a there is a requirement to establish a new, direct trustor-trustee, relationship. Therefore, recommendations are only necessary at the start of a new trust link (i.e., where there is no trust history) and are usually discarded in favour of direct assessments once the relationship is formed.

Functional trust is usually scored more on objective criteria of the associated context, and less on subjective criteria [63]. An objective trust criterion is one which is based on a functional parameter of the context and can be assessed quantitatively according to some metric or rule that has been defined within that context, thus ensuring its measurement is free from bias. On the other hand, a subjective trust criterion is assessed based on the bias of the trustor. For example, in a ride-hailing app, the distance driven on a journey is an objective parameter because its assessment is standardised; there exists a clear definition of distance and its measurement. However, the cleanliness of the vehicle (as measured by the rider) will be subjective and biased to the opinion of the rider if there is no defined metric or standard for its measurement. Therefore, a parameter may be subjective in one context but objective in another depending on the existence of a standardised assessment and scoring system in that context. Also, it is possible that a social trust factor can be a criterion in a functional trust context *if it is relevant to the function or task to be performed in that context.* For example, while colocation can be a factor used to extend a social trust network, it could be a functional parameter in a ride-hailing context if it is deemed important to the fulfilment of contextual tasks (in this case, getting a driver quickly to the user). In summary, functional trust relationships are formed based on the ability to fulfil tasks in a specific context and are isolated to that context.

It can be seen that reputation-based trust systems, such as [64], are entirely based on the trustors' subjective opinions which tend to be reinforced through an inherent feedback mechanism. This works well in large networks due to the "wisdom of the crowd" [65]. It is highly likely that the opinions of 1000 people about a seller on eBay will be a true reflection of the seller's activities. In a collaborative IoT scenario, however, the number of peers involved is small. The opinions of such a small number of rankers can be easily influenced and may not truly represent a trustee's trustworthiness. The feedback mechanism can cause multiple counting of the same behaviour, leading to aggravated rewards or punishments. Therefore, reputation should not be used alone for trust computation in such contexts.

The use of entirely subjective opinions of others to determine trust scores presents yet another problem in a collaborative IoT scenario. Take a collaborative download application as an example. Each possible helper peer may advertise the price charged per bandwidth used. A peer may receive a low ranking solely based on a higher price. This does not consider (and may not be a true reflection of) the helper peer's objective qualities in estimating its trustworthiness. The higher price may be a consequence of faster and better service offered. When this level of service is needed, the previous ranking will affect the trust score of the helper peer and may prevent another peer from patronising its service. Traditional reputation systems mitigate this issue by providing some feedback on rankings. This is achieved by eliciting reviews or by providing categorical scores alongside an overall trust score. This provides additional insights to the trustor and leads to a better trust decision. However, such a level of detail may not be feasible in IoT environments due to the limited computing requirements.

In [66], a recommender system is enhanced by adding a trust layer. However, the method assumes prior friendship between nodes and therefore only considers social trust parameters. Lastly, because reputation based TMSs use recommendations, they make the collaborative sphere more vulnerable to bad-mouthing and good-mouthing attacks [35]. This introduces an unnecessary bias into the trust model. It also corroborates the authors' argument against the assessment of trust solely on subjective opinions in a collaborative IoT context.

In summary, most existing trust models in the IoT are primarily based on recommendations, with the inherent risks as highlighted above. In contrast, our proposed trust model emphasises the parametrisation of trust. The trust parameters are based on the functional properties of nodes which are relevant to the application context. Recommendations are only used initially to augment the trustor's assessment. In addition, the process of trust decay is clearly and adequately modelled, which is an improvement upon existing models.

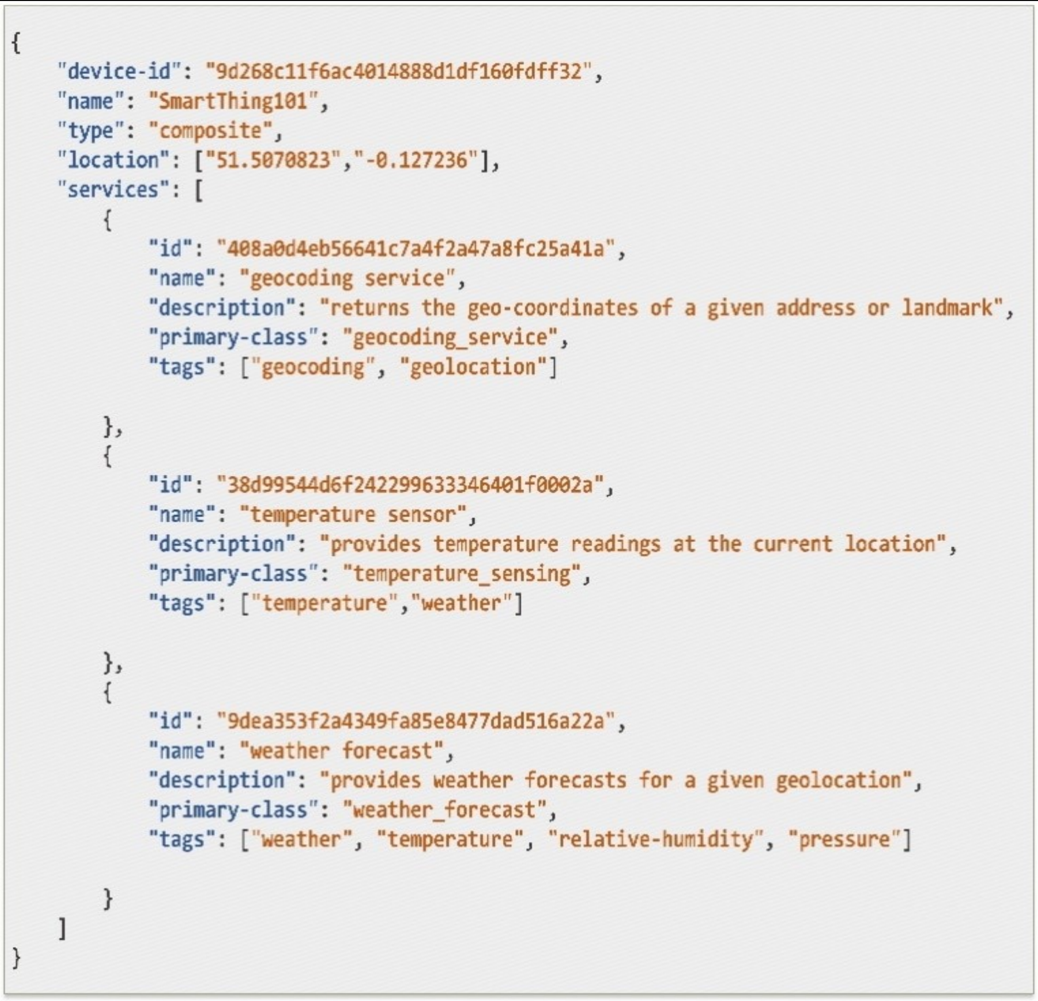
2.2 Trust-Based Service Composition and Provisioning

This section presents an overview on service composition in the IoT and its implications for trust modelling. In general, trust estimation in most IoT models focuses on recommendations impacted by social relationships. Little work has been done in other aspects of trust management, such as trust decay, trust parameter selection and the weighting of trust parameters. Trust parameters should be based on the relevant functional properties of the service context that determine the reliability and quality of an SP's services. Given that service compositions are based on the task(s) demanded by the user, the trust score of the composed service must be based on the efficient, reliable, risk-minimized completion of the task(s). While most of the work done on trust management has focused on trust evaluation of individual nodes, little consideration has been given to determining how to use the trust values of SPs to estimate the trust value of the composed service (that is, a bottom-up approach). A review of the existing literature suggests that no work has focused on the decomposition of trust scores assigned to composed services (that is, a top-down approach).

A suitable trust model for service compositions must include methods for reliable trust composition; it must adequately estimate the trust value of the composed service, based on the user's utility preferences and the current trust scores of candidate SPs. Also, it must adequately model trust decomposition; it should accept the trust value given by the user upon consumption of the composed service and decompose that value to update the trust scores of the underlying services in an impartial, appropriate, and transparent manner. These are discussed in detail in the subsections below. Section 2.2.1 gives an overview of service composition concepts. In Sections 2.2.2 and 2.2.3, we analyse the characteristics of various categories of service compositions and their effects on trust evaluations. In Section 2.2.4, we briefly discuss consensus mechanisms; finally, in Section 2.2.5, we review existing trust-based service composition models, evaluating their suitability for IoT contexts.

2.2.1 Service Compositions in the IoT

The IoT provides novel ways by which users can interact with things around them. The data received from various sensors in the environment can be utilized by multiple actuators to achieve a desired result. Sensors and actuators have a broader meaning in this context and are not limited to traditional electrical devices. For example, a refrigerator may "sense" it is empty of some food stuff and "actuate" an app on the owner's phone to create a shopping list of such items. This list may be passed to an online grocery delivery service at regular intervals that are determined by a bot. At such intervals, the bot “actuates” the online service to create an order and deliver the grocery. This concept of “social sensing” is one of the possibilities of an SOA-based IoT. Usually, the service requested by a user will entail the collaboration of several microservices, as in the previous example. Therefore, a bespoke service will be composed for the user, using two or more microservices which may be offered by different SPs. This architecture offers several advantages that are key to the realisation of the IoT vision, such as modularity, increased reliability and technology heterogeneity and interoperability. The requirements for the adoption of this architecture are being extensively investigated.



```
{
    "device-id": "9d268c11f6ac4014888d1df160fdff32",
    "name": "SmartThing101",
    "type": "composite",
    "location": ["51.5070823","-0.127236"],
    "services": [
        {
            "id": "408a0d4eb56641c7a4f2a47a8fc25a41a",
            "name": "geocoding service",
            "description": "returns the geo-coordinates of a given address or landmark",
            "primary-class": "geocoding_service",
            "tags": ["geocoding", "geolocation"]

        },
        {
            "id": "38d99544d6f242299633346401f0002a",
            "name": "temperature sensor",
            "description": "provides temperature readings at the current location",
            "primary-class": "temperature_sensing",
            "tags": ["temperature","weather"]

        },
        {
            "id": "9dea353f2a4349fa85e8477dad516a22a",
            "name": "weather forecast",
            "description": "provides weather forecasts for a given geolocation",
            "primary-class": "weather_forecast",
            "tags": ["weather", "temperature", "relative-humidity", "pressure"]

        }
    ]
}
```

Fig. 2.1 A JavaScript Object Notation (JSON) specification of a smart device’s services

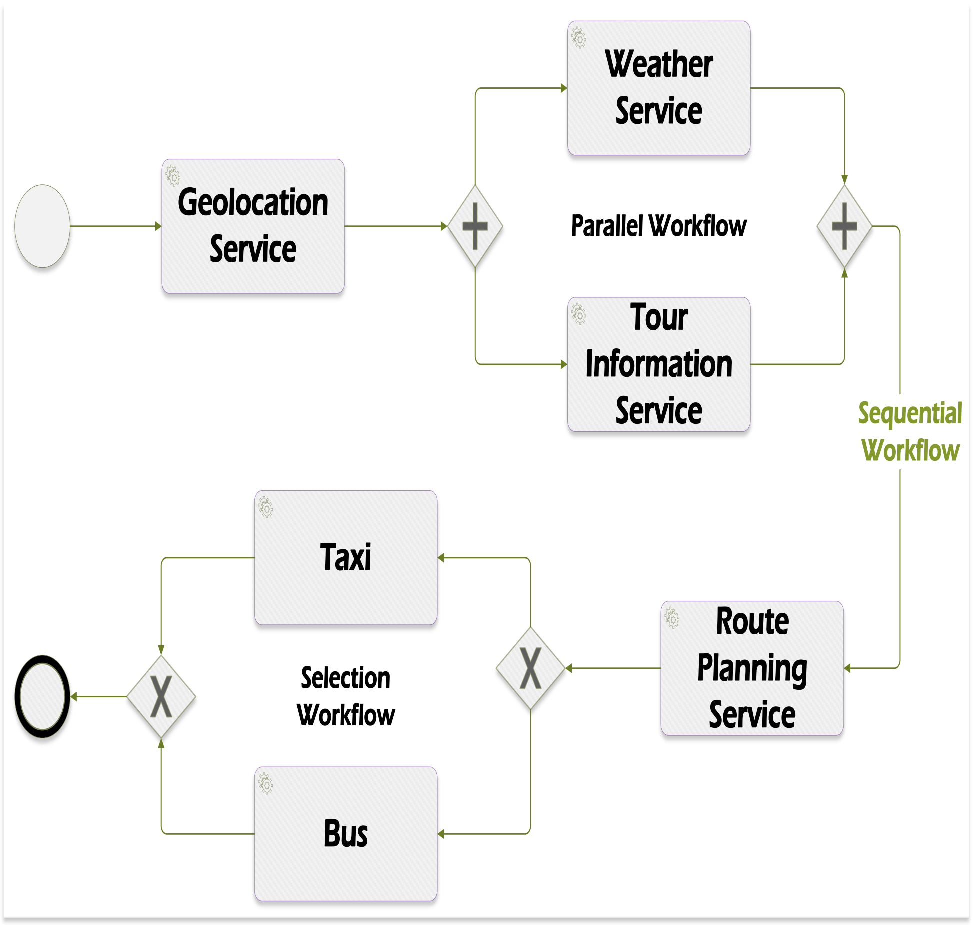


Fig. 2.2 A Business Process Model and Notation (BPMN) model of a hypothetical service composition

Fig. 2.1 illustrates a JavaScript Object Notation (JSON) representation of a smart device detailing its identity, location, and a description of the services it provides; this representation is an effective, lightweight method by which the smart device advertises its presence and capabilities to a composing platform in a universally parsable format. Fig. 2.2 illustrates a hypothetical service composition using Business Process Model and Notation (BPMN), which will be explained later. The manner and order by which services are composed plays a role in the trust evaluation strategy. Basically, service compositions may be categorised in two ways. First, the mix of different classes of services involved may be considered. A service class is a logical unit into which SPs offering a similar service may be grouped together. It is an abstraction of a service type. Every service is a member of at least one service class. Secondly, service compositions may also be classified according to the sequence or workflow in which the services are ordered. We discuss both in detail in the following sections.

2.2.2 Service Classes

In connection with the service class, there are two kinds of service compositions: homogenous and heterogeneous. In a homogenous composition, all the underlying services are from the same class. An example of this would be an SR requesting a list of the top five taxi car services within the vicinity. The middleware presents the user with the services estimated to be the most trustworthy. From this list, the user determines which taxi service to order, according to their subjective preference. Another example would be an application that allows the SR to request the assistance of some SPs in the collaborative download of a large file by pooling their bandwidths. In these cases, it is a simple collaborative application similar to those discussed in [67]; and a simple trust model may be applied by the user to select the most trustworthy taxi service. The only function of the middleware, in this case, was to reduce the number of potential service providers that the SR had to directly evaluate from, say, twenty taxi services to just five. It is important to note that each service class is a self-contained collaboration context, to which a suitable IoT trust model may be applied in selecting the most trustworthy SPs.

In a single-service trust model – such as used in a monolithic service - the trust evaluation is performed by nodes on one another. A trustor node evaluates trustee nodes based on their service performance and quality in line with the agreed trust parameters of the collaborating context and weighted by the trustor's subjective preferences. The trust score of the service is equivalent to the trust score of the node; there is no functional difference between a node and its service, as the nodes are assumed to be offering only one service of the same type. A distinction is made between the node (smart thing or device hosting a service) and the underlying service that it provides in a composition because we assume a node may host several services that it may provide to different compositions at various times. It may even provide multiple services of different classes to the same composition. Therefore, the trust scores are assigned to each underlying service rather than the node, and this is what is implied whenever the trust score of an SP is mentioned. The reason for this is that the provision of a good service in one class by a node or device does not necessarily mean the node will provide a similar level of performance in another class, although the node may not be malicious. Therefore, the trust model must distinguish between truly malicious nodes and other nodes which are benevolent but perform ineptly in one service class [30].

In a heterogeneous composition, the underlying services are mixed and matched from different classes, such as the composition of a geolocation service and a weather forecast service. In this scenario, the trust model must identify the most suitable SPs within each service class required for the service composition. The SPs are chosen based on their scores on relevant parameters and in line with the SR's subjective preferences. It is evident that heterogeneous compositions have far much more applications in the IoT than homogenous ones. Consequently, they are the major focus of service composition algorithms. As noted earlier, a node may simultaneously offer multiple services from more than one class. In such cases, there may be conflicts between the services which could lead to the degradation of the performance of some or all the services. For example, there might be a delay between the halt of one service and the instantiation of the next service. Also, there are energy and network implications. These are discussed in [10], where the authors introduce an algorithm to solve this problem in an energy-efficient manner. It is assumed that devices are involved in only one composition at a time, although they may provide multiple services to that composition. If there is a decrease in the QoS of one of the simultaneously provided services, the trust model should detect and adapt to this change by reassessing the most suitable SPs within that service class. Presumably, there is some incentive for the IoT devices participating in the composition. Therefore, the devices would be configured to provide services in such a way that the reward is maximized. A suitable trust model would ensure that this maximization is subject to the provision of an optimal service level required for the satisfactory fulfilment of the SR’s requests.

2.2.3 Service Workflow

The workflow refers to the order in which services are performed and composed. Generally, there are three basic types of workflows, which may be used in any combination. First, it may be a simple case of selection, where the SR receives a list of SPs along with their trust ratings. The SPs may not be from the same service class. Take an example of a user visiting a smart city for the first time. On receiving beacons about a special attraction taking place later that day, the user (the SR) queries a trust model for the top three ways to get to the concert at a certain time and from a certain location. The SR may receive a response listing a taxi service, a shared-ride service, and a rent-a-cycle service, depending on the parameters specified by the SR, and then chooses from the offered options. On receiving the choice of the SR, the trust model then proceeds to select the most suitable SP in the service class specified by the SR's choice. As can be seen, this is an interactive session between the user and the middleware. Rather than automatically selecting the best SP from across these service classes, the SR makes the decision on which mode of transportation to use. The initial response of the middleware may contain other relevant contextual information – such as the price range, average wait times, and the safety index – on each of these service classes, thereby assisting the SR in making more robust decisions.

Services may also be composed in a parallel workflow, where two or more SPs simultaneously provide the same or different services. As an example, suppose a user-bot (SR) requests all the available weather information for a specific location. On receipt of the request, the middleware selects and retrieves the relevant information (such as temperature, pressure, relative humidity, and wind direction) from the most trustworthy SPs in each service class associated with the weather data. This information is then aggregated and returned to the SR. The collaborative download application previously mentioned would also require a parallel workflow, since all the SPs would be downloading different byte ranges of the same file concurrently. A parallel workflow could also be used where the SR concurrently requests information about a topic which is subjective by nature. For example, suppose an SR queries the middleware to return a list of the best attractions in the city. Then, three different SPs could be queried to provide tour information service, and an aggregate of the responses could be returned to the SR. This information could be used to initiate another service request, depending on the SR's interest.

Thirdly, services may be composed in a sequential workflow where the results from one service are provided to the next and so on. For example, a user requests the relative humidity at a street address. The address is inputted to a geocoding service which returns the geographical coordinates, which are in turn inputted to a weather type service providing relative humidity information. The results of the second service are then returned to the SR. Unlike the other workflows, SPs cannot respond to or fulfil service requests concurrently in a sequential workflow. However, in most applications of IoT service compositions, this would be the most occurring workflow; microservices would often process the results of other services and then deliver the results to yet another service in the fulfilment of the user's request. A sequential workflow also makes it possible to transparently mix services (and other workflows) in innovative and resourceful ways. For example, a service composition workflow may contain a selection workflow, then execute services in parallel based on the user's response, and provide the output of those services to another service for processing before delivering the results to the SR. An example of such a complex workflow is illustrated in Fig. 2.2. In this example, a geolocation service returns the coordinates of the SR's location to the weather and tour information services, which are composed in parallel. The aggregate information from both of those services is used to derive the best place in the city to visit under the prevailing weather condition. This information is inputted to a route planning service, which requires the SR to select either bus or taxi as the preferred mode of transportation. Therefore, this example contains all three kinds of workflow. It is also possible to have cascading service compositions, where a service composition functions as an underlying service itself in a higher-order composition.

2.2.4 Consensus Mechanisms

Given the distributed architecture and heterogenous nature of IoT, the middleware layer may implement a consensus mechanism to ensure integrity and reliability in the system [68], [69]. This is particularly necessary where multiple trusted SPs provide different values for the same data point such as temperature, or varying workflows for the same service composition. It will be required to decide which data point is the most accurate, and to agree on the best workflow among suitable alternatives. It is important that the mechanism chosen must be resilient. Specifically, it should be byzantine fault tolerant, ensuring integrity and agreement always [70], [71]. Also, the characteristics of an IoT platform (such as requiring low energy and computation overhead) must inform the choice of a consensus mechanism. Given that participation is dynamic (SPs and SRs can join or leave the network at will) and the platform is decentralised, an ideal consensus protocol will be permissionless, encourage direct participation in consensus formation from the majority of nodes, and be resistant to sybil attacks.

Recently, blockchains have been increasingly utilized as middleware layers for service compositions in IoT applications [72]–[75] and in trust management systems [76]–[78]. Several consensus mechanisms have been proposed for blockchains since it was first described in 2008 [79], the most popular of which is the Proof of Work consensus mechanism, followed by Proof of Stake protocols [80]–[83]. Neither of this is suitable for IoT applications due to high energy usage in the former [68] and the tendency for super nodes (which in turn decreases decentralization of the network) in the latter [69]. However, other consensus mechanisms exist that may be more suitable, some of which have been implemented for IoT;

these include Tangle [84]; Proof of Comprehensive Performance [85]; Proof of Authority [86], [87]; Proof of Personhood [88]; Proof of Identity [89]; Proof of Reputation [90]–[93]; Proof of Benefit [94], [95]; Proof of History [96]; Proof of Elapsed Work [97]; Proof of Space (or Capacity) [69], [98]; and some variants of Byzantine Fault Tolerance consensus mechanisms [70], [71], [82]. Conversely, some authors have proposed using trust-based consensus mechanisms in blockchains [99]–[102]. In this thesis, details of the middleware are generally abstracted and only briefly highlighted, in line with the research scope.

2.2.5 Existing Trust-Based IoT Service Composition Models

Only a few trust models for service composition exist in the literature. This is due, in part, to the complexity of modelling trust in a decentralized manner [29]. Moreover, no previous works have provided methods for transparent trust composition and decomposition in the IoT. Existing models measure trust mainly on social trust parameters, such as friendship, honesty, cooperativeness and kindness, based on direct observations and recommendations [17], [32], [35], [103]. The reasoning behind this modelling paradigm is the notion of the "social" nature of service oriented IoT applications. Hence some social characteristics of human relationships are inaccurately applied and modelled in the IoT. It is important to observe that the social nature of the IoT is much different from human social interactions. In the IoT, the focus is much more on the ecosphere of humans, the environment, and smart things. Users interacting with services need not have a relationship with the devices providing such services, or their owners. Depending on the kind of service being provided, the SRs may not need to be co-located or share the same community of interest with SPs or their owners. The primary goal of the

SR in trusting a service or SP is the provision of a reliable, quality service that meets the specification of the SR without posing any risks. Thus, while a social interaction exists in the IoT, the relationships formed, and the parameters for measuring those relationships, are based on the required functional properties of the context of interaction; that is the request and provision of a service. As such, the dominant type of trust in this context is functional trust, which has been previously described and contrasted with social trust.

Moreover, although trust is primarily a social concept, it is not reasonable to directly apply the aspects of trust, as it applies in human relationships, to evaluating computational trust without some conceptual adaptation represented by mathematical models [28]. Even in human circles, the meaning of trust must be implied from the context and depends on the subjective assessment of the trustor [33], [34]. For example, the contextual meaning of trust in the statement “I trust my doctor” is much different from what is implied in the statement “I trust my spouse”. In both cases, however, the trustor makes a quantifiable evaluation of trust based on parameters that are relevant to the context and accounting for any risks that may be encountered in interactions with the trustee. A person trusts another person to be his or her doctor based on qualities the trustor believes a doctor should have, and not on generic personality traits. The trustor has a social relationship with the trusted doctor, but only with reference to the trustee’s profession and the service context. The trustor may trust this trustee in other spheres such as personal friendship, or as a husband or wife. This introduces the notion of multivariate trust, where a trustor has a trust relationship with a trustee in multiple trust contexts. Hence, we conclude that even in human relationships, trust in many contexts is predominantly functional, except for some cases of absolute trust or distrust. Therefore, in adapting trust to the SIoT, the notions of context and function must be preserved because they form the basis of functional trust [104]. This does not preclude the overall subjective nature of trust however, as trust scores on functional parameters may be weighted according to the trustor's subjective preferences.

In [22], a self-enforcing, privacy-preserving and decentralised TMS for SIoT is proposed. The protocol makes use of non-interactive zero-knowledge proofs in securing the network and for the reliable update of trust scores. However, the cryptographic protocol used is computationally intensive. While the computational overhead is manageable on systems with adequate computing resources, it is impractical for use on IoT devices which typically have limited computing resources. Also, the model does not satisfy the trust resilience property as it does not adapt to changes in the behaviour of a node during a trust session. Moreover, the model neither assigns weights to different trust parameters to indicate the trustor's subjective preferences, nor includes trust parametrization. Therefore, it cannot be applied to different contexts. In [105], a trustworthiness inference framework for SIoT is proposed based on familiarity and similarity trust, with contextual information based on time and location. The model does not consider the notion of functional trust or service contexts, which are important to guide service compositions. A trust-based service architecture for IoT is proposed in [106], with emphasis on improved efficiency of IoT services. However, the model operates in a centralized cloud architecture, thus limiting its applicability to service compositions in the IoT. Also, the model contains no notion of trust parametrization or service contexts, neither does the model in [107].

In [108], a trust architecture for software-defined networks in the IoT is proposed. The model uses reputation evaluation for trust establishment, with no notion of objective, functional parameters on which trust could be measured in a service-based application. In [64], a reputation-based trust system is proposed for IoT applications. However, a rigorous analysis conducted in [67] shows that reputation-based models used in the IoT are vulnerable to trust-related attacks such as bad-mouthing, ballot stuffing, and opportunistic service attacks, especially in a decentralized architecture. A similar argument may be applied to the trust-enhanced recommender system proposed in [66]. TMSs for dynamic trust management for SOA-IoT applications and service management in SIoT are proposed in [15], [31], [32]. The models produced are similar and measure the trust between nodes based on similarities in friendship, social contact, and community of interests. Functional constraints of the service contexts are not considered. Likewise, the models do not account for an SR's preferences and requirements, which should guide the service composition in an ideal trust model. Indeed, the models focus on achieving "subjective trust" by utilising a recommendation score that is primarily based on similarities and relationships between the owners of the IoT devices. This is contrary to the actual social nature of service applications in the IoT, which should be based on the service context. However, the models do consider transparent trust composition for the IoT based on the workflow, using a method derived from reliability/fault analysis. Thus, the trust score of a service composed of two microservices in sequence (series) is given to be the product of trust scores of the microservices. This assumption does not hold in many service applications, as discussed below.

It is possible to compose a service of high trust from microservices which have low or average trust scores. Take the case of a pizzeria which provides a pizza ordering service, makes excellent pizza but has slow delivery times. As a result, it is given an average trust score by an SR that gives significant weighting to delivery times. Suppose this SR also has a trust relationship with a ride-hire service which has exceptionally low wait times but rude drivers. The SR is uneasy for the duration of the ride because of the driver’s continuous boorish remarks; therefore, the SR gives a low rating to this service. Suppose that the two services are composed sequentially with the pizzeria producing the pizza and the fast ride-hire service delivering it to the SR. If the composition is done transparently to the SR, then the composed service would have a high trust rating, because the pizza is delivered faster, and the SR does not hear the driver’s remarks. This is contrary to the results that the reliability formula mentioned earlier would have produced: *low trust X average trust = lower trust*. Also, reliability analysis states that the reliability of a group of components is increased when the components operate in parallel, for redundancy. However, this does not necessarily apply to trust computation in IoT service composition. As an example, suppose that an SR requests a list of the top attractions in a city, and results from the two highly trusted information services are composed such that only items which appear on both lists are contained in the response to the SR. However, this may increase the response time and energy usage; consequently, the SR may assign a lower trust score to the resulting composed service.

From these examples, it is evident that to transparently compose trust in a service composition, there must be a mechanism to elicit the weighting that the SR assigns to each parameter, such that parameters irrelevant to the service composition (such as the driver’s behaviour in the previous example) would have little or no effect on the overall trust score. Thus, there may be a slight change in context for a service composition, dependent upon the workflow of the composition. Moreover, most of these models do not consider the temporal nature of trust and its effect on the decay of previously accumulated trust. It is necessary to track the behaviour of SPs in relation to the functional trust parameters and to detect and respond to changes over time. In summary, there exists a need for a suitable trust model for service compositions in the IoT, which correctly parametrizes trust, considers trust decay and provides mechanisms for both transparent trust composition and decomposition in addition to other relevant trust properties, which are enumerated in the next section.

2.3 Ideal IoT-Centric Trust Model

IoT applications generally consist of collaboration and service provisioning [17]. Therefore, an ideal IoT trust model should provide a balance between security, functionality and usability while considering the constraints imposed by the limited resources available to most IoT devices. Based on the previous work done in [26], [32], [67], [109], [110], a list of suitable attributes for an ideal TMS for IoT are enumerated below:

1. *Platform Consideration:* IoT devices have low computing capabilities, are pervasive and usually employ a decentralized architecture. Therefore, a TMS for the IoT should be lightweight, resilient, scalable, decentralized, and adaptable to different service contexts.

2. *Trust as a Decision (TaaD)*: The concept of trust in the IoT is essentially functional trust, and this should be modelled as a decision-making process with the objective and subjective trust properties of the trustor taken into consideration. Each trustor (i.e., SR) should be able to decide the importance of a trust criterion or recommendation. Thus, different trust values may be computed by different SRs on the

same node or service, depending on each SR’s subjective trust dispositions.

3. *Contextual Trust Parametrization*: The TMS should incorporate the necessary objective (quality of service, QoS) and subjective (social) properties of each service class in the composition as trust parameters so that the SR can make a well-informed trust decision as it applies to the current contexts. This also ensures that the model is robust and adaptable to different contexts.

4. *Trust Persistence*: Natural trust considers historical interactions in the trust relationship, which are accumulated and utilized to make a trust decision in the present. Hence, the TMS must provide an effective means for persisting trust values, considering both the low storage and decentralized architecture of IoT devices.

5. *Trust Decay*: Trust is temporal by nature. Stored or previous trust values degrade gracefully over time. A suitable function to evaluate trust decay should be included in the TMS.

6. *Risk Mitigation*: It should provide effective mitigation of self-promotion, bad-mouthing, ballot-stuffing, opportunistic service, and on-off attacks, as described in [17], [43], [104], [109]. The inclusion of a robust recommendation and belief function in the TMS would make it difficult for nodes to profit from malicious activities.

7. *Trust Accuracy*: This is a measure of how close the trust value computed for a node is to the ground trust. The ground trust for a node is the trust value that we would compute if we had perfect knowledge of its behaviour. The TMS must have a high degree of trust accuracy.

8. *Trust Convergence*: This is a measure of how long it takes for the trust value computed for a node to reach its ground trust and maintain it as long as the node's behaviour is consistent. The TMS should ensure trust converges quickly.

9. *Trust Resilience*: This is a measure of the ability of the TMS to adapt to changes in the trust community, such as an increase in the ratio of malicious SPs to good SPs. An ideal TMS should be sensitive to these changes and compute trust values in such a manner that it remains accurate by quickly converging to the new ground truth values of the affected nodes.

In addition to these, trust models for service composition in the IoT must also possess the following requirements:

10. *Transparent Trust Composition*: The trust model must include methods for estimating the trust score or trustworthiness of a composed service appropriately, considering the trust scores of the SPs, the service context, and the workflows involved in the composition. This should be done transparently to the SR; that is, the SR should not be aware of the internal details of the composition, or of the underlying services.

11. *Transparent Trust Decomposition*: Ideally, the SR would give a trust score after consuming the composed service. However, it has been established that the SRs will not interact directly with, or even know, the SPs in a service composition. Therefore, the TMS must incorporate methods to decompose the trust score from the SR and utilize it to update the trust scores of the underlying services in an impartial and transparent manner.

The basic components of a TMS are illustrated in Fig. 2.3 and shall be discussed in detail in the next chapter. The interactions of these components form the basis of the models proposed in Chapters 3 and 4, and these interactions shall be thoroughly explained in those chapters.

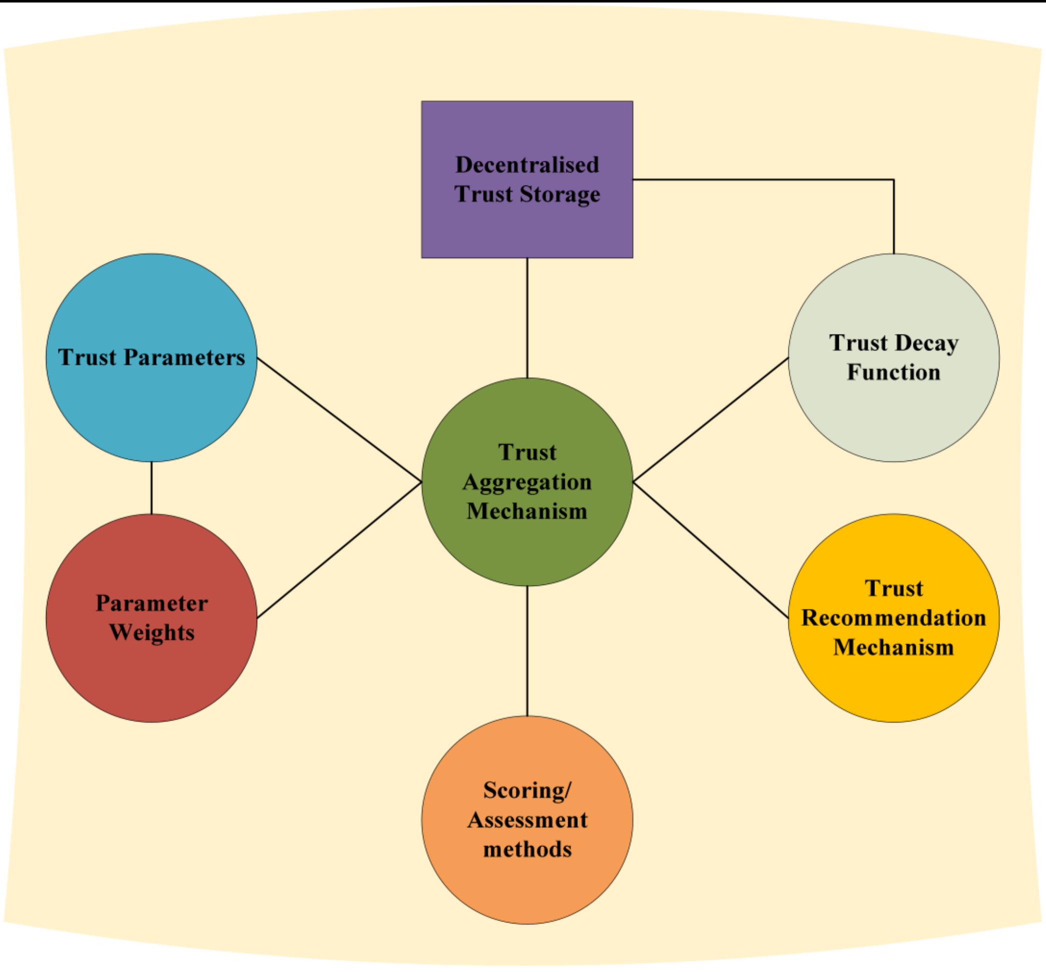

Decentralised Trust Storage
Trust Parameters
Trust Decay Function
Trust Aggregation Mechanism
Parameter Weights
Trust Recommendation Mechanism
Scoring/ Assessment methods

Fig. 2.3 Core components of a TMS and their interactions

2.4 Chapter Summary

In this chapter, an extensive review of the existing literature on trust aggregation, evaluation, and management for collaborative applications and service composition in the IoT was documented. The concept of computational trust was formalised, and key concepts were discussed.

The achievements and shortcomings of previous trust models for IoT contexts were detailed. The categories of service compositions were analysed with respect to the classes and workflows.

Based on the studies conducted, the necessary attributes and requirements for an ideal IoT-centric trust model were elicited and detailed. The rest of this work follows on from this by proposing trust models for different IoT contexts in accordance with requirements specified in this section. In the next chapter, a trust model for collaborative IoT applications is proposed, discussed, and evaluated.

CHAPTER 3 CTRUST: A DYNAMIC TRUST MODEL FOR COLLABORATIVE IOT APPLICATIONS

3.1 Model Design and Analysis

CTRUST is proposed as a suitable trust model to evaluate and manage trust between nodes in collaborative applications in the IoT. The trust a node has in another is based on its assessment of their current and past direct interactions and the recommendations it accepts from other nodes. Trust criteria form the basis on which assessments are made, and a trustor determines the weights of each criterion. A node can then compute trust scores which it uses to choose which other nodes to collaborate with. Trust scores are stored and are used to guide future interactions, although their importance declines over time. The model consists of trust assessment, decay recommendation and aggregation functions, all of which are discussed in detail in subsequent subsections. A high-level workflow of the model is illustrated in Fig. 3.1. The following gives an overview of CTRUST:

1. Trust would be composed of one or more trust criteria (parameters) relevant to a collaborating context. Each trust parameter could be objective (QoS) or subjective (social) in terms of assessment. An assessment of a node on a parameter is called a *partial trust score*.

2. A trustor would be able to assign weights to each trust parameter. The weights indicate the relative importance of each parameter to the trustor. Therefore, partial trust scores are weighted before trust aggregation.

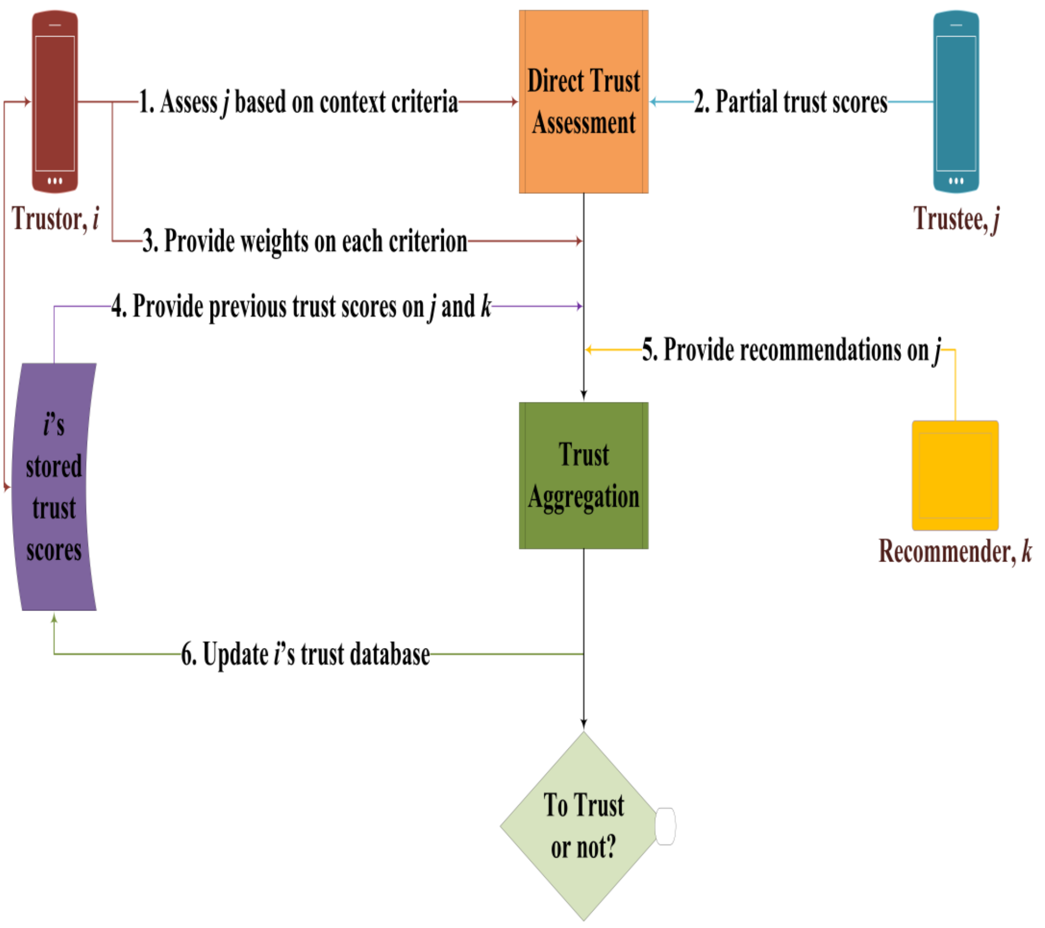

Trustor, *i*
1. Assess *j* based on context criteria
Direct Trust Assessment
2. Partial trust scores
Trustee, *j*
3. Provide weights on each criterion
4. Provide previous trust scores on *j* and *k*
5. Provide recommendations on *j*
i's stored trust scores
Trust Aggregation
Recommender, *k*
6. Update *i*'s trust database
To Trust or not?

Fig. 3.1 Flow diagram illustrating the basic steps involved in trust computation in the CTRUST model

3. Trust would be propagated in a distributed manner, with no intervening central authority. Each node stores its previously computed trust values and may accept trust recommendations on partial trust scored from other nodes.

4. A recommendation function is implemented to guide the degree of acceptance of trust recommendations on partial trust scores. We call this degree of acceptance *belief change*, and it is modelled based on social characteristics.

5. Trust scores decay over time based on a mathematically modelled trust decay function. We also define the points at which previous trust has decayed completely and when current trust has reached maturity.

6. A trust aggregation function determines how partial trusts are aggregated to compute an overall trust score for a node. The aggregation function chosen depends on the collaboration context. In this study, we used *a dynamic weighted sum* method. Trust updates (on partial trust scores) are *event-driven* and occur whenever nodes interact with one another.

The model may now be defined in detail. Let **C** be the set of all possible collaborating nodes under the current application or collaboration context.

T[C], the trust space over *C*, is then a sextuple expressed by the following notation:

$$T[C] \equiv \left[T_{ij}, P, W_i, V_{ij}, F, t_{\frac{1}{2}}(i)\right] \forall i, j \in C \qquad (3.1)$$

Where

T_{ij} is the trust score of node j as computed by node *i*;

$P = \{p_i, p_2, p_3, \ldots, p_n\}$ is the set of trust parameters or properties on which each node in *C* is assessed by other nodes;

$W_i = \{w_i(p_1), w_i(p_2), w_i(p_3), \ldots, w_i(p_n)\}$ is the set of weights on each parameter in *P,* as assigned by *i;*

$V_{ij} = \{V_{ij}(p_1), V_{ij}(p_2), V_{ij}(p_3), \ldots, V_{ij}(p_n)\}$ is the set of values denoting node *i's* assessment (partial trust score) of j on each parameter $p \in P$;

$F = f(W, V) \equiv T_{ij}$ is the trust aggregation function; and

$t_{\frac{1}{2}}(i)$ is the half-life of any partial trust score computed by i.

3.1.1 Trust Parameters

Our design follows a multi-criteria approach towards trust computation. A trustor makes a trust decision based on multiple criteria. Each criterion is a trust parameter. Parameters could either be objective or subjective. Parameters are considered objective if they are verifiably measured. Such properties include the speed of a transaction, reliability, rate of work, proximity, cost of a service, stake in the collaboration, etc. If p is an objective parameter, then for a node $j \in C, \left(V_{ij}(p)\right)_t$ is approximately the same if measured by any $i \in C$ at instant *t*. Subjective parameters such as honesty, cooperativeness or friendliness are assessed as perceived by the trustor. Therefore $(V_{ij}(p))_t$ does not have the same value from all $i \in C$, even if they all assessed *j* at the same instant, *t*. Therefore, the trust model supports both QoS and social trust parameters, which means a greater robustness and latitude of applications.

We do not explicitly specify the trust parameters in our model. This is because the choice of which parameters to use depends on the collaboration context and should be decided when C is set up. Suppose a service composition with *n* collaboration contexts,$C_1 \dots C_n$ is set up. One approach will be to populate the set *P* with all possible parameters for all the contexts wherein the trust model will be implemented. Then the weights of parameters which are not relevant to the current context can be set to 0.

3.1.2 Parameter Weights

The weights determine how important a parameter is relative to the overall trust score. Each node determines the weight of each parameter, based on their current subjective opinions. As a result, two nodes may have an equivalent value set, V, at a given instant, yet compute different trust scores, T, for a third node. The weights are assigned such that:

$$w_i(p) \in [0,1] \forall i \in C, p \in P \text{ and } \sum W_i = 1 \qquad (3.2)$$

Even though set P is the same for every node in C, a node can eliminate parameters that it does not want to consider by assigning them a weight of 0. The weights can be dynamically adjusted by the trustor at any time during a session of interactions. Nodes can update their record of set W at any time, according to changes in their perceptions of relative importance of each parameter. The dynamic weighting allows for more accurate modelling of human trust. The relative importance of the factors that determine the extent to which a person trusts another can vary greatly over time. Similarly, the relative importance of collaboration criteria can vary for each node from one session to another.

3.1.3 Partial Trust Scores and Aggregation

The set V_{ij} represents the normalised assessment of node j, by node i, on each of the trust parameters in P. The collaboration context defines the parameters, i.e. how they are measured and on what scale. Objective parameters such as rate of work or network speed are well defined and will be measured uniformly across C, while the measurement scale for subjective ones such as friendliness may differ from node to node. Each member of V_{ij} is a partial trust score as they determine the overall trust score, T_{ij}. The values are normalised to [0, 1] so that T_{ij} is also within the same range, and the normalisation method must be defined in C.

Three factors account for any $V_{ij}(p)$ at the current time: its previous value based on past interactions; the current, direct assessment of j by i; and indirect assessment of j by some other node k. These will be discussed in detail later.

The trust aggregation function, F, specifies how the partial trust scores are aggregated to compute T_{ij} and enables us to model trust evaluation as a decision-making problem and apply multiple-criteria decision analysis (MCDA) methods to solve it. In this scenario, a weighted sum function, $F = W \times V$, is used. Therefore,

$$T_{ij} = \sum_{x=1}^{n} w_i(p_x) \times V_{ij}(p_x) \forall i, j \in C, p_x \in P \qquad (3.3)$$

It can be observed that the derivation of this function incorporates objective and subjective properties of both the trustor and trustee. Furthermore, our trust model allows for the aggregation function to be left unspecified until the collaboration context is set up, which is expressed as $F = f(W, V)$. This is important because the context should determine the method by which partial trust scores are aggregated. Some contexts may require a product of weighted scores, or a more complex function such as Bayesian inference or regression analysis, to compute a trust value [109], [111]. Therefore, it is best to leave the aggregation function unspecified until the context is set up, as this makes the model robust and applicable to more contexts.

3.1.4 Trust Decay

It is necessary to model the impact, over time, of previous trust scores on the current trust values; this is achieved by introducing the concept of trust decay. The trust score, T_{ij}, gradually degrades over time when there is no interaction between i and j. In the social world, the longer we are further away from a person, the easier it is to distrust that person. This is so because we are not sure whether they still retain the values for which we admired them. Interactions help to re-evaluate our opinions of them on these values, and serve to reinforce the trust relationship, or score. Accurately modelling trust decay is exceedingly difficult, and there is limited existing literature on the subject. The following assumptions seem to hold true in the usual social context, and form the basis for the trust decay function of our model:

1. As the trust formation depends on partial trust values on each trust component, trust decay applies to these values and not the overall trust score, which may not correlate with the trust decay rate. The reason for this is that in the interval between interactions, the trustor's perception of the relative importance of some of the trust parameters may have changed.

2. The rate of trust decay is almost entirely subjective. It depends on the trustor's willingness to trust and the length of time or number of interactions the trustor requires to establish a node's behaviour in the present.

3. It is reasonable to assume that trust decays at an exponential rate in the absence of interactions [112]. The longer the period of inactivity between the peers, the greater the rate of decay.

4. When a new session of interactions is made in the present time, previous trust decays with every new interaction. This is so because the new interactions tend to form the trustor's new opinions and therefore, the trust score of the trustee. After a certain number of new interactions, past trust values may no longer be relevant to trust computation in the present.

The above assumptions provide the basis by which trust decay is incorporated into our model. Let $t_{\frac{1}{2}}(i)$ be the duration required for a partial trust score assigned by *i* to decay to half of its initial value, i.e. its half-life. The decay function follows an exponential trend and is represented by the following mathematical equations:

$$\begin{aligned}\left(V_{ij}(p)\right)_{0\to t} &= \left(V_{ij}(p)\right)_0 \times \frac{1}{2}^{\frac{t}{t_{\frac{1}{2}}(i)}} \\ &\equiv \left(V_{ij}(p)\right)_0 \times e^{-\lambda_i t} \\ &\equiv (\phi_i)_t \times \left(V_{ij}(p)\right)_0 \ \forall i,j \in C, p \in P\end{aligned} \tag{3.4}$$

$$\lambda_i = \frac{\ln 2}{t_{\frac{1}{2}}(i)} \approx \frac{0.693}{t_{\frac{1}{2}}(i)} \tag{3.5}$$

$$(\phi_i)_t = e^{-\lambda_i t} \tag{3.6}$$

Where

$(V_{ij}(p))_0$ is a partial trust score at the end of the last session of interactions between *i* and *j*;

$(V_{ij}(p))_{0\to t}$ is the current value of $(V_{ij}(p))_0$ after time *t* of no interactions between *i* and *j*;

λ_i is the decay constant for partial trust scores from *i*; and

$(\phi_i)_t$ is the trust decay multiplier for node i.

The multiplier is the proportion of the partial trust score that has not decayed after time, t, of no interaction. Equation (3.4) can then be simplified and rewritten as:

$$\left(V_{ij}(p)\right)_{0\to t} = (\phi_i)_t \times \left(V_{ij}(p)\right)_0 \; \forall i,j \in C, p \in P \qquad (3.7)$$

After an adequate number of interactions in a new session, the effective proportion of $(V_{ij}(p))_0$ that determines $V_{ij}(p)$ in the current session becomes 0, according to Assumption (4) above. At this point, the trust has attained maturity in that session. Trust maturity is discussed further in Section 3.1.6.

3.1.5 Trust Recommendations and the Belief Function

There may be times when node i is about to start a new session of interactions with node *j*, and it is currently in a session with another node *k*, which has some assessment on *j*. Node i may make use of this assessment to make an initial update of *j*'s partial trust scores prior to initiating interactions. This is called an indirect assessment, or a recommendation, on j by i, through k. The drawbacks of trust systems solely based on reputation or recommendations (see Section 2.1.2), must be avoided. A recommendation belief function is therefore required to determine the degree to which any node i accepts *k*'s recommendations on *j*. The following premises should be taken into consideration to model the belief function accurately:

1. The purpose of recommendations is usually to guide *i* as it attempts to initiate or re-initiate interactions with *j*. They either make the trustor initially more sceptical or more open to trusting *j*. After interactions

are made, k's indirect recommendations are quickly discarded, as i's direct assessment forms the basis of the trust score.

2. Recommendations do not necessarily affect i's trust relations with k, even if they are proven to be incorrect. k might have been misinformed. It follows then, that the best way to prevent good and bad-mouthing attacks is to minimise the effect of indirect recommendations on a partial trust score, and hence, T_{ij}.

3. Let the change between k's current recommendation and i's previous partial trust score of j on some parameter be ΔV. The smaller the absolute proportion of change, $|\frac{\Delta V}{(V_{ij}(p))_0}|$, the easier it is for i to accept. This stems from the observation that in a social context, we are less likely to receive a recommendation about a person if the recommendation represents a significant difference from previously observed behaviour.

4. The longer the time t that has passed since the last session of interactions between i and j, the more open i will be in accepting k's recommendation. This is because of trust decay; the longer the time t, the smaller the proportion, ϕ_i, of previous trust left. The value of trust is a function of time; the extent to which we would believe a value that implies a significant change in a person's behaviour depends on the time that has elapsed since our last interaction with them.

5. The greater the value of T_{ik}, or more specifically $V_{ik}(p)$, the more likely we are to receive the recommendation of k on j on some trust parameter, p. In a social context, we more readily believe recommendations on a subject from someone who we rate high on the same subject.

The belief function can be then derived mathematically from premises (3) – (5) above:

$$\text{Belief}, \beta_{ij\leftarrow k} \propto \left(1 - \left|\frac{V_{kj}(p) - \left(V_{ij}(p)\right)_0}{\left(V_{ij}(p)\right)_0}\right|\right) \tag{3.8}$$
$$\beta_{ij\leftarrow k} \propto (1-\phi_i)$$
$$\beta_{ij\leftarrow k} \propto V_{ik}(p)$$

$$\Rightarrow \beta_{ij\leftarrow k} = K\left(1 - \left|\frac{V_{kj}(p) - \left(V_{ij}(p)\right)_0}{\left(V_{ij}(p)\right)_0}\right|\right) \times (1-\phi_i) \times V_{ik}(p) \tag{3.9}$$

Where *K* is a constant. The value of *K* does not need to be verified. Since the change belief is a relative indicator of how much a recommendation is to be accepted, the exact value of *K* need not be known. Therefore, we set *K* = 1 so that at the beginning of a new session of interactions between *i* and *j*:

$$\beta_{ij\leftarrow k} = \left(1 - \left|\frac{V_{kj}(p) - \left(V_{ij}(p)\right)_0}{\left(V_{ij}(p)\right)_0}\right|\right) \times (1-\phi_i) \times V_{ik}(p) \tag{3.10}$$

This belief function indicates how much node *i* is willing to accept a recommendation on *j* from *k*, i.e., the weight *i* assigns to that recommendation. It therefore determines *i*'s indirect assessment of *j* through *k*, on parameter *p*, which is expressed as $\Psi_{ij\leftarrow k}(p)$, and defined by:

$$\Psi_{ij\leftarrow k}(p) = \beta_{ij\leftarrow k} \times V_{kj}(p), j \neq k \tag{3.11}$$

A node cannot provide recommendations on itself. This comprehensively defends against self-promotion attacks. Once new interactions begin between i and j, then according to premise (1) above, $\beta_{ij\leftarrow k} = 0$. This renders any good-mouthing or ballot-stuffing attacks by k useless. Together with the trust decay function, it also provides an effective defence against opportunistic or on-off attacks by k. This is so because i does not accept recommendations on nodes it is currently interacting with. Also, a node performing random attacks simply degrades the partial trust score it receives from i. Thus, a malicious node stands little chance to gain by providing a bad recommendation or service to i, and its ability to impact $V_{ij}(p)$ is severely limited.

3.1.6 Trust Update and Maturity

Let $D_{ij}(p)$ be i's direct assessment of j on trust parameter p in the current session of interactions. The method of assessing $D_{ij}(p)$ depends on the parameter. It could be a rate of work done, which can be computed simply based on observation. It could also be a co-location score, for which a formula needs to be applied. Generally, the method for evaluating direct assessments must be specified for each parameter when designing the collaboration context.

To update trust reliably, the concept of trust maturity must now be introduced. In addition to direct assessments, previous trust scores and recommendations impact the current value of any partial trust score, and we have described trust decay and recommendation belief functions for these. There should be a point in time at which direct assessments are sufficient to compute trust scores. Let Γ be the number of interactions required to reliably measure $V_{ij}(p)$ based on $D_{ij}(p)$ only. After Γ interactions between two nodes in any session, the trust score computed by one on the other attains maturity or equilibrium. In other words, trust maturity is a state that is attained in a collaborative session when direct assessments of the interactions between any two nodes are sufficient for either node to accurately assess the other's trust scores. At this point, past trust between the nodes is assumed to have decayed completely and recommendations are not considered in computing the trust scores of either node.

The value of Γ depends on the collaboration context, C, and must be determined by initial experiments. Once this value has been determined, it can be used in future sessions to weight previous trust scores. For example, after Z interactions between i and j in a new session, the effective proportion of a previous trust score, $\left(V_{ij}(p)\right)_0$, is given by:

$$\mu_i = \max\left(\left(1 - \frac{Z}{\Gamma}\right) \times (\phi_i)_t, 0\right) \qquad (3.12)$$

In other words, once $Z \geq \Gamma, \mu_i = 0$ in accordance with Assumption (4) in section 3.1.4. At the start of a new session of interactions between i and j, the initial value of $D_{ij}(p) = 0.5$. This is the midpoint between complete distrust (0) and perfect trust (1). It is taken as the neutral trust value in the absence of any information. It is also the default assessment value that is used for computing trust scores for a node with which i has had no previous interactions.

We have now discussed all the three factors needed to compute $V_{ij}(p)$ at current time, $\left(V_{ij}(p)\right)_t$. Let $G \subseteq C$ be the set of nodes currently in a collaboration session with node i. Suppose there are s nodes in G, $G(1) \ldots G(s)$, that have a recommendation on j, then:

$$\left(V_{ij}(p)\right)_t = \begin{cases} \frac{D_{ij}(p)+\left(\phi_i\times\left(V_{ij}(p)\right)_0\right)+\sum_{x=1}^{S}\Psi_{ij\leftarrow G(x)}(p)}{1+\phi_i+\sum_{x=1}^{S}\beta_{ij\leftarrow G(x)}(p)} & Z=0 \\ \frac{D_{ij}(p)+\left(\mu_i\times\left(V_{ij}(p)\right)_0\right)}{1+\mu_i} & Z>0 \end{cases},$$

$$\forall p \in P, G(x) \in G, j \notin G \tag{3.13}$$

Table 3.1 List of CTRUST Model Properties

Symbol	Description	Type
T_{ij}	Trust value of j as computed by i, at the current instance and context	Derived
p	A trust metric or parameter by which trust is assessed in the current context	Design
$w_i(p)$	The importance of p as determined by i	Input
$D_{ij}(p)$	The direct assessment score of j, as measured or perceived by i, on parameter p	Derived
$\left(V_{ij}(p)\right)_t$	The trust score of j, as determined by i, on parameter p, at time t	Derived
$\left(V_{ij}(p)\right)_0$	The trust score of j, as determined by i, on parameter p, at the end of the last session	Derived
$(\phi_i)_t$	weight of $\left(V_{ij}(p)\right)_0$ in next session of interactions after time t of no interactions between i and j	Input
$\left(V_{ij}(p)\right)_0$	The real value of $\left(V_{ij}(p)\right)_0$ that determines $\left(V_{ij}(p)\right)_t$ at time t, in the current session	Derived
μ_i	Best defined as $\left(V_{ij}(p)\right)_{0\rightarrow t}$ / $\left(V_{ij}(p)\right)_0$	Derived
$\beta_{ij\leftarrow k}$	The proportion of a recommendation on j, from k, that i is willing to accept	Derived
$\Psi_{ij\leftarrow k}(p)$	The indirect assessment score of j on parameter p, as received by i from k, based on $\beta_{ij\leftarrow k}$	Derived
Γ	Number of interactions in a session required to reliably measure trust by direct assessment only	Design
N[C]	Number of all nodes in the collaboration context and community, C	Input
N[G]	Number of nodes in C that are actively in collaboration with i at the current instance	Input

Every property required to setup the CTRUST model has now been defined. A brief description of each property is given in Table 3.1 for easy reference. Fig. 3.1, as noted earlier, is a basic illustration of the trust computation process in CTRUST which has been discussed in the preceding subsections.

It has been shown that CTRUST supports multiple QoS and social parameters. While CTRUST allows the aggregation function to be specified according to the context, a dynamic weighted sum is used in this case. Also, trust recommendations in CTRUST are propagated in a distributed and decentralised manner. It has also been shown that the trust update mode is event-based; i.e. trust scores are updated after every interaction. In [43] and [109], trust computation models are classified according to their trust composition, propagation, aggregation, update and formation. Using that classification scheme and notation, CTRUST is a QoS + Social / Distributed / Dynamic weighted sum / Event / Multi-Trust with dynamic weighted sum. The performance of the model is evaluated in the next section.

3.2 Model Performance and Evaluation

For evaluation, the trust model was implemented in a collaborative download application. The aim here is to measure the performance of the model in a real-word collaborative context, in terms of the trust properties of trust accuracy, convergence and resilience. In turn, this will prove the effectiveness of the trust composition, persistence, decay, and risk mitigation methods applied in the model, as discussed in the previous section. The trust properties of platform consideration (IoT) and trust as a decision (TaaD) are implicit in the model's design. Thus, it shall be proven that CTRUST satisfies all the trust properties of a suitable TMS for the IoT as enumerated in Section 2.3.

In the following subsection, the experimental collaboration setup is introduced and explained. In Section 3.2.2, the trust parameters are defined, and initialization values are provided for other model parameters as required. The actual evaluation is discussed in Sections 3.2.3-3.2.5.

3.2.1 Context Overview: Collaborative Downloading

The concept of collaborative downloading (CD) has been addressed in previous works [113]–[115]. Collaborative downloading is a peer-to-peer (P2P) paradigm where the bandwidth of multiple devices is pooled to download a resource. Usually, a peer requiring a resource requests that the other collaborating peers assist to download the resource using their Internet connection and bandwidth, as illustrated in Fig. 3.2. This is especially useful where the peers or nodes are nomadic, and individual mobile bandwidth is small relative to the resource to be downloaded. One scenario is where the content to be downloaded is commonly requested by the collaborating peers. Rather than downloading the resource individually, they can collaborate so that each peer downloads partitioned data ranges of the resource. These partitions can then be aggregated and delivered to each peer.

Another scenario is in places where there is limited or no Wi-Fi, ADSL or other broadband, and the only available Internet connectivity is the more expensive and/or slower mobile 2/3/4 G network. Peers may collaborate to save money and time in such cases. Access to a resource server or WSN sink is also optimised using this technique. Rather than making multiple connections to the same server (or sink in WSN) for the same data ranges, each peer downloads a different data range at a time, thus optimising the server or sink uplink.

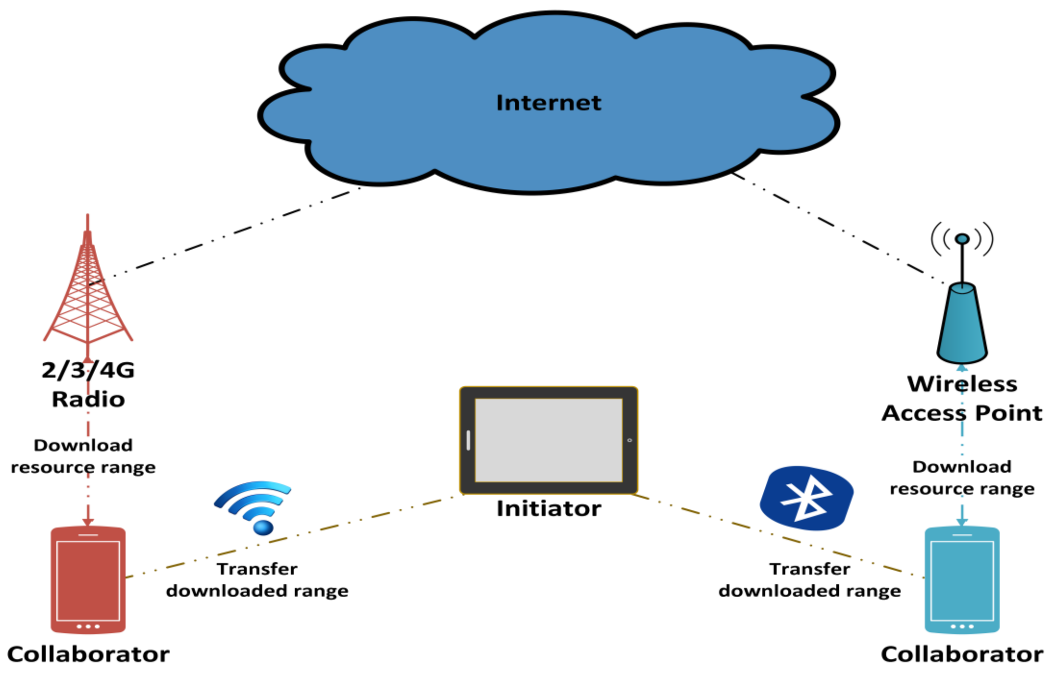


Fig. 3.2. Illustration of a collaborative download session Collaborators download byte ranges of the requested resource and transfer back to the initiator using a WLAN connection.

In a CD system, nodes available to help with a download send out broadcasts of their availability. These broadcasts are seen by all other nodes in the same geographic vicinity. These nodes form the set *C*. A node wishing to initiate a download (we call this node the initiator) will pick collaborators from this set of nodes using some selection algorithm. The selected nodes form the set *G*. The initiator sends out the URL of the resource (workload) to be downloaded to these nodes. The workload is divided into blocks. Each block is a range of bytes of the workload. The blocks are distributed among each node in *G* using some work schedule algorithm. The nodes transfer the completed blocks by uploading the byte range downloaded as a file object to initiating device over the wireless communication channel. Therefore, the CD application could be thought of as a distributed download manager. When all the blocks have been downloaded, the file objects are combined to retrieve the original resource.

3.2.2 Collaboration Context Setup

We identified three criteria to judge the suitability of a node as a collaborator in the CD context described above: its download speed, reliability in successfully completing workloads and the level of security risk it poses to the collaboration. Based on these criteria, we now define the following three trust parameters to evaluate the CTRUST model in this collaboration context:

1. *Successful Completion Rate (SCR)*: This is a measure of the reliability of a collaborating node. It is based on the number of times, in the current session, that a node has successfully both downloaded and transferred a work queue block back to the initiating node. In a new session of collaboration, the direct assessment $D_{ij}(SCR)$ begins at 0.5. The initiating node keeps a cumulative count of the total number of both assigned and successful blocks in the current session.

2. *Cumulative Bandwidth Average (CBA)*: This is a measure of the work speed of the collaborating node. It is determined by the average bandwidth of a node, as measured by the initiator. It is cumulative over the current session of interactions. The initiating node keeps a running total of the total number of bytes and time taken, for each collaborating node in the current session. The CBA is then normalised. At the beginning of a new session of interactions the default value of $D_{ij}(CBA_{norm}) = 0.5$ is used.

3. *Inverse Risk Index (IRI)*: A malicious collaborating node may modify blocks before sending them to the initiator. It may even just send a block of the expected size, with all bytes set to 0 or 1. It may also try to disrupt the CD session, by ignoring the work queue order, for example. The introduction of a parameter to assess the nodes' malicious intent is required to keep would-be malicious nodes in check. Whenever some tampering or fraud is discovered, either in a block marked as complete or in the work queue, the initiator marks that block (or whatever block the malicious node is currently downloading) as a bad block and updates $D_{ij}(SCR)$ to reflect the risk that the node poses. IRI is cumulative over a session. As usual, the initial value of $D_{ij}(IRI)$ at the beginning of a new collaboration session is 0.5.

These parameters were considered by the proposer to be the most important functional parameters in a collaborative download context. For any context, a decision must be made to determine which functional requirements may serve as trust parameters. This is an initial step in setting up the collaboration context without which the trust model cannot be implemented.

We now proceed to compute $\left(V_{ij}(p)\right)_t$ as follows. If there has been no previous interaction between i and j, and there are no recommendations on j from any of the other collaborating nodes, then $\left(V_{ij}(p)\right)_t = \left(D_{ij}(p)\right)_t$. At the end of a session of interactions (which is one download session), the final computed value for $\left(V_{ij}(p)\right)_t$ becomes $\left(V_{ij}(p)\right)_0$ for the next collaboration session. Indirect assessments are handled as described in section 3.1.5. Once the initiator has set the weights for each parameter, T_{ij} can be computed as described in section 3.1.3. The results of the simulation are discussed below.

We set up the collaboration community with size, $N[C] = 10$ per session. Also, the maximum number of nodes the initiator i collaborates with at any time, $N[G] = 5$, except where it is necessary to increase the group size to illustrate a point. An example of such case is the speedup illustration in Fig. 3.3 that will be explained later. In this experiment, we assumed that the subjective utility function of the initiator is linear and that all parameters are of equal importance. Therefore, we used an equal weight for all the parameters, i.e. $w_1 = w_2 = w_3 = \frac{1}{3}$. In each simulation, there were 60 download sessions of interactions. The default trust value is 0.5. In simulating the behaviour of nodes with respect to the parameters, nodes are randomly set up such that they tend to complete anywhere between 50-100% of the blocks assigned to them. The same goes for block tampering or risk. A random average bandwidth between 0.5B to 1B is assigned to each node. The performance of the model is discussed in the following subsections.

3.2.3 Utility of the Model in Collaboration Context

The aim of the collaboration is to increase the download speed of a resource. Fig. 3.3 illustrates the speedup achieved. To understand the effect of the computing and time overhead expended on node selection, we run two different modes of CTRUST. In the first, the initiator utilises the nodes selected at the beginning of a session until the end. In the second, after a trust update of a node, the initiator decides whether to continue with that node, or to check [C] for another known node with a higher trust score. Trust update is triggered by the event of change to the status of a block; that is, whenever a block is returned to the initiator. This second mode is an adaptive mode, which we call CTRUST-A. For comparison, we run another simulation with the same group of nodes but selected randomly. In this experiment N[C] = 10, that is the maximum number of nodes available for collaboration.

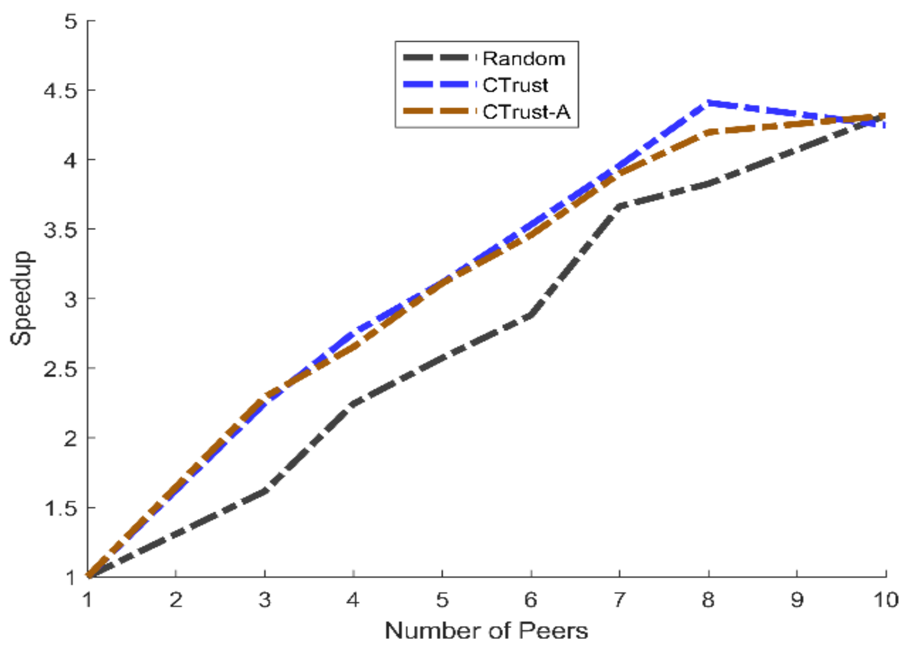


Fig. 3.3 Plot of Speed-up against different group sizes, N[G]
This shows the download speed-up achieved using either modes of CTRUST for node selection compared to a random selection of nodes.

Table 3.2 Two-Sample T-Test Comparing Session Speeds Obtained Using CTRUST and Random Modes for Node Selection

	Random vs. CTRUST	Random vs. CTRUST-A	CTRUST vs. CTRUST-A
Observations	60	60	60
P value at (α=0.05)	1.27E-17	3.77E-23	0.0516 (lowest value obtained)

We observe that even when the utilization level is up to 70% of the nodes in such a small community (i.e. N[G] = 7), there is still a significant improvement in the speedup achieved when nodes are selected based on the trust model as opposed to randomly. Beyond this point, i.e. if N[C]/N[G] > 0.7, then the initiator has limited ability to discriminate based on its preferences, since it can reject only 30% or less of the available nodes regardless of their trust scores. As a result, the impact of the trust model on speedup rapidly decreases. When N[G] =N[C], no selection takes place since all the nodes in the collaboration community are being utilised for downloading. Thus, there is no difference in speedup when N[G] = 10, as can be seen in Fig. 3.3.

The speedup achieved is comparable to the results that were obtained in [113], with the added advantage of trust. We observe that even with the extra computation involved, CTRUST and CTRUST-A outperform the random selection in speedup. A two-tailed test also shows a significant difference in the overall average speed obtained per session between random selection and CTRUST, as shown in Table 3.2. Hence, we conclude that incorporating the trust model into the CD protocol does not negatively impact on the performance of the protocol. The difference in speedup over the course of a session between the two modes of C-Trust is statistically insignificant. However, CTRUST-A computes fresh trust scores after each interaction. Therefore, this mode is more sensitive and adaptive to changes in node behaviour within a session. This adaptability, known as trust resilience, is a desired property and is evaluated below. For this reason, CTRUST-A is the default mode used in our experiments.

3.2.4 Evaluating Trust Model Accuracy and Convergence

We now compare the trust scores obtained by the model to the ground truth status, and how long it takes to converge to ground truth status. The ground truth status is obtained by computing what the trust score should be based on the randomly assigned nodal characteristics. It is the truth value that would be assigned to the node if the trustor had perfect knowledge of its behaviour. This comparison is important because it shows the effectiveness of the model in accurately estimating trustworthiness of nodes in a reasonable time. The results obtained are presented in Fig. 3.4-3.6.

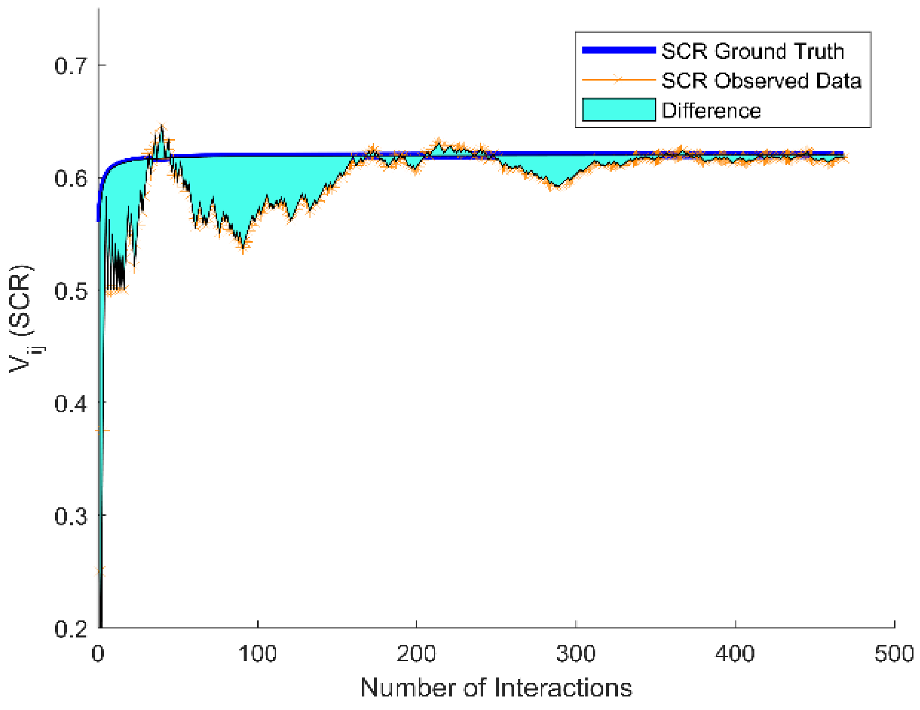


Fig. 3.4 Convergence of SCR to ground truth.

The results show that the trust value of the node being assessed converges to the ground truth status after about 250-300 interactions. This number, though seemingly high, is to be expected. This collaboration context requires a short timeframe for each interaction. Therefore, a relatively high number of interactions would be required to accurately compute trust values. Also, the fluctuations that can be observed are to be expected because the behaviour of a node may be *perceived* differently due to environmental or external factors. However, it should be noted that if the node characteristic remains the same, the trust value will converge back to the ground truth.

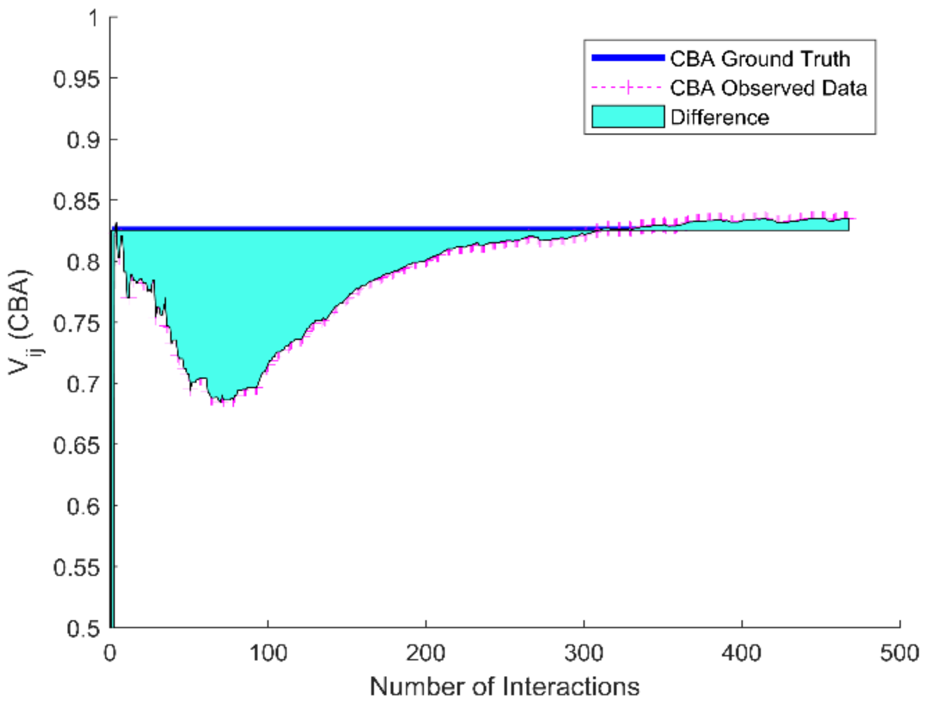


Fig. 3.5 Convergence of CBA to ground truth.

The large dip in Fig. 3.6 after initial convergence is due to the sensitivity of the parameter (IRI) to slight changes in assessment. This is a property of this collaboration context. Unlike in [35], where the trust value is tracked over 100 hours, the results here show the trust value over the interactions in one session. This is more logical in our opinion, as trust scores are a function of interactions over time. For comparison, however, the usual duration of one session is about 2 hours.

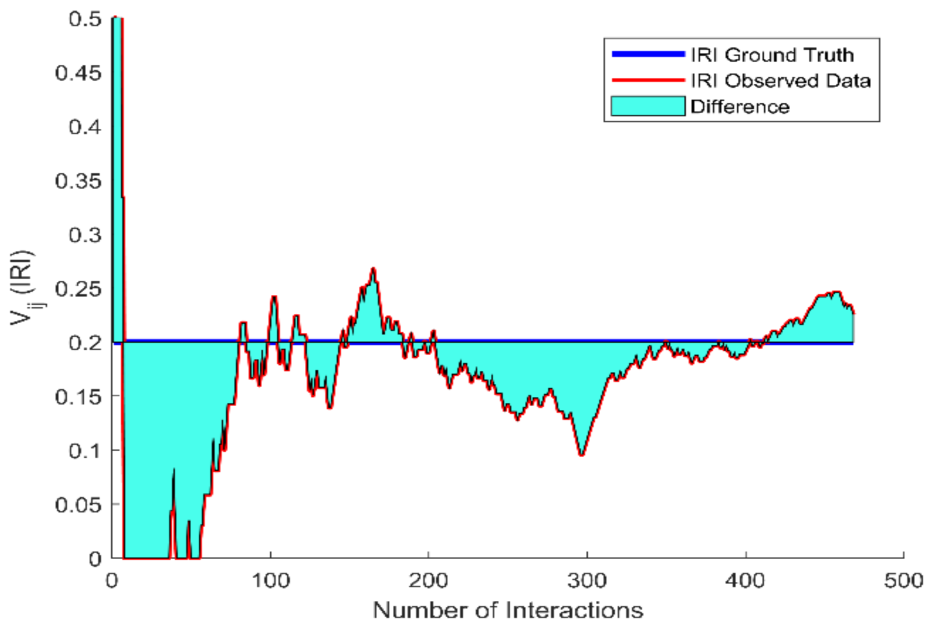


Fig. 3.6 Convergence of IRI to ground truth

3.2.5 Evaluating Trust Model Resilience

Resilience is the ability of the model to adapt to changes in the collaboration community and maintain a high efficiency under such circumstances. The three major factors affecting this are bad recommendations from other nodes, change in node behaviour, and increase in the proportion of malicious nodes in the collaboration community. By design, bad recommendations have negligible effect on CTRUST, as explained in Section 3.1.5. Recommendations are only used at the beginning of a new session of interactions with a node. A bad recommendation will show only as a minor initial fluctuation on the graph. The adaptive mode of CTRUST ensures this. Therefore, this is not discussed further.

CTRUST adapts to change in a node's behaviour both within a session and between sessions. The former has been illustrated in the previous subsection. The results show that any nodes can reliably assess one another without the necessity of recommendations after about 250 direct interactions between them. Thus, trust maturity is reached after 250 interactions, i.e. $\Gamma = 250$. Fig. 3.7-3.9 illustrate the resilience of CTRUST to changes in node behaviour across two sessions, using the same node as in the previous simulations. The second session begins with the 470th interaction, at which point $\phi_i = 0.8$. For comparison, there are two plots for each parameter; one with the trust decay function described in section 3.1.4, and the other without it.

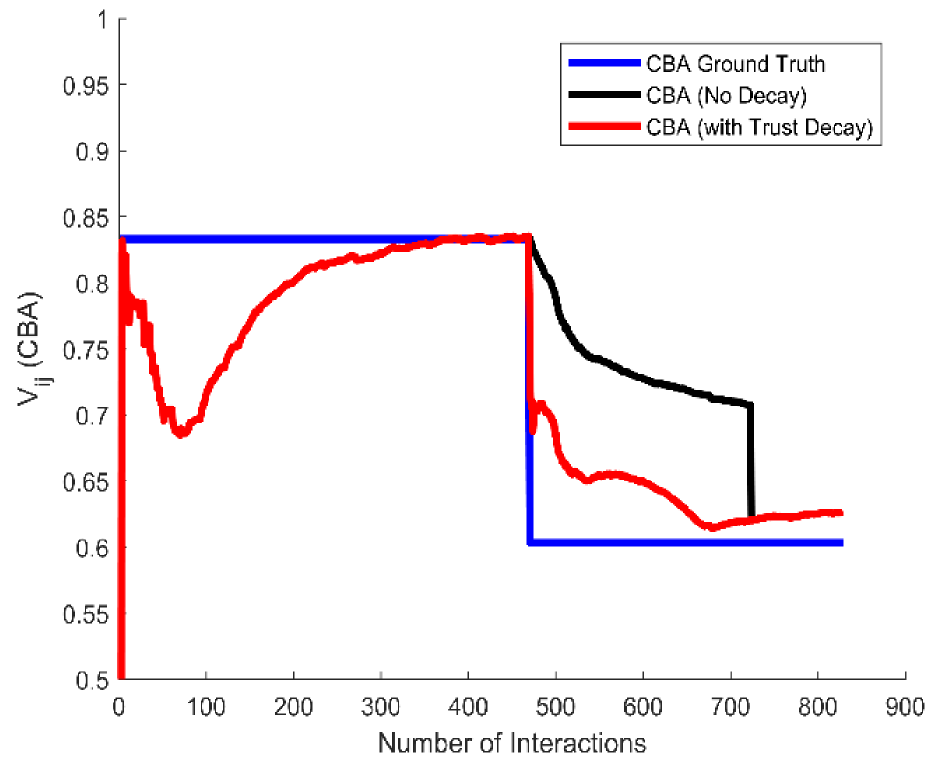


Fig. 3.7 Resilience of CTRUST to change in CBA

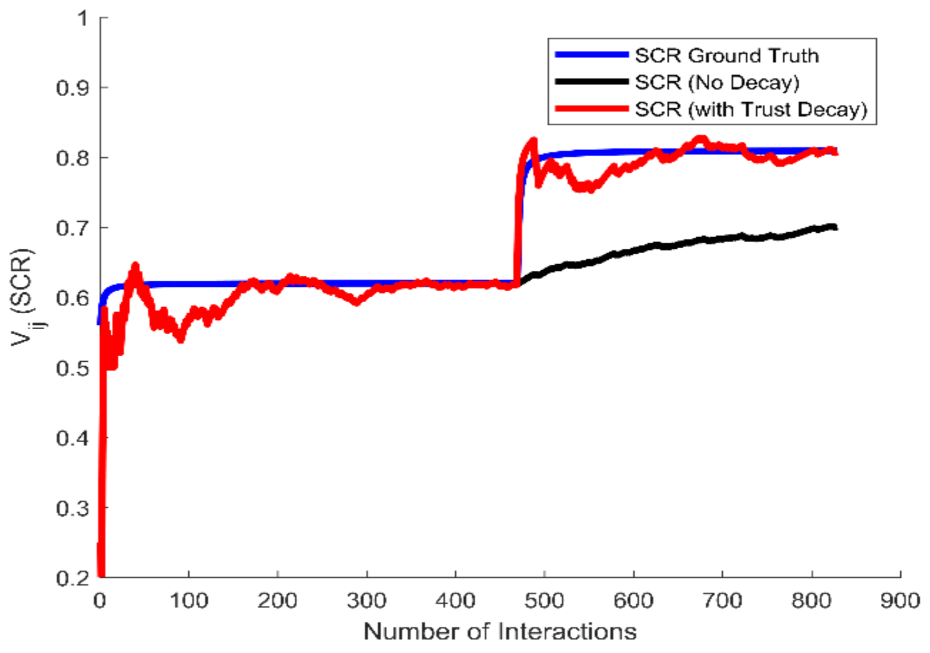


Fig. 3.8 Resilience of CTRUST to change in SCR

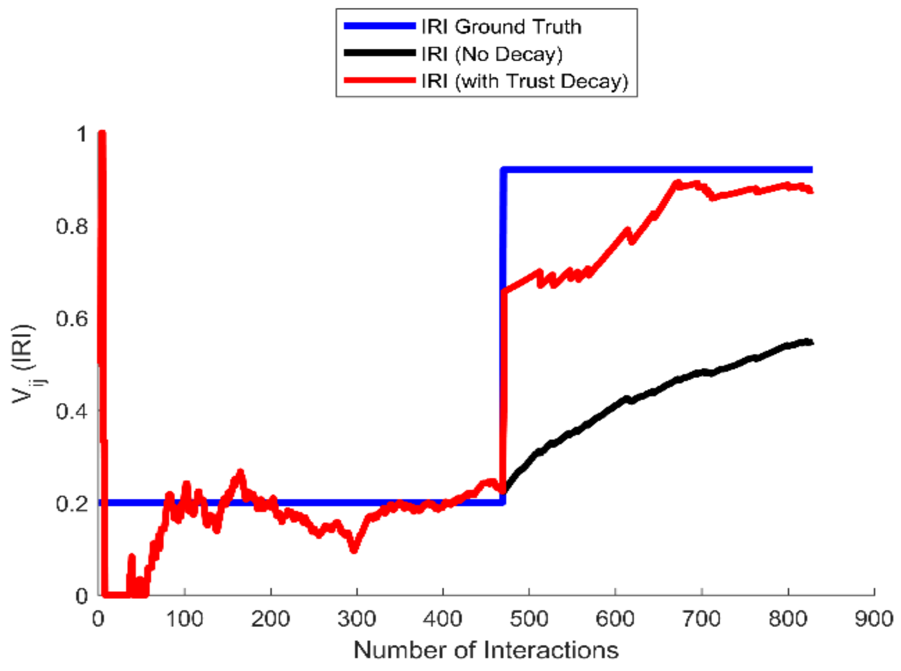


Fig. 3.9 Resilience of CTRUST to change in IRI

The results show that without the trust decay function, it takes much longer to start to converge, and it may never reach the new ground truth status. With the trust decay function however, the new ground truth is reached after about 250 interactions, thus keeping the trust maturity index, Γ, constant in the collaboration context. This proves the efficacy of the trust decay function.

In Fig. 3.7, an early dip in the curve is noticeable. This is due to the nature of the parameter being assessed and the behaviour of the node being assessed. CBA is a cumulative measure of reliability. Therefore, if a node achieves little in initially completing workloads, then it requires a steeper curve to reach ground truth status afterwards. The ground truth value for CBA, in this case, was 0.82. The assessed value dropped to 0.68 due to some failed blocks in the simulation. The dynamic update method in CTRUST-A implies that trust scores are updated after every interaction. This accounts for the variations that can be seen in Fig. 3.7-3.9. In the non-adaptive mode of CTRUST, the variations would balance out over the course of a session of interactions. The dynamic mode is preferred, however, because changes in the behaviour of collaborating nodes are continuously tracked and this makes it easier to spot malicious behaviour or a sudden drop in service level.

The initiator can define a minimum expected level of speedup to address the problem of suspected increase in the proportion of malicious nodes in the community, especially when the derived utility (speedup) suddenly drops significantly. The initiator can also alter the relative weights of the parameters to achieve its desired service level. For instance, if there are a lot of failed blocks, the weight of SCR may be increased by 20%. The same rule applies to every other parameter. If after two iterations the service level is not met, then the initiator may never achieve its minimum utility. The initiator should terminate the collaboration at this point because its objective cannot be achieved.

Having shown that CTRUST meets the required criteria for an ideal trust model for collaborative applications in the IoT, as outlined in Section 2.3, the model is compared to related work in the next section.

3.2.6 Comparison to Related Work

A survey of trust models in the existing literature has already been reported in Chapter 2. In this section, CTRUST is compared to the similar models for trust management for IoT applications. This is done to show the importance and distinction of the work, and the contributions it makes to this field of study.

In Section 2.1, the limitations of existing trust and reputation models have been discussed, particularly with respect to an overdependence on recommendations – which are usually utilised unmodified - and the lack of a trust decay function to account for the temporal degradation of trust values. In CTRUST, however, there are separate recommendation and trust decay functions, each of which takes the temporal nature of trust into consideration. The recommendation function aggregates multiple recommendations received around the same time on the same node into a group recommendation score. Finally, we combine weighted values of direct assessments, past trust scores and group recommendations to compute present trust scores. This ensures that trust scores are weighted once, and it mitigates the risks posed by a ring of bad recommenders.

Chapter 2 discussed the risks inherent in using trust models that are solely based on reputation for IoT, such as [35], [44], [45]. CTRUST, on the other hand, utilises both objective and subjective trust parameters in computing trust values for nodes. The trustor also decides the relative importance of each parameter. Hence, rather than having to use two different trust models in an IoT application context where both QoS and social trust must be considered, our trust model ensures we can compute one aggregate trust value in such contexts. Our model also dynamically adapts to changes in the trust community similar to [15], [32]. Both models correctly state that bad recommendations make it difficult to reach a new ground truth status quickly, as was also discussed in this chapter. In those models, however, two different model parameters must be tuned to achieve a high degree of trust resilience. The method with which we manage recommendations in our model ensures that we achieve a similar level of trust resilience without having to continuously tune system parameters during interactions.

In most of the trust models cited in Section 2.1, the reputation of a node providing a recommendation is the only factor that is taken into consideration in deciding the weight of that recommendation. In CTRUST, the recommendation function also considers how recently direct interactions were made between the trustor and the node on which a recommendation is being provided, and the difference between the past trust value and the recommendation value for that node. This makes recommendations more robust and effectively deters opportunistic service attacks. We also determined the point at which the trust between two nodes reaches maturity. When trust maturity is reached within a session, a node can reliably compute trust values based on direct assessments alone. This reduces both the processing and storage overhead involved in trust computation, which is vital in the IoT platform where low computing power is a major characteristic.

3.3 Chapter Summary

CTRUST models trust as a result-oriented metric evaluated on parameters that are mapped to the functional requirements in the applied context. The model was evaluated in a collaborative download context. The analysis shows that the model is effective, and its trust estimation and performance show a high degree of accuracy, reliability, and resilience. The model is adaptable to several collaborative contexts. CTRUST effectively addresses self-promotion, good-mouthing, ballot-stuffing, opportunistic service, and on-off attacks. It requires little computing and energy resources for trust computation.

CTRUST is flexible since several design parameters can be set up based on the applied context. We implemented a robust trust decay function and mathematically modelled the acceptance of recommendations based on insights from social interactions. We were also able to determine the number of direct interactions required to achieve trust maturity between any two nodes. From the literature available to us, we conclude these are novel and important contributions to the study of trust management in the IoT. It should be noted that while the importance of trust-based security in the IoT has been recognised, trust management for IoT is still evolving. The proposed model, CTRUST, is a major contribution to this field, fills important research gaps, and will provide a better well-rounded approach to trust modelling in the IoT. This work has been published in the IEEE Internet of things Journal [67]. In the next chapter, CTRUST is extended to design a new model for trust evaluation and management in service compositions.

CHAPTER 4 SC-TRUST: A DYNAMIC MODEL FOR TRUSTWORTHY SERVICE COMPOSITION IN THE INTERNET OF THINGS

In the previous chapter, CTRUST was proposed as an ideal trust model for use in the IoT. However, the work focused only on collaborative applications where the trustor (i.e. SR) directly requests and receives services from SPs and can, therefore, provide a direct trust rating of the nodes based on their performance, thus establishing a trustor-trustee relationship. Even though service compositions are a special category of collaborative applications, there is no direct trustor-trustee relationship, meaning that the SR is unlikely to be aware of the identities or trust scores of the SPs given that the detail of the composition is abstracted from the SR. The general properties of an ideal trust model for the IoT, as enumerated by Requirements (1-9) in Section 2.3, are satisfied by CTRUST. However, no previous trust models meet Requirements (10-11) listed in the same section. Thus, the model described hereafter encapsulates and significantly extends the CTRUST model by fulfilling the latter requirements, with a focus on dynamic trust management for service compositions in the IoT.

4.1 Model Design and Analysis

Based on the study discussed in Sections 2.2-2.3, SC-TRUST is proposed as a suitable model to guide compositions of IoT services. Underlying services and SPs are assumed to have trust scores assigned to them prior to the composition, based on their direct interactions with other nodes or peers outside a service composition context. Just as in CTRUST, a trust score is computed based on the objective measurement of a node's performance on functional parameters, which form the basis on which the trust assessment is made. Trust criteria are modelled based on these parameters; a trustor determines the weights of each criterion. Trust scores are stored and are used to guide future interactions, although their importance declines over time. SC-TRUST extends the trust assessment, decay recommendation and aggregation functions of CTRUST to include two novel functions for transparent trust composition and decomposition. A high-level workflow of the model is illustrated in Fig. 4.1. A stepwise overview of the service composition process can be thus described:

1. Underlying services are assessed on one or more trust criteria (parameters) relevant to their service class. Each trust parameter could be objective (QoS) or subjective (social) in terms of the assessment. The assessment results are called *partial trust scores*.

2. An SR requests a *request-granting service*, which is the middleware for composing the services and upon which SC-TRUST is built. Further details of the middleware are abstracted; it is assumed that the middleware is impartial, agnostic and can operate in a decentralized architecture. A candidate platform would be a blockchain built for this purpose, similar to those proposed in [72]–[76].

3. The middleware composes a service workflow pattern to match the SR's requests. The details of the mechanism for this are not relevant to this discussion. It is sufficient to note that examples of candidate middleware solutions exist in the literature [16], [19]. SC-TRUST can be integrated into a suitable middleware platform.

4. After the service workflow is decided, a handover is made to SC-TRUST to guide the actual composition process. For each service class required in the composition, SC-TRUST identifies the parameters relevant to the composition, to which the SR may assign weights.

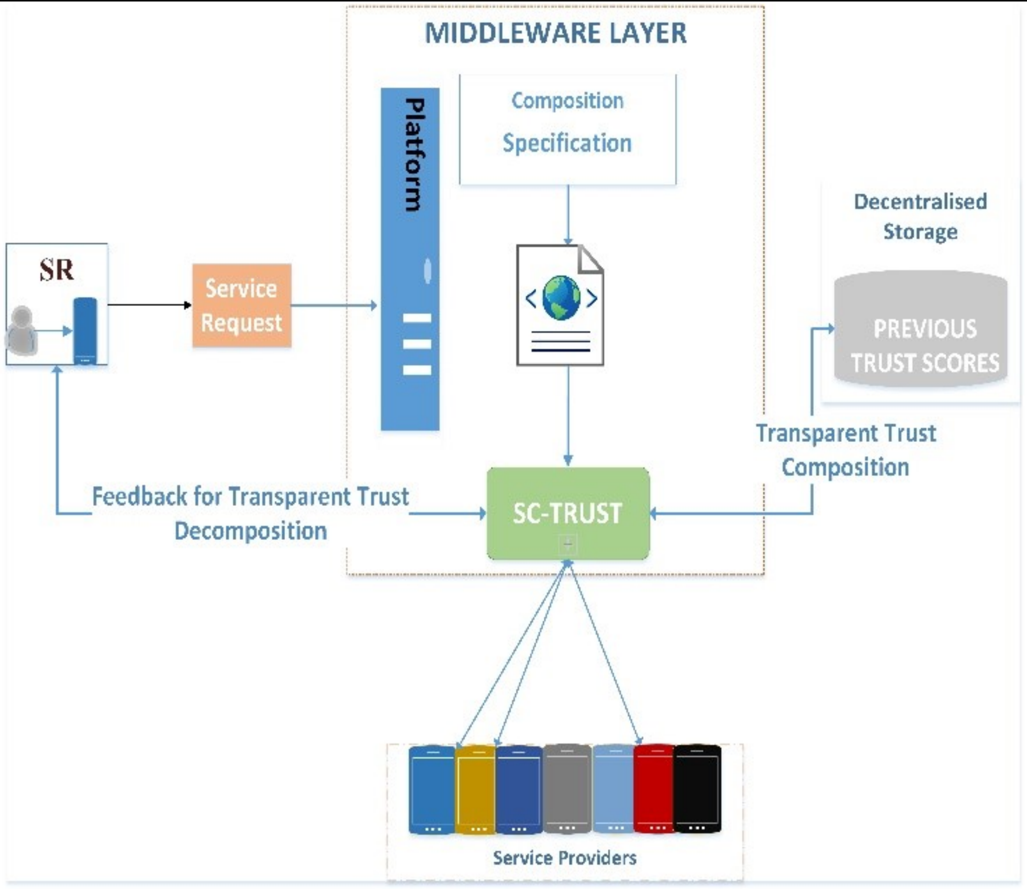


Fig. 4.1 Basic processes involved for trust computation in SC-TRUST

5. The prior partial trust scores on the parameters identified in (4) for each SP are aggregated according to the SR's weight assignment, to calculate a single trust score for each SP. The trust aggregation function in CTRUST is utilized here. The most trustworthy SP(s) in each service class is(are) selected according to this score.

6. The trust score of the composed service is estimated transparently through a trust composition function of SC-TRUST based on the workflow, relevant partial trust scores of the SPs, and weights assigned by the SR.

7. Upon service consumption, the SR may provide a score on each parameter; this is used to transparently update the posterior partial trust scores of the SPs through a trust decomposition function of SC-TRUST.

8. The trust decay and recommendation functions of CTRUST are inherited and utilized as required.

Let ***S*** be the set containing the SR and all the SPs available for composition under the model. ***T[S]***, the trust space over *S*, is an octuple expressed by the following notation:

$$T[S] \equiv \left[T_{ij}, P, W_i, V_{ij}, F, t_{\frac{1}{2}}(i), C, D\right] \forall i, j \in S \qquad (4.1)$$

Where

- T_{ij} is the trust score of SP j from the perspective of SR i;

- $P = \{p_i, p_2, p_3, \dots, p_n\}$ is the set of all trust parameters or properties from every service class in the composition;

- $W_i = \{w_i(p_1), w_i(p_2), w_i(p_3), \dots, w_i(p_n)\}$ is the set of weights on each parameter in *P,* as assigned by SR *i;*

- $V_{ij} = \{V_{ij}(p_1), V_{ij}(p_2), V_{ij}(p_3), \dots, V_{ij}(p_n)\}$ is the set of values denoting SR *i's* perceived assessment (partial trust score) of SP j on each parameter in *P*;

- $F = f(W, V) \equiv T_{ij}$ is the trust aggregation function;

- $t_{\frac{1}{2}}(i)$ is the half-life of any partial trust score computed for *i*, that is, the time required for a partial trust score to decay to half of its original value;

- *C* is the trust composition function; and

- *D* is the trust decomposition function.

The CTRUST functions described in Section 3.1 apply to SC-TRUST; therefore, the discussion in this chapter focuses on the extensions provided by the model, specifically transparent trust composition and decomposition, which are described in the subsequent subsections.

4.1.1 Transparent Trust Composition

It is necessary to provide an SR with a reliably estimated prior trust score of the composed service requested, to assist the SR in making an informed decision to accept or reject the service offer. The goal is to provide the SR a guarantee of the minimum level of satisfaction derived from the composed service. As such, this is not a trust judgement on the SPs themselves; rather, it is an indication of the likelihood that the composed service meets the SR's requests. Upon consumption, the SR's posterior feedback may indicate a higher or lower level of satisfaction compared to the prior estimated score. A reliable prior trust estimate should not be higher than the posterior feedback from the SR. Therefore, a novel, bottom-up approach is proposed to compose the trust value. The objective here is to *satisfice* [116], that is to initially find a suitable composition which meets the SR's threshold of acceptability, even if it is not the best possible composition. This is an optimal strategy because it reduces the time, energy and computational costs involved in composing a service. Then, based on the feedback from the SR, we can update the trust scores of the SPs and compose a better service with every iteration. Thus, this strategy leads to an optimal service composition eventually.

The set of prior partial trust scores of each selected SP j is denoted now as V_j, and the set P contains all parameters for all service classes represented in the composition. These sets may be stored in a decentralised architecture, such as in a blockchain [74], [77], [78], where similar composition platforms may access these values. The details of the storage and retrieval mechanisms are not relevant to the model design at hand. The SR assigns weights to each of these parameters to indicate their order of relative importance. This could be implemented in several ways. One way would be to assign a default weighting of 0.5 on a scale of 0 to 1.0 for each parameter. Then the SR can adjust the weighting of any parameter as desired. Another method could be to request that the SR provide pairwise comparisons or ratios on matched parameters (e.g. taxi fare vs cleanliness, cleanliness vs vehicle emissions). Analytic Hierarchy Process (AHP) methods can then be used to elicit consistent weights for each parameter. Once the weights have been determined, we can proceed to estimate the trust score of the composed service recursively, as determined by the workflow. Each workflow is resolved to an equivalent single service characterized by an appropriate set of partial trust scores. The method by which this is done for each workflow type is discussed in the following subsections.

4.1.1.1 Selection Workflow

In a selection workflow, the SR must select one SP from a group of two or more SPs that are from the same or associated service classes and offer similar services. Therefore, we know that this group can be represented by the SP with that highest trust score, as this is the SP most likely to be chosen by the SR. Once an SP is chosen, the services of the others are not used (not at this level, at least) and therefore do not impact on the composite trust score. Therefore, to resolve or simplify this workflow, we determine the SP with the maximum prior trust score based on the partial trust values on the same set of parameters, using weights assigned by the

SR. Let Q be the set of SPs from which a selection is to be made, and $R \subseteq P$ be the set of parameters common to every member of Q. We compute:

$$T_j = \sum_{p_x \in R} w(p_x) \times V_j(p_x), \ \forall j \in Q \qquad (4.2)$$

Therefore, the set Q can be resolved to a single logical service that has the equivalent trust characteristics (the same set of partial trust scores, V) as the SP with the highest trust score as computed by the above equation. Two or SPs may tie for the highest trust score. This does not affect the resolution of the services, because the trust score represents the estimated maximum utility that the SR may derive from the consumption of any of such services. Unless there are other non-functional constraints, we can choose the partial trust score set of any of these SPs for the equivalent logical service. This set is represented mathematically by:

$$V \overset{\text{def}}{=} V_j, \text{where } T_j = max(\{T_l\}_{l \in Q}) \qquad (4.3)$$

4.1.1.2 Parallel Workflow

In a parallel workflow, multiple SPs provide services concurrently. The SPs may be of the same or different service classes. Consider the case of SPs of the same class (and therefore offering the same service). An example of this would be a collaborative download session. If two SPs have a score of 0.7 and 0.9 respectively on a trust parameter based on bandwidth, then the average bandwidth would be 0.8. In this case, the collective services provided by this group of SPs could be reliably substituted by a single service with a partial trust score equal to the average group score on each trust parameter.

Suppose that the SPs are from different service classes. For example, an SR could request readings from a relative humidity SP and a temperature

SP simultaneously. In this case, the SPs could be logically replaced by a single service that offers both services. This single service has a set of partial trust scores that is given by the union of the sets of partial trust scores of all the SPs. Where two or more SPs have a partial trust score on the same trust parameter, the replacement service is assigned the average (arithmetic mean) value of the scores on this parameter. This is true for both cases above.

To formalise the above cases, let Q be the set of SPs in the parallel workflow and $R \subseteq P$ be the set of unique parameters for all the SPs in Q. Then the set V for a single service to replace Q is given by:

$$\begin{aligned} V &\overset{\text{def}}{=} \left\{ \frac{\sum_{j \in Q \,\wedge\, o_j(p_x)=1} V_j(p_x)}{\sum_{j \in Q} o_j(p_x)} \right\}_{p_x \in R} \\ o_j(p_x) &= \begin{cases} 1, \text{if } V_j(p_x) \in V_j \\ 0, otherwise \end{cases}, \forall j \in Q, p_x \in R \end{aligned} \tag{4.4}$$

Here, $o_j(p_x)$ is used to decide whether partial trust score $V_j(p_x)$ is in SP j's set V_j. Every value in V is the average partial trust score computed separately on each parameter p_x in R from all the partial trust scores on p_x collated from the SPs in Q.

4.1.1.3 Sequential Workflow

Every service composition can eventually resolve to a sequential workflow. To understand how two microservices composed in sequence may affect the prior trust estimation of the composed service, we study two examples. First, suppose that we have a food delivery service composed of a restaurant service class and a driver/delivery service class. This is a sequential workflow. Assume that the restaurant service class has two parameters, food quality and eat-in/indoor quality with a weight of 0.65 and 0.35 respectively assigned by the SR in previous direct interactions. The driver/delivery class has two parameters: politeness and timeliness, with an equally assigned weight of 0.5 each. Suppose that the SR requests a pizza; the pizzeria (SP) with the highest trust ratings is Alice Pizzas with a food quality score of 0.9 and an eat-in score of 0.5. Therefore, its individual trust score would be *0.65*0.9 + 0.35*0.5 = 0.76*. Similarly, the trust score for a selected driver with scores of 0.4 and 0.9 on politeness and timeliness respectively would be *0.5*0.4 + 0.5*0.9 = 0.65*. Note that both the weights and trust scores are normalized in the interval [0,1]. However, in delivering food, the politeness score of the driver is not relevant as the SR does not interact with the driver. A similar reasoning applies to the eat-in parameter. Therefore, the composed service can be scored on two parameters: food quality and timeliness, to which the SR now assigns weights of 0.6 and 0.4 respectively (the other parameters are assigned weights of 0). Therefore, the estimated trust rating of the composed service at this level would be:

$$0.6*0.9 + 0.4*0.9 = 0.9$$

The second example is somewhat more complex. Suppose that the SR is visiting a new city and does not speak or understand the language of this city. The SR is interested in a museum and wants a summary of Internet articles about the museum translated to the SR's native language. Two microservices are required for the composed service: a summarization service assumed to have accuracy and response time parameters; then a translation service with an accuracy parameter. Suppose that the selected summarization SP has prior partial trust scores of 0.7 and 0.9 on accuracy and response time, respectively. The translation service has an accuracy of 0.8. In this case, the overall accuracy of the composed service cannot be reliably estimated to be greater than 0.7 prior to feedback from the SR.

The accuracy parameter is a shared parameter that simultaneously belongs to two or more service classes in the composition. If the SR-assigned weights are 0.7 and 0.3 for accuracy and response time respectively, then the prior trust score of the composed service is computed as:

*Min. (0.7, 0.8) * 0.7 + 0.9*0.3 = 0.76*

Therefore, we can now define the trust characteristics for a single logical service that is reasonably equivalent to two or more services in a sequential workflow. Let Q be the set of all SPs in a sequential workflow and $R \subseteq P$ be the set of unique parameters for all the SPs in Q. Then the set V of the single service is the union of all distinct elements which have a non-zero weighting, taken from the sets of partial trust scores of all the SPs in Q. Where two or more SPs each have a partial trust score on the same (shared) parameter, the set V contains the minimum partial trust score on this parameter. The minimum partial trust score is computed separately for each parameter in R. This is represented mathematically by:

$$V \overset{\text{def}}{=} \left\{ \min \left(\left\{ V_j(p_x) \right\}_{j \in Q \,\wedge\, o_j(p_x)=1} \right) \right\}_{p_x \,\in\, R} \qquad (4.5)$$

Having discussed the methods by which each type of workflow may be resolved into and replaced by a single equivalent logical service, we can apply them recursively as required until the whole composition is replaced by an equivalent logical service that is defined by the appropriate trust characteristics in its set V. Then the estimated trust score of the composed service is equivalent to the trust score of this logical service and computed according to the following equation, where $w(p_x) = 0$ if no weight is assigned to p_x by the SR:

$$T = \sum\nolimits_{p_x \,\in\, P} w(p_x) \times V(p_x) \qquad (4.6)$$

We derive this estimation by resolving the chain of services, regardless of its complexity, to a conceptual model consisting of a single logical service that is an equivalent abstraction of the composed service. This model is the view that the SR "sees" in interacting with the composed service. The SR need not have any knowledge of the individual underlying microservices. Thus, we have successfully composed the trust transparently for the SR. Also, none of the SPs gains any knowledge of either the SR or another SP from the composition process. Therefore, the method is privacy-preserving and makes it sufficiently difficult for an adversary to bad-mouth, ballot-stuff or perform on-off attacks. The SR then gives a posterior trust evaluation after consuming the service. The method by which this posterior score is transparently decomposed to update the partial trust scores of the SPs involved is discussed in the next section.

4.1.2 Transparent Trust Decomposition

The problem of transparent trust decomposition is inherent in the trust composition model presented in the previous section; the SR consumes and gives feedback on a single logical service based on the same parameters from Equation 4.6. However, the composed service consists of several microservices. Therefore, a method is required to reliably decompose the partial trust scores given by the SR to the underlying services in a manner that is both reasonable and impartial to all the SPs involved. This should be done through a top-down approach while persisting the logical view of the composed service to the SR and without revealing any details of the underlying services. While this is a novel and complex problem in the IoT, it is somewhat similar to the problem of group assessment in education, which has been widely studied [117]–

[119]. Therefore, we analyse the composed service and adapt concepts from methods proposed to assess individual contributions to group assessments.

Some observations can be made from the studies cited above. First, group assessments can be an effective method of assessment for both the assessor (SR) and the assessee (SPs) and it is widely used in both formal and informal education. It can increase coordination, cooperation, and collaboration among the assesses while providing a straightforward way for the assessor to evaluate the performance of all the group members at once. However, it may give malicious SPs an inducement to "free-ride" because the differential contributions of SPs are not acknowledged given that everyone gets the same scores on the same parameters. Also, it may de-incentivize high performing SPs because they may be punished for the low performance of other SPs. This leads to a degradation in the quality of experience for the SR because there is no motivation for SPs to provide better services. Moreover, some SPs may refuse to offer their services to the composition platform if they reckon that the cost of the services provided is not commensurate with the returns they receive, in terms of their trust scores. The focus of the studies in the literature has been to develop techniques to maximize the stated advantages and eliminate or minimize the problems raised.

Several methods have been proposed to improve the quality and fairness of group assessments which may be classified into one of three broad categories. First, the group members may be asked to provide a peer assessment or feedback that details what was done by each member. This can be used to assign marks or augment the assessor's marks to create a distribution of scores that is fairer to the group members than a general score. This method cannot be adapted to our proposed context for several reasons. One, it does not preserve the privacy and transparency of the service composition. Secondly, because this is a type of recommendation, it creates opportunities for malicious nodes to carry out bad-mouthing and ballot stuffing attacks. Therefore, it is counterproductive as good and reliable SPs could be unrewarded or even punished despite offering quality services. Another method proposes that the group members moderate the marks given by the assessor. This also is susceptible to trust attacks previously mentioned and is not suitable for trust decomposition in the IoT.

A third method assigns both a group score and an individual score to the group members. The individual component is based on a separate piece of work produced by each group member in addition to the group work. Thus, the advantages of collaboration and group assessment may be achieved while providing an incentive for the improvement of individual performance. Similarly, the individual component punishes bad or low-performing members, thus creating a fair distribution of the marks among the group members. These characteristics make this method feasible for use in our context. Moreover, the group members are not required to provide feedback on their peers. Therefore, it is possible to maintain the privacy aspects of the composition with this method. The same assessor may evaluate and award both the group and individual components. Although this method does not result in a perfect distribution of scores, it creates a much fairer distribution that will be reasonably accepted by the SPs. Therefore, it may be used to update their trust scores posteriorly. It is also required that we do this transparently. Now, it must be determined how this method can be adapted for trust decomposition in the IoT.

Oftentimes, only one SP per service class would be required in a service composition. Also, each service class is usually differentiated by one parameter, at least. Classes may share parameters. To express this symbolically, out of *n* parameters on which trust is evaluated in a service class, at most n-1 would be shared. This creates a method to give an individual rating to an SP of the same class. Therefore, an SP j from any service class would share at most n-1 trust ratings with another SP from a different service class, and the rating for at least one parameter would uniquely apply to j. Usually, this unique parameter of the class is also the defining parameter that is likely to be regarded as the most important by most SRs. In any case, this parameter would be relevant enough to be one of the trust characteristics of the logical model of the service composition, as derived in the previous section. It is reasonable to assume that the subset of unique parameters of the logical model would carry a significant fraction of the overall weight on trust estimation. Thus, it is reasonable and appropriate to expect that these parameters would incentivize better quality of service from good nodes and significantly lower the trust scores of malicious nodes. Therefore, when the SR provides feedback on the composed service, any score given on this parameter would only affect the SP from that service class. If a distinction between the service classes is ensured, then this method provides a reliable means to assign differential trust scores to the SPs. The set of unique parameters, $U \subseteq P$, may be expressed mathematically by the following equation:

$$U \overset{\text{def}}{=} \left\{ p \,:\, p \in P \wedge \sum\nolimits_{j \in S} o_j(p) = 1 \right\} \tag{4.7}$$

Suppose that an SP adjusts its service provision such that it receives good ratings on its unique parameters but bad ratings on the others. This is an opportunistic service attack. However, the attack fails because the SP destroys its own reputation as well with no overall gains. Our model can also deter such attacks. First, the SP does not know the identities of other

SPs involved in the collaboration. Therefore, it cannot perform selective attacks. Second, the middleware platform composing the service is a "wholesale buyer" of services and interacts directly with the SP. Even though the agnostic middleware platform does not provide trust ratings on the SP, the SP cannot distinguish between a direct interaction from a conventional SR (as in the cases covered by the CTRUST model) and the interaction from the middleware (which appears as an SR to the SP). Therefore, the SP cannot selectively attack service compositions because it cannot tell whether its services are being utilized in a composition or otherwise. If any malicious behaviour from the SP is detected, the SP is removed from the service composition and receives a negative trust feedback. This harms its reputation, making it unsuitable for any kind of service provision in the future. Because SC-TRUST is built atop CTRUST, it quickly converges to the ground truth of the nodes as we show in the next section. Third, given that there are costs to service provisioning and the SP cannot gain from malicious behaviour or disrupt the composition eventually, a rational SP would be motivated to provide good services.

On a shared parameter, the post-service trust score received from an SR is decomposed according to the individual partial trust scores (given by Equation 3.13 in the previous chapter) AND the composed partial trust score on that parameter (as defined in Equation 4.5). Suppose $V(p)$ and $V^+(p)$ are the composed and post-service feedback scores, respectively, on parameter p. Then, the partial trust score of each SP sharing this parameter is updated according to the following equation:

$$V_j^+(p) = V_j(p) + \left(V^+(p) - V(p)\right) \times \frac{V_j(p)}{\sum_{j \in S \wedge o_j(p)=1} V_j(p)} \qquad (4.8)$$

Where $V_j(p)\ and\ V_j^+(p)$ are the pre-service and updated partial trust scores, respectively, for SP j on parameter p. Equation 4.8 utilizes a game theory approach that de-incentivizes freeloading. This is achieved by differentially penalizing each SP sharing the parameter if the post-service score is lower than the prior estimated score. Given that the SPs have zero knowledge of the partial trust scores of one another, a rational SP would act to avoid the penalty by ensuring that the posterior score received on the shared parameter is as high as possible. Also, given that an SP cannot readily tell whether its services are being offered directly to a node or to a composition platform, a rational SP would seek to avoid being penalized so that its partial trust scores can remain high enough to be selected to provide services in the future. This fosters cooperation and motivates each SP to provide their best possible service.

The above discussion shows that the SC-TRUST model is privacy-preserving and transparent to both the SP and the SR and can significantly reduce the ability of either peer to perform maliciously. In addition to the deterrence of the SP's misbehaviour, SC-TRUST can also deter the middleware platform from being biased or malicious. This is because good SPs who have received good ratings from other nodes or platforms would eventually decline to interact with a misbehaving middleware platform. Therefore, the quality of its composed services would be low, and SRs would no longer subscribe to its services, leading to a loss of reputation and revenue. Given that there are other such composing platforms from which SRs can request services, the provider of the platform is sufficiently motivated to remain neutral and provide trustworthy services.

Having described the SC-TRUST in detail, the next section evaluates the performance of the model.

4.2 Model Performance and Evaluation

SC-TRUST was evaluated in a collaborative download (CD) application, utilising a similar experimental setup to the one detailed in Section 3.2.2 but with a service composition approach, instead. The evaluation aims to measure the performance of the model in a practical application and is approached along two different strategies. In the first, we evaluate the impact of the model on the level of utility gained by an SR. We compare the performance of the trust-based composition to a random composition and an SR-composed composition (that is, the SR directly shops for the individual services). In the second, we evaluate three major trust properties of the model, namely trust accuracy, convergence, and resilience. We compare the results to those obtained in CTRUST. Through this we show the efficiency of the model in service composition applications, as well as the effectiveness of the model in mitigating trust-related attacks. The model is privacy-preserving by design, as we have shown in the preceding section.

Note that as pointed out in Section 2.2.4, there is a lack of existing work on trust modelling for service compositions, so there is no other suitable model for a direct comparison with SC-TRUST.

The purpose of experiments detailed below is to show that the trust scores obtained by both transparent trust composition and decomposition are not significantly different from the scores that would have been elicited if the SR was in a direct collaboration with the SPs. Therefore, we show that by utilising a robust trust model as SC-TRUST, the advantages of service composition in the IoT can be achieved while mitigating the realisation and effects of trust risks. Given that the model is based on CTRUST, it inherits the verified methods of trust persistence, decay, and parametrization. The trust properties of *platform consideration* and *Trust as a decision (TaaD)* are implicit in the model's design. We will show that the trust composition and decomposition methods provide reliable estimates of the trust values of an SP and approach the ground truth values. Overall, we prove that SC-TRUST includes all the attributes required in an ideal trust model for IoT service composition, as enumerated in Section 2.3.

The simulation environment is largely the same as that used to evaluate CTRUST in the preceding chapter except for a few changes required for the service composition context. The hardware remains the same, but the virtual environment used is Mininet, an emulator for prototyping Software Defined Networks (SDN), running in VirtualBox. An emulator, rather than a simulator, is used to better model the host and network constraints expected in the IoT networks. Mininet was chosen because it is already suited for SDN-type networks and is easily extensible for our purpose. In addition, it is open source, written in python and well-documented. This supports reproducibility of both the experimental setup and results.

The composition process is as follows: first, an SR requests a CD session, sending the URL of the file to be downloaded as input. Then the middleware composes the services as follows: SPs which offer the download services are selected through the trust management system. The download services are composed in sequence with an aggregation service offered by a different SP. The aggregation SP divides the resource into workloads or blocks, verifies the downloaded contents, and checks for errors or malicious modifications. Then, it aggregates the verified blocks into the originally requested resource and sends the complete content to the SR over the local network connection. The communication protocol is managed by the middleware layer in a way that preserves the privacy of the SPs. This setup is illustrated in Fig. 4.2.

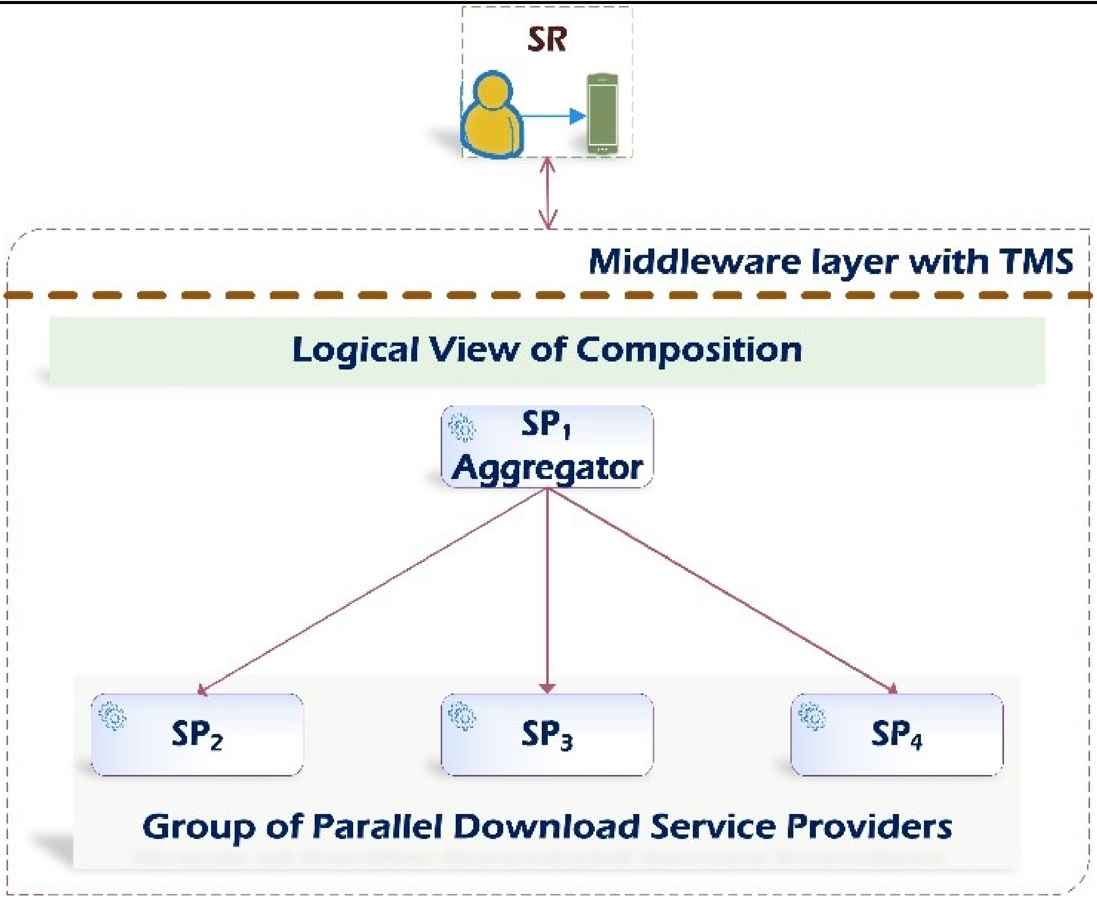

SR
Middleware layer with TMS
Logical View of Composition
SP$_1$
Aggregator
SP$_2$
SP$_3$
SP$_4$
Group of Parallel Download Service Providers

Fig. 4.2 Service composition for a collaborative download
This setup consists of one aggregation SP and three download SPs working in parallel.

In our simulation, there are 5 members of the aggregation service class, from which one is selected per CD session. There are 10 members of the download service class, from which a range of 3-5 SPs are selected for each CD session. During a CD session, the aggregation SP may not be changed, but download SPs may be removed and/or added to the composition as required. The SPs are simulated using a uniform random distribution such that their behaviour should yield a partial trust score (that is, the ground truth value) in the range [0.5,1]. Thus, there is a uniform distribution of malicious or underperforming SPs as well as good, high-performance SPs. The simulation involves 100 such download sessions.

In Section 4.2.1, the utility of the model is evaluated to determine the accuracy of the transparent trust composition in SC-TRUST. Then the trust properties (accuracy, convergence, and resilience) of the model are evaluated in Section 4.2, and they show the effectiveness of the transparent trust decomposition method utilized in the model.

4.2.1 Evaluating Utility Gain in SC-TRUST

The purpose of utilising multiple download SPs is to increase the throughput, that is the speed at which the content is retrieved and delivered to the SR. Therefore, a faster download increases the utility gained by the SR. Fig. 4.3 illustrates the relative speedup (in comparison to the SR's average download speed) achieved for different cardinalities of the set Q of download SPs. We compare the speedups achieved using: (i) SC-TRUST in a service composition, (ii) CTRUST in a collaboration context, as in [67] and (iii) a random selection of SPs in a service composition. SC-TRUST achieves a similar speedup to CTRUST for each group size. SC-TRUST performs marginally better than CTRUST initially. This is due to the logistical overhead incurred by CTRUST, because in a direct collaboration, the service requester must select the SPs and compose the service directly.

As the session progresses, CTRUST slightly overtakes SC-TRUST in terms of speedup. This is probably due to the recurring overhead involved in communicating with the middleware layer. However, the difference between both models is statistically insignificant at a significance level, $\alpha=0.05$, as shown in Table 2. Therefore, we can conclude that SC-TRUST provides as much utility gain for the SR as CTRUST, while providing the benefits of an automatic service composition. For example, using SC-TRUST, the SR does not need to request and assess SPs directly. Also, the SR does not incur energy and computation costs involved in running a service composition, as these are shifted to and borne by the middleware. Moreover, the SR does not need to keep a record of known SPs but can rely on the middleware to select appropriate SPs according to its service request.

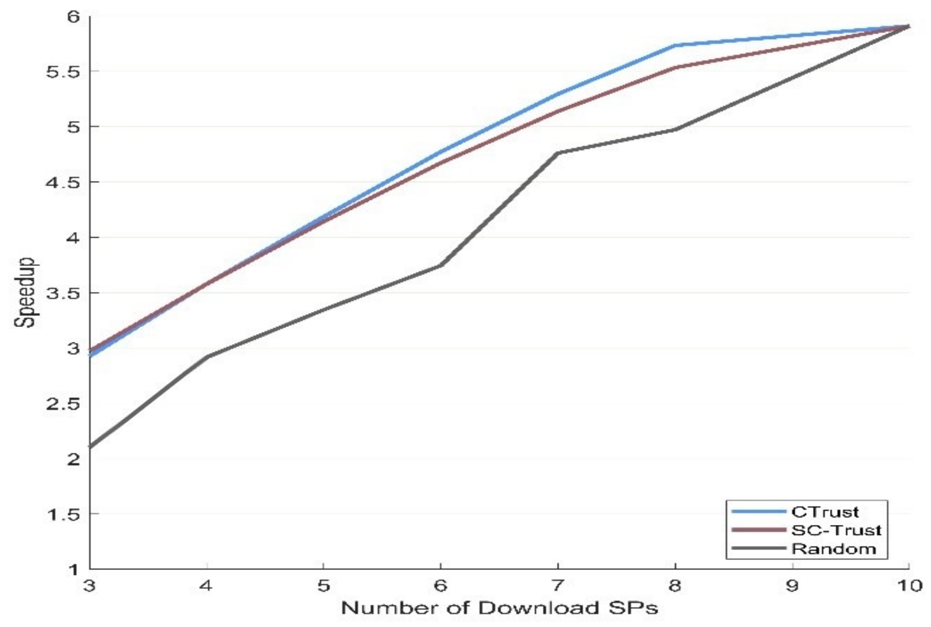


Fig. 4.3 Plot of Speed-up against varied sizes of the set Q of download SPs working in parallel
The figure shows the speed-up achieved using SC-TRUST in a service composition, compared to a random selection of SPs, or to CTRUST in a similar collaborative context.

Additionally, it is evident that the use of SC-TRUST increases the speedup quite significantly as compared to a random selection. We observe that SC-TRUST outperforms the random selection and that there is a consistent increase in the speedup even when the utilisation level of

Table 4.1 Two-Tailed Paired Sample T-Test Comparing the Speedup Obtained Using SC-TRUST, CTRUST and Random modes for Selection of SPs

	SC-TRUST vs. Random	**SC-TRUST vs. CTRUST**
Observations	100	100
P value at (α=0.05)	4.25E-14	0.053 (lowest value obtained)

available download SPs is almost 80% (i.e. the number of selected SPs in set Q is 8, which is denoted as $n(Q)$ = 8). Thus, SC-TRUST selects the most reliable SPs for service provision until there is no alternative. However, once $n(Q)$ >= 6, the speedup of the random selection begins to sharply increase to match that of SC-TRUST and eventually a similar speedup is achieved at $n(Q)$ = 10. This is so because SC-TRUST accurately selects the most trustworthy SPs first. Therefore, the marginal increase in speedup reduces as $n(Q)$ approaches $n(S)$ (i.e. the total number of SPs). This is because the random mode is more likely to select trustworthy SPs, which were originally left out, as $n(Q)$ increases. When $n(Q) = n(S)$, there is no difference in speedup between both modes, as the trust model cannot perform any choice because $\binom{n}{n} = 1$.

Therefore, given that the speedup achieved is comparable to the results that were obtained in [67], [120] and [113], we conclude that the transparent trust composition in SC-TRUST yields an accurate trust score with a performance level equalling that of CTRUST. Also, utilising SC-TRUST in a service composition increases the utility gained by the SR with no significant overhead incurred, while providing the trust-based security required for the realization of all the potential benefits of an SOA-based IoT application.

4.2.2 Evaluating Trust Model Accuracy, Resilience and Convergence

In evaluating the accuracy and convergence of SC-TRUST, it is necessary to establish the ground truth. The ground truth value is obtained by computing the trust score of an SP based on the perfect information of its behaviour and trust characteristics. This information is obtained from the record of the random trust behaviour assigned to each SP at the start of the simulation. In real-world applications, neither the SR nor the middleware would have perfect knowledge of the behaviour of any SP. Hence, there is the need for a trust model in the first instance. A trust model which performs accurately in simulations by closely matching known ground truth values in a reasonable time will perform well in real-world applications. Trust resilience is a measure of the ability of the model to adapt to changes in the behaviour of SPs and maintain optimum performance (in terms of the utility derived by the SR) under such circumstances. This is important because the trust characteristics of SPs may be changed during a session due to malicious or non-malicious reasons. A resilient trust model must identify and adapt to these changes and converge to the new trust score quickly and accurately. By doing this, a high-level of utility is maintained (as current high-performing nodes are selected) and trust risks are minimized. The results obtained for SC-TRUST are presented in Fig. 4.4-4.7.

To generate sufficient interactions for these evaluations, the CD sessions were adapted for collaborative streaming. The only difference is that instead of waiting for the entire content to be downloaded before consumption by the SR, each downloaded block is streamed instantly to the SR. If the stream progresses successfully with no block missing, then the SCR increases. The inverse is also true. Similarly, if there is no buffering, then the CBA is increased. If the block arrives but not in time or sequence, then the SR automatically gives a negative feedback score on the CBA. SC-TRUST decomposes the SR's feedback according to the method detailed in Section 4.1.2. Each streaming session includes 400-800 blocks and an equivalent number of interactions.

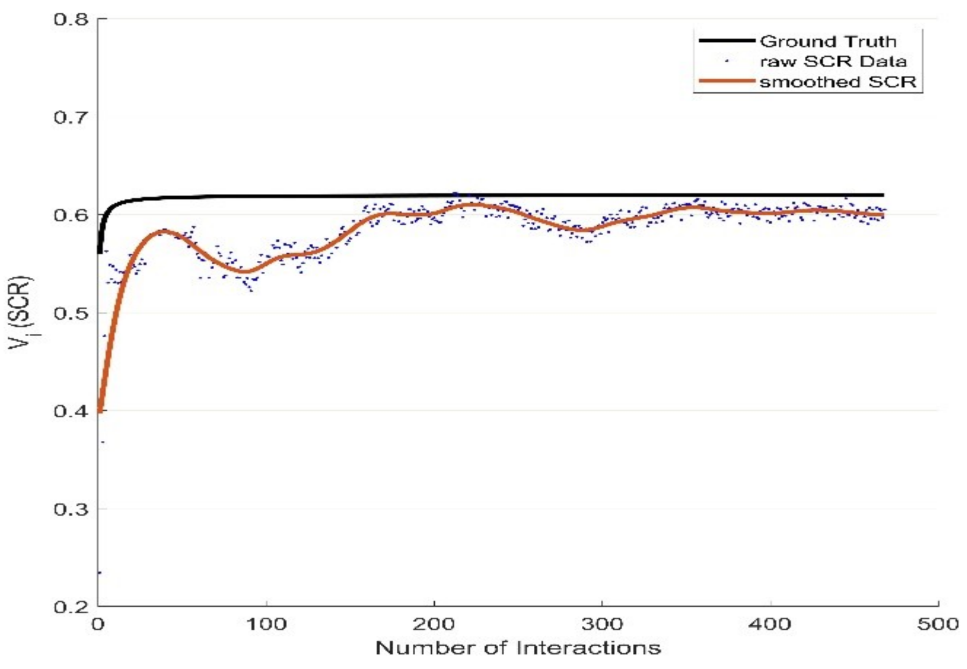


Fig. 4.4 Convergence of a Trust Parameter to the ground truth based on the decomposition of feedback from the SR

Fig. 4.4 shows the results obtained on the SCR parameter after both trust composition and decomposition. Since the SCR parameter is unique to the aggregation service class, the decomposed trust scores only affect the selected aggregation SP. A smoothed plot of the decomposed SCR values in Fig. 4.4 shows that it converges from default state (no previous interaction) to the ground truth in less than 400 interactions. This is slightly higher than the ≈300 interactions required by CTRUST to converge to the ground truth. The reason for this is that the interactions in this simulation are shorter than those in the CTRUST simulation. Overall, the duration of the session in both models are similar. Therefore, SC-TRUST shows a high degree of convergence and accuracy. The fluctuations seen in the raw data are expected, as the utility of the SR and its perception of the composed service are sensitive to changes in the behaviour of the SP. However, if the trust characteristics of the SP are consistent, then the decomposed trust value will always converge to the ground truth.

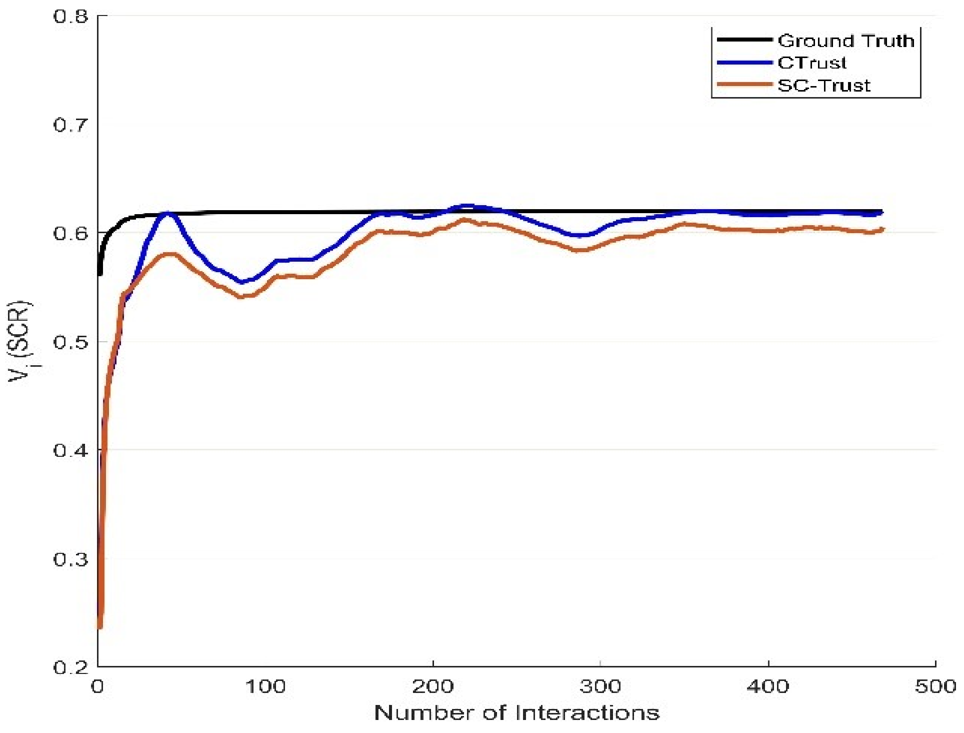


Fig. 4.5 Comparison of the trust accuracy and convergence properties in SC-TRUST and CTRUST

In Fig. 4.5, the SCR measured by SC-TRUST is compared to that of CTRUST. We see that while the value of CTRUST is closer to the ground truth, SC-TRUST follows the same pattern with a slight lag in the measurement of the trust score. However, this difference is statistically insignificant. It should be recalled that these values were decomposed transparently from the SR's feedbacks. Therefore, we can conclude that the trust decomposition method in the model not only converges to the ground truth but offers a level of accuracy on par with CTRUST, but in a service composition context. It can be observed that the trend in both plots is similar. This is so because SC-TRUST is built on top of CTRUST and utilizes some of the latter's methods.

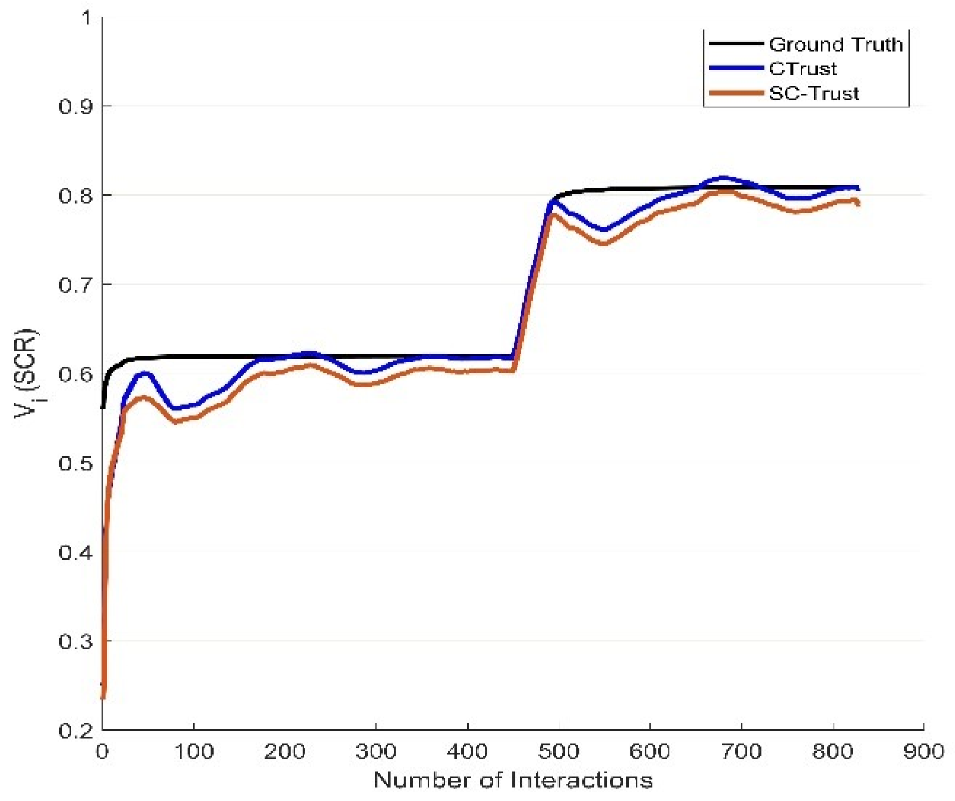


Fig. 4.6 Comparison of the trust resilience on a unique parameter in SC-TRUST and CTRUST

In Fig. 4.6, the trust characteristic (ground truth) of the aggregation SP is modified to a higher value. It can be observed that SC-TRUST converges quickly to the new ground truth within 400 interactions. Also, it should be observed that SC-TRUST is conservative in trust evaluation; that is, the composed or decomposed trust value is never higher than the ground truth. This is in accordance with the specifications in Sections 3.2 and 3.3; therefore, the SR always receives the estimated level of utility or higher, but never lower. This, in turn, increases the SR's trust in both the composing platform and composition services offered on the platform. As previously noted, SC-TRUST slightly lags behind CTRUST in the measured trust value, but this difference is insignificant and expected. CTRUST measures the trust scores from direct interactions; therefore, it is not suitable for transparent trust computations which are required in these service compositions.

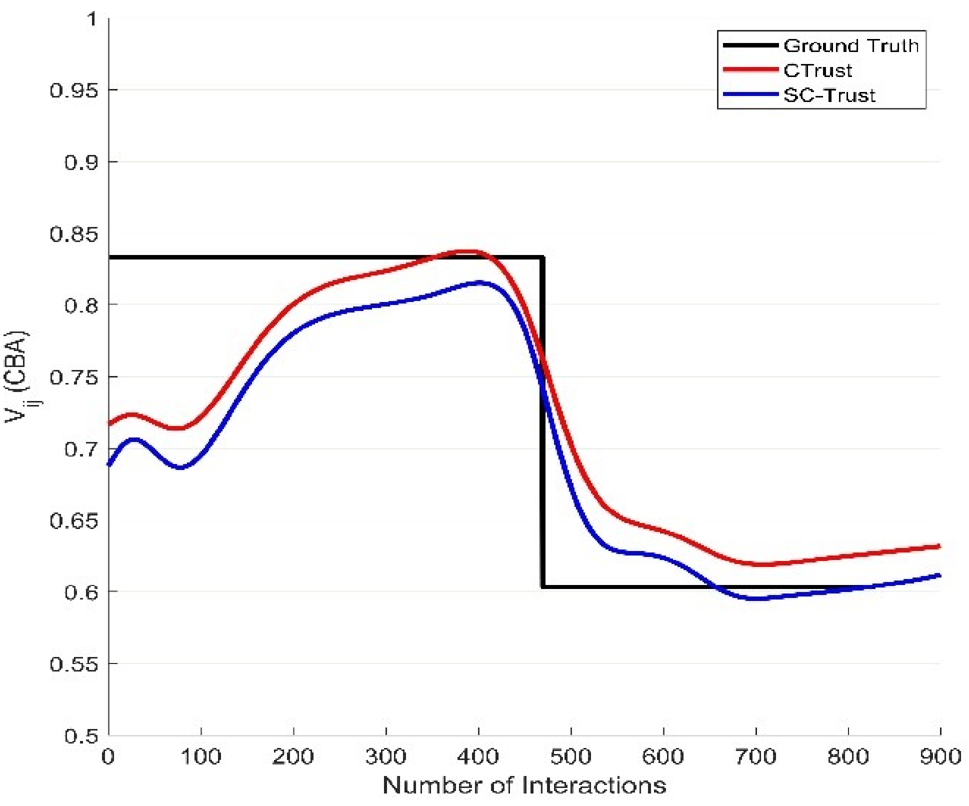


Fig. 4.7 Comparison of the trust resilience on a shared parameter in SC-TRUST and CTRUST

Finally, in Fig. 4.7, we investigate the trust properties of SC-TRUST in the measurement of a shared parameter. The CBA is a parameter common to all download SPs. Therefore, it is more difficult to accurately decompose the SR's feedback in a manner that is fair to each SP.

However, due to the internal use of the IRI parameter to identify malicious and underperforming SPs, SC-TRUST performs reasonably well in converging to the ground truth on this parameter. It can be observed that while CTRUST may produce trust scores higher than the ground truth (and therefore misleading the SR in a service composition), both the composed and decomposed trust scores in SC-TRUST are almost always lower than or equal to the ground truth. After 450 interactions, the trust characteristic of this SP is changed to a lower value. Again, SC-TRUST adapts and converges to the ground truth in a reasonable time, no more than required to establish and converge to the initial ground truth value. After 850 interactions, the value of SC-TRUST is a little higher than the ground truth. In this exceptional case, however, the increase over the ground truth is less than 1%. Therefore, SC-TRUST produces reliable and highly accurate trust scores, within the margin of low and acceptable errors.

From the above analysis, it is evident that SC-TRUST meets Requirements (10-11) for an ideal trust model for IoT service compositions, as enumerated in Section 2.3, in addition to Requirements (1-9) which are fulfilled by inheritance from CTRUST. In comparison to the few existing trust models for service composition, SC-TRUST produces a more accurate and reliable score. For example, the trust scores produced by the models in both [15] and [31] diverge significantly from the ground truth when the percentage of malicious SPs is greater than 30%. In contrast, SC-TRUST shows a high degree of resilience such that even when half of the SPs are malicious, it retains its high accuracy and convergence.

4.3 Chapter Summary

SC-TRUST was designed as a suitable model for service compositions in the SOA-based IoT context, utilising concepts discussed in Section 2.2 and in accordance with the requirements specification in Section 2.3. The simulations and evaluations performed show that the use of SC-TRUST in a service composition increased the utility gained. Also, SC-TRUST showed a robust performance in trust accuracy, convergence, and resilience. Therefore, the model minimizes the impact of trust-related attacks, including ballot stuffing, bad-mouthing, and opportunistic service attacks. SC-TRUST was modelled with the platform characteristics of the IoT in consideration, so its trust evaluations and algorithms require minimal computational resources. In addition, the flexibility of the design ensures that the model can be easily applied to any service composition context. Thus, SC-TRUST addresses critical gaps that exist in the trust management research for the IoT by providing a dynamic, systematic, and holistic approach to trust modelling in SOA-based IoT, especially for trust-based service compositions. In addition, the elegant solutions for transparent trust composition and decomposition are novel contributions. The proposed model has been peer-reviewed and published in the IEEE Internet of Things Journal [121].

CHAPTER 5 CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this thesis, the concept and meanings of computational trust has been formalised and an extensive study has been done on trust modelling, evaluation and management in the IoT. The limitations of existing models, based on a thorough review of the existing literature, have been clearly elucidated. Consequently, requirements for an ideal trust model for collaborative applications and service compositions in the IoT were specified. Also, the techniques for collaborations and service compositions were analysed in detail, in relation to trust management for the IoT. Based on findings, two models were proposed in this work, CTRUST to guide peer selection in collaborative applications, and SC-TRUST for trust-based service composition in the IoT.

In CTRUST, trust is accurately parametrised while recommendations are evaluated through belief functions. The effects of trust decay and maturity on the trust evaluation process were studied. Each trust component is neatly modelled by appropriate mathematical functions. CTRUST was implemented in a collaborative download application and its performance was evaluated based on the utility derived and its trust accuracy, convergence, and resiliency. The results indicate that IoT collaborative applications based on CTRUST gain a significant improvement in performance, in terms of efficiency and security.

SC-TRUST was designed with consideration of the trust properties of service compositions and the effect of service workflows on transparent trust composition and decomposition. It was implemented in a suitable application and its performance, in terms of the utility derived and the trust accuracy, convergence and resiliency, was evaluated. The results show that SC-TRUST improves the quality-of-service compositions and adequately mitigates trust-related attacks, thus increasing both efficiency and security.

5.2 Further Work

The following are research directions stemming from the work done in this thesis:

1. In normalising values on each trust parameter, a linear value function was used. However, sometimes the utility function of the initiator is marginally non-linear. To model trust more accurately in these applications, it would be useful to consider defining a utility function and threshold scales for each parameter in a future work.

2. It would also be useful to extend both models to automate and dynamically update parameter weights, in response to changes in the availability and behaviour of nodes, for example, changes in the ratio of good to malicious service providers.

3. CTRUST could be extended to and evaluated in other collaboration contexts in the IoT, such as collaborative sensing, storage, and processing.

4. It is observed that, by increasing performance and reducing trust attacks, SC-TRUST seems to reduce the overall energy usage required in a composed service. However, this requires further investigation.

5. Finally, the effects non-functional constraints, such as price and energy, exert on the service composition were not considered. The prices which SPs charge for their services may affect their selection

depending on the limit of the SR’s budget. However, the price is not a trust characteristic, as it is not a functional parameter of the composition context, but rather an external constraint on the process. It may be argued that more trustworthy compositions would generally cost more because the price is an incentive for an SP to produce better services. It would be beneficial to study the effects of external constraints, such as price and an SR’s budget, on the service composition process and the trust scores of SPs.

References

[1] S. Li, L. Da Xu, and S. Zhao, “5G Internet of Things: A survey,” *Journal of Industrial Information Integration*, vol. 10, pp. 1–9, Jun. 2018, doi: 10.1016/j.jii.2018.01.005.

[2] C. L. Hsu and J. C. C. Lin, “An empirical examination of consumer adoption of Internet of Things services: Network externalities and concern for information privacy perspectives,” *Computers in Human Behavior*, vol. 62, pp. 516–527, Sep. 2016, doi: 10.1016/j.chb.2016.04.023.

[3] P. Rawat, K. D. Singh, and J. M. Bonnin, “Cognitive radio for M2M and Internet of Things: A survey,” *Computer Communications*, vol. 94, pp. 1–29, Nov. 2016, doi: 10.1016/j.comcom.2016.07.012.

[4] E. Borgia, “The Internet of Things vision: Key features, applications and open issues,” *Computer Communications*, vol. 54, no. 7, pp. 1–31, Dec. 2014, doi: 10.1016/j.comcom.2014.09.008.

[5] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, “Internet of things: Vision, applications and research challenges,” *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, Sep. 2012, doi: 10.1016/j.adhoc.2012.02.016.

[6] J. Ding, M. Nemati, C. Ranaweera, and J. Choi, “IoT Connectivity Technologies and Applications: A Survey,” *IEEE Access*, vol. 8, pp. 67646–67673, 2020, doi: 10.1109/ACCESS.2020.2985932.

[7] H. Al-Hamadi and I. R. Chen, “Trust-Based Decision Making for Health IoT Systems,” *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1408–1419, Oct. 2017, doi: 10.1109/JIOT.2017.2736446.

[8] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013, doi: 10.1016/j.future.2013.01.010.

[9] M. Hamzei and N. Jafari Navimipour, “Toward Efficient Service Composition Techniques in the Internet of Things,” *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3774–3787, Oct. 2018, doi: 10.1109/JIOT.2018.2861742.

[10] M. Sun, Z. Shi, S. Chen, Z. Zhou, and Y. Duan, “Energy-Efficient Composition of Configurable Internet of Things Services,” *IEEE Access*, vol. 5, pp. 25609–25622, 2017, doi: 10.1109/ACCESS.2017.2768544.

[11] Z.-Z. Liu, D.-H. Chu, Z.-P. Jia, J.-Q. Shen, and L. Wang, “Two-stage approach for reliable dynamic Web service composition,” *Knowledge-Based Systems*, vol. 97, pp. 123–143, Apr. 2016, doi: 10.1016/j.knosys.2016.01.010.

[12] J. Crowcroft, *Open Distributed Systems*. USA: Artech House, Inc., 1996.

[13] E. Yahyapour *et al.*, *Towards a Service-Based Internet*, vol. 6481. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[14] W. Z. Khan, Q.-A. Arshad, S. Hakak, M. K. Khan, and Saeed-Ur-Rehman, “Trust Management in Social Internet of Things: Architectures, Recent Advancements, and Future Challenges,” *IEEE Internet of Things Journal*, vol. 8, no. 10, pp. 7768–7788, May 2021, doi: 10.1109/JIOT.2020.3039296.

[15] I.-R. Chen, F. Bao, and J. Guo, “Trust-Based Service Management for Social Internet of Things Systems,” *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 6, pp. 684–696, Nov. 2016, doi: 10.1109/TDSC.2015.2420552.

[16] A. H. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and M. Z. Sheng, “IoT Middleware: A Survey on Issues and Enabling technologies,” *IEEE Internet of Things Journal*, pp. 1–1, 2016, doi: 10.1109/JIOT.2016.2615180.

[17] M. S. Roopa, S. Pattar, R. Buyya, K. R. Venugopal, S. S. Iyengar, and L. M. Patnaik, “Social Internet of Things (SIoT): Foundations, thrust areas, systematic review and future directions,” *Computer Communications*, vol. 139, pp. 32–57, May 2019, doi: 10.1016/j.comcom.2019.03.009.

[18] M. Nitti, R. Girau, and L. Atzori, “Trustworthiness Management in the Social Internet of Things,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 5, pp. 1253–1266, May 2014, doi: 10.1109/TKDE.2013.105.

[19] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Cla, “Middleware for internet of things: A survey,” *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, Feb. 2016, doi: 10.1109/JIOT.2015.2498900.

[20] L. Atzori, A. Iera, G. Morabito, and M. Nitti, “The Social Internet of Things (SIoT) – When social networks meet the Internet of Things: Concept, architecture and network characterization,” *Computer Networks*, vol. 56, no. 16, pp. 3594–3608, Nov. 2012, doi: 10.1016/j.comnet.2012.07.010.

[21] L. Atzori, A. Iera, and G. Morabito, “SIoT: Giving a Social Structure to the Internet of Things,” *IEEE Communications Letters*, vol. 15, no. 11, pp. 1193–1195, Nov. 2011, doi: 10.1109/LCOMM.2011.090911.111340.

[22] M. A. Azad, S. Bag, F. Hao, and A. Shalaginov, “Decentralized Self-Enforcing Trust Management System for Social Internet of Things,” *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 2690–2703, Apr. 2020, doi: 10.1109/JIOT.2019.2962282.

[23] L. Wei, J. Wu, C. Long, and B. Li, “On Designing Context-Aware Trust Model and Service Delegation for Social Internet of Things,” *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4775–4787, Mar. 2021, doi: 10.1109/JIOT.2020.3028380.

[24] S. Sicari, A. Rizzardi, L. A. A. Grieco, and A. Coen-Porisini, “Security, privacy and trust in Internet of Things: The road ahead,” *Computer Networks*, vol. 76, pp. 146–164, Jan. 2015, doi: 10.1016/j.comnet.2014.11.008.

[25] J. A. Stankovic, “Research Directions for the Internet of Things,” *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 3–9, Feb. 2014, doi: 10.1109/JIOT.2014.2312291.

[26] Z. Yan, P. Zhang, and A. V. Vasilakos, “A survey on trust management for Internet of Things,” *Journal of Network and Computer Applications*, vol. 42, pp. 120–134, Jun. 2014, doi: 10.1016/j.jnca.2014.01.014.

[27] C. D. Jensen, “The role of trust in computer security,” in *2012 Tenth Annual International Conference on Privacy, Security and Trust*, Jul. 2012, pp. 236–236, doi: 10.1109/PST.2012.6297950.

[28] W. Leister and T. Schulz, “Ideas for a Trust Indicator in the Internet of Things,” in *SMART 2012—The First International Conference on Smart Systems, Devices and Technologies*, 2012, pp. 31–34, [Online]. Available: http://www.academia.edu/download/40609237/Ideas_for_a_Trust_Indicator_in_the_Inter20151203-10909-1ucwd6v.pdf.

[29] U. E. Tahta, S. Sen, and A. B. Can, “GenTrust: A genetic trust management model for peer-to-peer systems,” *Applied Soft Computing*, vol. 34, pp. 693–704, Sep. 2015, doi: 10.1016/j.asoc.2015.04.053.

[30] Y. Ben Saied, A. Olivereau, D. Zeghlache, and M. Laurent, “Trust management system design for the Internet of Things: A context-aware and multi-service approach,” *Computers & Security*, vol. 39, no. PART B, pp. 351–365, Nov. 2013, doi: 10.1016/j.cose.2013.09.001.

[31] I.-R. Chen, J. Guo, D.-C. Wang, J. J. P. Tsai, H. Al-Hamadi, and I. You, “Trust-Based Service Management for Mobile Cloud IoT Systems,” *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, pp. 246–263, Mar. 2019, doi: 10.1109/TNSM.2018.2886379.

[32] I.-R. Chen, J. Guo, and F. Bao, “Trust Management for SOA-Based IoT and Its Application to Service Composition,” *IEEE Transactions on Services Computing*, vol. 9, no. 3, pp. 482–495, May 2016, doi: 10.1109/TSC.2014.2365797.

[33] D. H. Mcknight and N. L. Chervany, “The Meanings of Trust,” 1996. [Online]. Available:

http://misrc.umn.edu/wpaper/WorkingPapers/9604.pdf.

[34] Y. D. Wang and H. H. Emurian, “An overview of online trust: Concepts, elements, and implications,” *Computers in Human Behavior*, vol. 21, no. 1, pp. 105–125, Jan. 2005, doi: 10.1016/j.chb.2003.11.008.

[35] F. Bao and I.-R. Chen, “Dynamic trust management for internet of things applications,” in *Proceedings of the 2012 international workshop on Self-aware internet of things - Self-IoT ’12*, 2012, p. 1, doi: 10.1145/2378023.2378025.

[36] C. Boudagdigue, A. Benslimane, A. Kobbane, and J. Liu, “Trust Management in Industrial Internet of Things,” *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3667–3682, 2020, doi: 10.1109/TIFS.2020.2997179.

[37] C. Marche and M. Nitti, “Trust-Related Attacks and Their Detection: A Trust Management Model for the Social IoT,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3297–3308, Sep. 2021, doi: 10.1109/TNSM.2020.3046906.

[38] A. Almogren, I. Mohiuddin, I. U. Din, H. Almajed, and N. Guizani, “FTM-IoMT: Fuzzy-Based Trust Management for Preventing Sybil Attacks in Internet of Medical Things,” *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4485–4497, Mar. 2021, doi: 10.1109/JIOT.2020.3027440.

[39] G. Rathee, A. Sharma, R. Kumar, F. Ahmad, and R. Iqbal, “A trust management scheme to secure mobile information centric networks,” *Computer Communications*, vol. 151, pp. 66–75, Feb. 2020, doi: 10.1016/j.comcom.2019.12.024.

[40] D. Chen, G. Chang, D. Sun, J. Li, J. Jia, and X. Wang, “TRM-IoT: A trust management model based on fuzzy reputation for internet of things,” *Computer Science and Information Systems*, vol. 8, no. 4, pp. 1207–1228, 2011, doi: 10.2298/CSIS110303056C.

[41] S. Talbi and A. Bouabdallah, “Interest-based trust management scheme for social internet of things,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 11, no. 3, pp. 1129–1140, Mar. 2020, doi: 10.1007/s12652-019-01256-8.

[42] Fenye Bao and Ing-Ray Chen, “Trust management for the internet of things and its application to service composition,” in *2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, Jun. 2012, pp. 1–6, doi: 10.1109/WoWMoM.2012.6263792.

[43] J. Guo and I.-R. Chen, “A Classification of Trust Computation Models for Service-Oriented Internet of Things Systems,” in *2015 IEEE International Conference on Services Computing*, Jun. 2015, pp. 324–331, doi: 10.1109/SCC.2015.52.

[44] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, “The Eigentrust algorithm for reputation management in P2P networks,” in *Proceedings of the twelfth international conference on World Wide Web - WWW ’03*, May 2003, p. 640, doi: 10.1145/775240.775242.

[45] Li Xiong and Ling Liu, “PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 07, pp. 843–857, Jul. 2004, doi: 10.1109/TKDE.2004.1318566.

[46] W. Sherchan, S. Nepal, and C. Paris, “A survey of trust in social networks,” *ACM Computing Surveys*, vol. 45, no. 4, pp. 1–33, Aug. 2013, doi: 10.1145/2501654.2501661.

[47] S. Valenzuela, N. Park, and K. F. Kee, “Is There social capital in a social network site?: Facebook use and college student’s life satisfaction, trust, and participation1,” *Journal of Computer-Mediated Communication*, vol. 14, no. 4, pp. 875–901, Jul. 2009, doi: 10.1111/j.1083-6101.2009.01474.x.

[48] J. Huang, F. Nie, H. Huang, Y. Lei, and C. Ding, “Social trust prediction using rank-κ matrix recovery,” *IJCAI International Joint Conference on Artificial Intelligence*, pp. 2647–2653, 2013.

[49] J. Huang, F. Nie, H. Huang, Y. C. Tu, and Y. Lei, “Social trust prediction using heterogeneous networks,” *ACM Transactions on Knowledge Discovery from Data*, vol. 7, no. 4, pp. 1–21, Nov. 2013, doi: 10.1145/2541268.2541270.

[50] B. Yang, Y. Lei, J. Liu, and W. Li, “Social Collaborative Filtering by Trust,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 8, pp. 1633–1647, Aug. 2017, doi: 10.1109/TPAMI.2016.2605085.

[51] J. Masthoff, “Group Recommender Systems: Combining Individual Models,” in *Recommender Systems Handbook*, 2011, pp. 677–702.

[52] G. Guo, J. Zhang, and D. Thalmann, “Merging trust in collaborative filtering to alleviate data sparsity and cold start,” *Knowledge-Based Systems*, vol. 57, pp. 57–68, Feb. 2014, doi: 10.1016/j.knosys.2013.12.007.

[53] J. Guo, Y. Zhou, P. Zhang, B. Song, and C. Chen, “Trust-aware recommendation based on heterogeneous multi-relational graphs fusion,” *Information Fusion*, vol. 74, pp. 87–95, Oct. 2021, doi: 10.1016/j.inffus.2021.04.001.

[54] Y. Li, G. Kou, G. Li, and H. Wang, “Multi-attribute group decision making with opinion dynamics based on social trust network,” *Information Fusion*, vol. 75, pp. 102–115, Nov. 2021, doi: 10.1016/j.inffus.2021.04.010.

[55] H. Ma, I. King, and M. R. Lyu, “Learning to recommend with explicit and implicit social relations,” *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 1–19, Apr. 2011, doi: 10.1145/1961189.1961201.

[56] W. Li and H. Song, “ART: An Attack-Resistant Trust Management Scheme for Securing Vehicular Ad Hoc Networks,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 960–969, Apr. 2016, doi: 10.1109/TITS.2015.2494017.

[57] K. Thirunarayan, D. K. Althuru, C. A. Henson., and A. P. Sheth, “A local qualitative approach to referral and functional trust,” *Proceedings of the 4th Indian International Conference on Artificial Intelligence, IICAI 2009*, pp. 574–588, 2009, [Online]. Available: https://corescholar.libraries.wright.edu/knoesis/675.

[58] P. Anantharam, C. A. Henson, K. Thirunarayan, and A. P. Sheth, “Trust model for semantic sensor and social networks: A preliminary report,” in *Proceedings of the IEEE 2010 National Aerospace and Electronics Conference, NAECON 2010*, Jul. 2010, pp. 1–5, doi: 10.1109/NAECON.2010.5712915.

[59] T. Wang *et al.*, “A Comprehensive Trustworthy Data Collection Approach in Sensor-Cloud Systems,” *IEEE Transactions on Big Data*, vol. 8, no. 1, pp. 140–151, Feb. 2018, doi: 10.1109/tbdata.2018.2811501.

[60] P. Munoz, A. Perez-Vereda, N. Moreno, J. Troya, and A. Vallecillo, “Incorporating Trust into Collaborative Social Computing Applications,” in *Proceedings - 2021 IEEE 25th International Enterprise Distributed Object Computing Conference, EDOC 2021*, Oct. 2021, pp. 21–30, doi: 10.1109/EDOC52215.2021.00020.

[61] K. Thiranarayan, P. Anantharam, C. A. Henson, and A. P. Sheth, “Some trust issues in social networks and sensor networks,” in *2010 International Symposium on Collaborative Technologies and Systems, CTS 2010*, 2010, pp. 573–580, doi: 10.1109/CTS.2010.5478462.

[62] C. Haydar, A. Roussanaly, and A. Boyer, “Local trust versus global trust networks in subjective logic,” in *Proceedings - 2013 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2013*, Nov. 2013, vol. 1, pp. 29–36, doi: 10.1109/WI-IAT.2013.5.

[63] W. Jiang, G. Wang, and J. Wu, “Generating trusted graphs for trust evaluation in online social networks,” *Future Generation Computer Systems*, vol. 31, no. 1, pp. 48–58, Feb. 2014, doi: 10.1016/j.future.2012.06.010.

[64] S. Asiri and A. Miri, “An IoT trust and reputation model based on recommender systems,” in *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, Dec. 2016, pp. 561–568, doi: 10.1109/PST.2016.7907017.

[65] S. K. M. Yi, M. Steyvers, M. D. Lee, and M. J. Dry, “The Wisdom of the Crowd in Combinatorial Problems,” *Cognitive Science*, vol. 36, no. 3, pp. 452–470, Apr. 2012, doi: 10.1111/j.1551-6709.2011.01223.x.

[66] N. S. Nizamkari, “A graph-based trust-enhanced recommender system for service selection in IOT,” in *2017 International Conference on Inventive Systems and Control (ICISC)*, Jan. 2017, pp. 1–5, doi: 10.1109/ICISC.2017.8068714.

[67] A. A. Adewuyi, H. Cheng, Q. Shi, J. Cao, A. MacDermott, and X. Wang, “CTRUST: A dynamic trust model for collaborative applications in the internet of things,” *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5432–5445, Feb. 2019, doi: 10.1109/JIOT.2019.2902022.

[68] M. Salimitari, M. Chatterjee, and Y. P. Fallah, “A survey on consensus methods in blockchain for resource-constrained IoT networks,” *Internet of Things (Netherlands)*, vol. 11, p. 100212, Sep. 2020, doi: 10.1016/j.iot.2020.100212.

[69] L. Lao, Z. Li, S. Hou, B. Xiao, S. Guo, and Y. Yang, “A survey of IoT applications in blockchain systems: Architecture, consensus, and traffic modeling,” *ACM Computing Surveys*, vol. 53, no. 1, pp. 1–32, Jan. 2020, doi: 10.1145/3372136.

[70] O. Alfandi, S. Otoum, and Y. Jararweh, “Blockchain Solution for IoT-based Critical Infrastructures: Byzantine Fault Tolerance,” in *Proceedings of IEEE/IFIP Network Operations and Management Symposium 2020: Management in the Age of Softwarization and Artificial Intelligence, NOMS 2020*, Apr. 2020, pp. 1–4, doi:

10.1109/NOMS47738.2020.9110312.

[71] R. Han, V. Gramoli, and X. Xu, “Evaluating Blockchains for IoT,” in *2018 9th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2018 - Proceedings*, Feb. 2018, vol. 2018-Janua, pp. 1–5, doi: 10.1109/NTMS.2018.8328736.

[72] W. Viriyasitavat, L. Da Xu, Z. Bi, and A. Sapsomboon, “Blockchain-based business process management (BPM) framework for service composition in industry 4.0,” *Journal of Intelligent Manufacturing*, May 2018, doi: 10.1007/s10845-018-1422-y.

[73] P. Wang *et al.*, “Smart Contract-Based Negotiation for Adaptive QoS-Aware Service Composition,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 6, pp. 1403–1420, Jun. 2019, doi: 10.1109/TPDS.2018.2885746.

[74] C. Yu, L. Zhang, W. Zhao, and S. Zhang, “A blockchain-based service composition architecture in cloud manufacturing,” *International Journal of Computer Integrated Manufacturing*, pp. 1–15, Feb. 2019, doi: 10.1080/0951192X.2019.1571234.

[75] “IEEE Standard for Framework of Blockchain-based Internet of Things (IoT) Data Management,” *IEEE Std 2144.1-2020*. pp. 1–20, 2021, doi: 10.1109/IEEESTD.2021.9329260.

[76] D. E. Kouicem, Y. Imine, A. Bouabdallah, and H. Lakhlef, “A Decentralized Blockchain-Based Trust Management Protocol for the Internet of Things,” *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2020, doi: 10.1109/TDSC.2020.3003232.

[77] M. Zhaofeng, W. Lingyun, W. Xiaochang, W. Zhen, and Z. Weizhe, “Blockchain-Enabled Decentralized Trust Management and Secure Usage Control of IoT Big Data,” *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4000–4015, May 2020, doi: 10.1109/JIOT.2019.2960526.

[78] X. Liu, H. Huang, F. Xiao, and Z. Ma, “A Blockchain-Based Trust Management With Conditional Privacy-Preserving Announcement Scheme for VANETs,” *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4101–4112, May 2020, doi: 10.1109/JIOT.2019.2957421.

[79] S. Nakamoto, “A peer-to-peer electronic cash system,” 2008, [Online]. Available: http://www.bitcoin.org/bitcoin.pdf.

[80] W. Gu, J. Li, and Z. Tang, “A Survey on Consensus Mechanisms for Blockchain Technology,” in *Proceedings - 2021 International Conference on Artificial Intelligence, Big Data and Algorithms, CAIBDA 2021*, May 2021, pp. 46–49, doi: 10.1109/CAIBDA53561.2021.00017.

[81] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, “An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends,” in *Proceedings - 2017 IEEE 6th International Congress on Big Data, BigData Congress 2017*, Jun. 2017, pp. 557–564, doi: 10.1109/BigDataCongress.2017.85.

[82] L. M. Bach, B. Mihaljevic, and M. Zagar, “Comparative analysis of blockchain consensus algorithms,” in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018 - Proceedings*, May 2018, pp. 1545–1550, doi: 10.23919/MIPRO.2018.8400278.

[83] S. M. H. Bamakan, A. Motavali, and A. Babaei Bondarti, "A survey of blockchain consensus algorithms performance evaluation criteria," *Expert Systems with Applications*, vol. 154, p. 113385, Sep. 2020, doi: 10.1016/j.eswa.2020.113385.

[84] X. Wang *et al.*, "Survey on blockchain for Internet of Things," *Computer Communications*, vol. 136, pp. 10–29, Feb. 2019, doi: 10.1016/j.comcom.2019.01.006.

[85] C. Zhang, X. Cao, J. Liu, and K. Ren, "Proof of comprehensive performance," in *SBC 2021 - Proceedings of the 9th International Workshop on Security in Blockchain and Cloud Computing, co-located with ASIA CCS 2021*, May 2021, pp. 29–34, doi: 10.1145/3457977.3460296.

[86] N. Al Asad, M. T. Elahi, A. Al Hasan, and M. A. Yousuf, "Permission-based blockchain with proof of authority for secured healthcare data sharing," in *2020 2nd International Conference on Advanced Information and Communication Technology, ICAICT 2020*, Nov. 2020, pp. 35–40, doi: 10.1109/ICAICT51780.2020.9333488.

[87] P. K. Singh, R. Singh, S. K. Nandi, and S. Nandi, "Managing Smart Home Appliances with Proof of Authority and Blockchain," in *Communications in Computer and Information Science*, vol. 1041, 2019, pp. 221–232.

[88] M. Borge, E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, and B. Ford, "Proof-of-Personhood: Redemocratizing Permissionless Cryptocurrencies," in *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, Apr.

2017, pp. 23–26, doi: 10.1109/EuroSPW.2017.46.

[89] B. Houtan, A. S. Hafid, and D. Makrakis, “A Survey on Blockchain-Based Self-Sovereign Patient Identity in Healthcare,” *IEEE Access*, vol. 8, pp. 90478–90494, 2020, doi: 10.1109/ACCESS.2020.2994090.

[90] J. Bou Abdo, R. El Sibai, and J. Demerjian, “Permissionless proof-of-reputation-X: A hybrid reputation-based consensus algorithm for permissionless blockchains,” *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, Jan. 2021, doi: 10.1002/ett.4148.

[91] Q. Zhuang, Y. Liu, L. Chen, and Z. Ai, “Proof of reputation: A reputation-based consensus protocol for blockchain based systems,” in *ACM International Conference Proceeding Series*, Jul. 2019, pp. 131–138, doi: 10.1145/3343147.3343169.

[92] T. Do, T. Nguyen, and H. Pham, “Delegated proof of reputation: A novel blockchain consensus,” in *ACM International Conference Proceeding Series*, Jul. 2019, pp. 90–98, doi: 10.1145/3343147.3343160.

[93] H. Chai, S. Leng, K. Zhang, and S. Mao, “Proof-of-Reputation Based-Consortium Blockchain for Trust Resource Sharing in Internet of Vehicles,” *IEEE Access*, vol. 7, pp. 175744–175757, 2019, doi: 10.1109/ACCESS.2019.2956955.

[94] C. Liu, K. K. Chai, X. Zhang, and Y. Chen, “Peer-to-peer electricity trading system: smart contracts based proof-of-benefit consensus protocol,” *Wireless Networks*, vol. 27, no. 6, pp. 4217–4228, Aug. 2021, doi: 10.1007/s11276-019-01949-0.

[95] C. Liu, K. K. Chai, X. Zhang, and Y. Chen, "Proof-of-benefit: A blockchain-enabled ev charging scheme," in *IEEE Vehicular Technology Conference*, Apr. 2019, vol. 2019-April, pp. 1–6, doi: 10.1109/VTCSpring.2019.8746399.

[96] A. Yakovenko, "Solana : A new architecture for a high performance blockchain," *Whitepaper*, pp. 1–32, 2019, [Online]. Available: https://solana.com/solana-whitepaper.pdf.

[97] Raghav, N. Andola, S. Venkatesan, and S. Verma, "PoEWAL: A lightweight consensus mechanism for blockchain in IoT," *Pervasive and Mobile Computing*, vol. 69, p. 101291, Nov. 2020, doi: 10.1016/j.pmcj.2020.101291.

[98] S. Tang, J. Zheng, Y. Deng, and Q. Cao, "Resisting newborn attacks via shared Proof-of-Space," *Journal of Parallel and Distributed Computing*, vol. 150, pp. 85–95, Apr. 2021, doi: 10.1016/j.jpdc.2020.12.011.

[99] A. S. Yadav, N. Singh, and D. S. Kushwaha, "A scalable trust based consensus mechanism for secure and tamper free property transaction mechanism using DLT," *International Journal of Systems Assurance Engineering and Management*, Sep. 2021, doi: 10.1007/s13198-021-01335-0.

[100] A. S. Yadav and D. S. Kushwaha, "Blockchain-based digitization of land record through trust value-based consensus algorithm," *Peer-to-Peer Networking and Applications*, vol. 14, no. 6, pp. 3540–3558, Nov. 2021, doi: 10.1007/s12083-021-01207-1.

[101] J. Yun, Y. Goh, and J. M. Chung, "Trust-Based Shard Distribution Scheme for Fault-Tolerant Shard Blockchain Networks," *IEEE*

Access, vol. 7, pp. 135164–135175, 2019, doi: 10.1109/ACCESS.2019.2942003.

[102] A. Prabhakar and T. Anjali, “TCON - A lightweight Trust-dependent Consensus framework for blockchain,” in *2019 11th International Conference on Communication Systems and Networks, COMSNETS 2019*, Jan. 2019, pp. 1–6, doi: 10.1109/COMSNETS.2019.8711448.

[103] J.-H. Cho, K. Chan, and S. Adali, “A Survey on Trust Modeling,” *ACM Computing Surveys*, vol. 48, no. 2, pp. 1–40, Nov. 2015, doi: 10.1145/2815595.

[104] Z. Lin and L. Dong, “Clarifying Trust in Social Internet of Things,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 2, pp. 234–248, Apr. 2017, doi: 10.1109/TKDE.2017.2762678.

[105] H. Xia, F. Xiao, S. Zhang, C. Hu, and X. Cheng, “Trustworthiness Inference Framework in the Social Internet of Things: A Context-Aware Approach,” in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, Apr. 2019, pp. 838–846, doi: 10.1109/INFOCOM.2019.8737491.

[106] T. Wang, G. Zhang, A. Liu, M. Z. A. Bhuiyan, and Q. Jin, “A Secure IoT Service Architecture With an Efficient Balance Dynamics Based on Cloud and Edge Computing,” *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4831–4843, Jun. 2019, doi: 10.1109/JIOT.2018.2870288.

[107] C. Esposito, O. Tamburis, X. Su, and C. Choi, “Robust Decentralised Trust Management for the Internet of Things by Using Game Theory,” *Information Processing & Management*, vol.

57, no. 6, p. 102308, Nov. 2020, doi: 10.1016/j.ipm.2020.102308.

[108] J. Chen, Z. Tian, X. Cui, L. Yin, and X. Wang, "Trust architecture and reputation evaluation for internet of things," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 8, pp. 3099–3107, Aug. 2019, doi: 10.1007/s12652-018-0887-z.

[109] J. Guo, I.-R. Chen, and J. J. P. Tsai, "A survey of trust computation models for service management in internet of things systems," *Computer Communications*, vol. 97, pp. 1–14, Jan. 2017, doi: 10.1016/j.comcom.2016.10.012.

[110] D. Gessner, A. Olivereau, A. S. Segura, and A. Serbanati, "Trustworthy Infrastructure Services for a Secure and Privacy-Respecting Internet of Things," in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, Jun. 2012, pp. 998–1003, doi: 10.1109/TrustCom.2012.286.

[111] A. Jøsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decision Support Systems*, vol. 43, no. 2, pp. 618–644, Mar. 2007, doi: 10.1016/j.dss.2005.05.019.

[112] E. ElSalamouny, K. T. Krukow, and V. Sassone, "An analysis of the exponential decay principle in probabilistic trust models," *Theoretical Computer Science*, vol. 410, no. 41, pp. 4067–4084, Sep. 2009, doi: 10.1016/j.tcs.2009.06.011.

[113] G. Ananthanarayanan, V. N. Padmanabhan, L. Ravindranath, and C. a. Thekkath, "COMBINE," in *Proceedings of the 5th international conference on Mobile systems, applications and*

services - MobiSys '07, 2007, p. 286, doi: 10.1145/1247660.1247693.

[114] A. Sharma, V. Navda, R. Ramjee, V. N. Padmanabhan, and E. M. Belding, "Cool-Tether : Energy Efficient On-the-fly WiFi Hot-spots," *ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, pp. 109–120, 2009, doi: 10.1145/1658939.1658952.

[115] P. Jassal, K. Yadav, A. Kumar, V. Naik, V. Narwal, and A. Singh, "Unity: Collaborative downloading content using co-located socially connected peers," in *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, Mar. 2013, pp. 66–71, doi: 10.1109/PerComW.2013.6529458.

[116] H. A. Simon, "Rational choice and the structure of the environment.," *Psychological Review*, vol. 63, no. 2, pp. 129–138, Mar. 1956, doi: 10.1037/h0042769.

[117] M. M. Rahman, "Assessing small group assignments," in *7th Brunei International Conference on Engineering and Technology 2018 (BICET 2018)*, 2018, pp. 49 (4 pp.)-49 (4 pp.), doi: 10.1049/cp.2018.1546.

[118] L. Johnston and L. Miles, "Assessing contributions to group assignments," *Assessment and Evaluation in Higher Education*, vol. 29, no. 6, pp. 751–768, Dec. 2004, doi: 10.1080/0260293042000227272.

[119] J. Goldfinch and R. Raeside, "DEVELOPMENT OF A PEER ASSESSMENT TECHNIQUE FOR OBTAINING INDIVIDUAL

MARKS ON A GROUP PROJECT," *Assessment & Evaluation in Higher Education*, vol. 15, no. 3, pp. 210–231, Sep. 1990, doi: 10.1080/0260293900150304.

[120] G. Iosifidis, L. Gao, J. Huang, and L. Tassiulas, "Efficient and Fair Collaborative Mobile Internet Access," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1386–1400, Jun. 2017, doi: 10.1109/TNET.2016.2638939.

[121] A. A. Adewuyi, H. Cheng, Q. Shi, J. Cao, X. Wang, and B. Zhou, "SC-TRUST: A Dynamic Model for Trustworthy Service Composition in the Internet of Things," *IEEE Internet of Things Journal*, pp. 1–1, 2021, doi: 10.1109/JIOT.2021.3097980.

APPENDIX A: DEFINITION OF TERMS

- Functional Trust: Functional trust refers to the degree to which a trustor believes that the trustee is both competent and willing to execute required task(s) reliably in a specific functional context, which defines the trust scope. It is dependent on the trustee's ability to perform certain functions and its historical performance as measured directly by the trustor. It is usually asymmetric, and not necessarily or transitive. Functional trust relationships are formed based on the ability to fulfil tasks in a specific context and are isolated to that context. See Section 2.1.2 for a further discussion of this term.

- Social Trust: The social trust between nodes usually refers to the degree or strength of the connection between them. Consequently, factors such as similarity, colocation, friendliness, and honesty are primary determinants or parameters of the trust score of the relationship. It is usually symmetric, transitive, and mutual. See See Section 2.1.2 for a further discussion of this term.

- Objective Parameter: An objective trust parameter is one which is based on a functional parameter of the context and can be assessed quantitatively according to some metric or rule that has been defined within that context, thus ensuring its measurement is free from bias. Its assessment is standardised; there exists a clear definition of distance and its measurement.

- Subjective Parameter: A subjective trust parameter is assessed based on the bias of the trustor. A parameter will be subjective and biased to the opinion of the rider if there is no defined metric or standard for its measurement. Therefore, a parameter may be subjective in one context but objective in another depending on the existence of a standardised assessment and scoring system in that context.

- Recommendation: A recommendation, also known as a referral trust score, is trust value received from a third party about another node, usually for which the receiving party has little or no previous trust history. For example, if node A accesses node C, and passes that trust score to node B, then B is said to have received a recommendation on C from A.

- Trustworthiness: The degree, assessed *post priori,* to which a node reliably performs an assigned task in line with the *a priori* estimated trust value. In this thesis, this is the same as the current value of the functional trust score.

APPENDIX B: MAIN CODEBASE

```
// NewPeer.CS

using System;
using System.Collections.Generic;
using System.Linq;
using System.ComponentModel;
using System.Threading;
using System.Net;
using System.Diagnostics;
using System.IO;
using System.Security.Cryptography;
using System.Xml.Linq;
using NewFriends.Properties;

namespace NewFriends
{
  class PeerNode
  {
    public string ID { get; private set; }
    //number of assigned blocks and number of successful blocks downloaded.
    //members, e.g. of class, are automatically assigned. Locals (of methods) aren't. rememeber...
    long nAssigned, nSuccessful, TotalBytesDownloaded; //sim=simulated.
    double simReliability, simBWFactor, simRiskFactor, TotalTimeTaken;//in milliseconds
    public int SessionCount { get; set; }
    public BackgroundWorker BgWorker;
    public static XElement PL;
    public static Dictionary<string, PeerNode> PeerList;

    public PeerNode()
    {
      ID = Guid.NewGuid().ToString("n").Substring(0, 16);
      InitBgWorker();
      simReliability = SetReliability();
      simBWFactor = SetBandwidthFactor();
      simRiskFactor = SetRiskFactor();
    }

    public static void LoadPeerList()
    {//try to get stored PeerList else return leaving PL null.
      if (Settings.Default.PeerList == "") return;
      PL = XElement.Parse(Settings.Default.PeerList);
      if (PL.Elements("Peer").Count() != 0)
      {
        PeerList = new Dictionary<string, PeerNode>();
        foreach (var l in PL.Elements("Peer"))
        {
          string s = l.Element("ID").Value;
          PeerList.Add(s, new PeerNode());
          PeerList[s].ID = s;
          PeerList[s].nAssigned = (long)l.Element("nAssigned");
          PeerList[s].nSuccessful = (long)l.Element("nSuccessful");
          PeerList[s].TotalBytesDownloaded = (long)l.Element("TotalBytesDownloaded");
          PeerList[s].TotalTimeTaken = (double)l.Element("TotalTimeTaken");
          PeerList[s].SessionCount = (int)l.Element("SessionCount");
          PeerList[s].simBWFactor = (double)l.Element("simBWFactor");
          PeerList[s].simReliability = (double)l.Element("simReliability");
          PeerList[s].simRiskFactor = (double)l.Element("simRiskFactor");
```

```
            }

            foreach (PeerNode p in PeerList.Values)
            {
                File.AppendAllText(@"C:\NewFriends\PeerList.txt", string.Format("{0,-16} \t{1,-
5} \t{2,-5} \t{3,-
5}{4}", p.ID, p.simBWFactor, p.simReliability, p.simRiskFactor, Environment.NewLine));
            }
        }
    }

    public static void CreatePeerList()
    {
        PL = new XElement("PeerList", new XComment("List of Discovered Peers"));

        foreach (PeerNode p in PeerList.Values)
        {
            PL.Add(new XElement("Peer",
                new XElement("ID", p.ID),
                new XElement("nAssigned", p.nAssigned),
                new XElement("nSuccessful", p.nSuccessful),
                new XElement("TotalBytesDownloaded", p.TotalBytesDownloaded),
                new XElement("TotalTimeTaken", p.TotalTimeTaken),
                new XElement("SessionCount", p.SessionCount),
                new XElement("simBWFactor", p.simBWFactor),
                new XElement("simReliability", p.simReliability),
                new XElement("simRiskFactor", p.simRiskFactor)
                ));
            File.AppendAllText(@"C:\NewFriends\PeerList.txt", string.Format("{0,-16} \t{1,-5} \t{2,-
5} \t{3,-5}{4}", p.ID, p.simBWFactor, p.simReliability, p.simRiskFactor, Environment.NewLine));
        }
        Settings.Default.PeerList = PL.ToString();
        Settings.Default.Save();
    }

    public static void UpdatePLSettings(string id)
    {
        foreach (var peer in PL.Elements("Peer"))
        {
            if (peer.Element("ID").Value == id)
            {
                try
                {
                    peer.ReplaceNodes(new XElement("ID", id),
                        new XElement("nAssigned", PeerList[id].nAssigned),
                new XElement("nSuccessful", PeerList[id].nSuccessful),
                new XElement("TotalBytesDownloaded", PeerList[id].TotalBytesDownloaded),
                new XElement("TotalTimeTaken", PeerList[id].TotalTimeTaken),
                new XElement("SessionCount", PeerList[id].SessionCount),
                new XElement("simBWFactor", PeerList[id].simBWFactor),
                new XElement("simReliability", PeerList[id].simReliability),
                new XElement("simRiskFactor", PeerList[id].simRiskFactor)
                );
                    Settings.Default.PeerList = PL.ToString();
                    Settings.Default.Save();
                }
                catch (Exception)
                {
                    //Wait till next opportunity to try;
                }
            }
        }
    }

    private static double SetBandwidthFactor()
    {
        using (RNGCryptoServiceProvider key = new RNGCryptoServiceProvider())
        {
            byte[] b = new byte[4];
            key.GetNonZeroBytes(b);
            Random rng = new Random(1 * b[1] * b[2] * b[3]);
            return (double)rng.Next(50, 101) / 100;
        }
    }

    private static double SetReliability()
    {
        using (RNGCryptoServiceProvider key = new RNGCryptoServiceProvider())
        {
            byte[] b = new byte[4];
            key.GetNonZeroBytes(b);
            Random rng = new Random(1 * b[1] * b[2] * b[3]);
            return (double)rng.Next(50, 101) / 100; //using bigger values for more space => better rand
omness?
        }
    }

    private static double SetRiskFactor()
    {
        using (RNGCryptoServiceProvider key = new RNGCryptoServiceProvider())
        {
            byte[] b = new byte[4];
            key.GetNonZeroBytes(b);
            Random rng = new Random(1 * b[1] * b[2] * b[3]);
            return (double)rng.Next(0, 51) / 100; //return is btw 0.00 and and 0.50.
        }
    }

    public void InitBgWorker()
    {
        if (simBWFactor == 0) BgWorker = new BackgroundWorker();
        BgWorker.WorkerReportsProgress = true;
        BgWorker.WorkerSupportsCancellation = true;
        BgWorker.DoWork += new DoWorkEventHandler(BgWorker_DoWork);
        BgWorker.RunWorkerCompleted += new RunWorkerCompletedEventHandler(BgWorker_R
unWorkerCompleted);
        BgWorker.ProgressChanged += new ProgressChangedEventHandler(BgWorker_ProgressCha
nged);
    }
    object[] SimulateDownloadBlock(Block block, BackgroundWorker worker, DoWorkEventArgs
e)
    {
        object[] obj = new object[2];
        obj[0] = block.Position;
        if (worker.CancellationPending)
        {
            e.Cancel = true;
        }
        else
        {
```

```
            try
            {
                /* assume reference bandwidth, B = 2 MB/s (2048KB/s  or 16Mb/s).
                A block will take (block.size (in bytes) / peer.SimBWFactor (in B)) time to download
                This is equal to (block.size bytes)/(peer.SimBWFactor*2048*1024 bytes/second)
                We multiply that by 1000 to get the sleep time, in milliseconds, of the thread, since we ar
e simulating
                */
                int sleepTime = (int)Math.Round((block.Size * 1000) / (simBWFactor * 2048 * 1024));
                Thread.Sleep(sleepTime);
                obj[1] = MyMethods.Table((byte)1, block.Size).ToArray();
            }
            catch (Exception ex)
            {
                //notify peer of error
                Debug.WriteLine(ex.Message + " " + block.Position, "Error");
                //Thread.Sleep(2000);
                // e.Cancel = true;
                //MainForm.BlockList[block.Position].status = BlockStatus.NotDownloaded;
            }
        }
        return obj;
    }

    void BgWorker_DoWork(object sender, DoWorkEventArgs e)
    {
        // Get the BackgroundWorker that raised this event.
        BackgroundWorker worker = sender as BackgroundWorker;

        // Assign the result of the computation to the Result property of the DoWorkEventArgs object.
 This will be available to the RunWorkerCompleted eventhandler.

        e.Result = SimulateDownloadBlock((Block)e.Argument, worker, e);
    }

    void BgWorker_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
    {
        nAssigned += 1;
        if (e.Cancelled)
        { /*echo to peer: "You cancelled the download"*/
            return;
        }
        object[] obj = e.Result as object[];
        long n = (long)obj[0];
        if (FrontEnd.BlockList[n].Assignee == ID)
        {
            if (e.Error != null || obj[1] == null || SimulateFailure(simReliability))
            {
                /*echo to peer: "Download failed"*/
                Debug.WriteLine("Position {0} failed", n);
                FrontEnd.BlockList[n].Status = BlockStatus.NotDownloaded;
            }
            else
            {
                if (SimulateTampering(simRiskFactor)) { TamperWithBlock(ref obj[1]); }
                FrontEnd.BlockList[n].content = obj[1] as byte[];
                nSuccessful += 1;
                TotalBytesDownloaded += (obj[1] as byte[]).Length;
                double TimeTaken = Math.Round((DateTime.UtcNow - FrontEnd.BlockList[n].StartTim
e).TotalMilliseconds);
```

```
                TotalTimeTaken += TimeTaken;
                Debug.WriteLine("Position {0} returned as completed; Time Taken: {1}ms", n, TimeTa
ken);
                FrontEnd.BlockList[n].Status = BlockStatus.Downloaded;
            }
        }
        else { Debug.WriteLine("Block in position {0} has been reassigned somewhere along the way
. Original assignee took longer than could be allowed", n);
            /*also echo to this to peer in a message: */  }
    }

    void BgWorker_ProgressChanged(object sender, ProgressChangedEventArgs e)
    {

    }

    private static bool SimulateFailure(double s)
    {
        using (RNGCryptoServiceProvider key = new RNGCryptoServiceProvider())
        {
            byte[] b = new byte[4];
            key.GetNonZeroBytes(b);
            Random rng = new Random(1 * b[1] * b[2] * b[3]);
            //reliability happens 100*s % of the time, so failure happens the rest of the time.
            return !(rng.NextDouble() < s);
        }
    }

    private static bool SimulateTampering(double r)
    {
        using (RNGCryptoServiceProvider key = new RNGCryptoServiceProvider())
        {
            byte[] b = new byte[4];
            key.GetNonZeroBytes(b);
            Random rng = new Random(1 * b[1] * b[2] * b[3]);
            //this happens 100*r % of the time, so do what happens 100*r % of the time.
            return (rng.NextDouble() < r);
        }
    }

    static void TamperWithBlock(ref object o)
    {
        using (RNGCryptoServiceProvider key = new RNGCryptoServiceProvider())
        {
            key.GetNonZeroBytes((o as byte[]));
        }
    }

    public double CBA
    {
        get
        {
            return (TotalBytesDownloaded == 0) ? 0 : (TotalBytesDownloaded * 1000) / (TotalTimeTa
ken * 1024); //converting to kilobytes per second.
        }
    }

    public double SCR
    {
        get
        {
            return (nSuccessful + 0.5) / (nAssigned + 1);
        }
    }


}

public class Block
{
    public string Assignee { get; set; }
    public string Link { get; private set; }
    public long Position { get; private set; }
    public long StartAddress { get; private set; }
    public long Size { get; private set; }
    public DateTime StartTime { get; private set; }
    private BlockStatus _status;
    public BlockStatus Status
    {
        get
        {
            return _status;
        }
        set
        {
            _status = value;
            if (value == BlockStatus.Downloading)
            { StartTime = DateTime.UtcNow; }
            else if (value == BlockStatus.NotDownloaded || value == BlockStatus.Downloaded)
            { OnStatusChanged(EventArgs.Empty); }
        }
    }
    public byte[] content;

    public event EventHandler BlockStatusChanged;

    void OnStatusChanged(EventArgs e)
    {
        // Make a temporary copy of the event to avoid possibility of
        // a race condition if the last subscriber unsubscribes
        // immediately after the null check and before the event is raised.
        // see https://stackoverflow.com/questions/1609430/copying-delegates
        EventHandler handler = BlockStatusChanged;
        if (handler != null) { handler(this, e); }
    }

    public void BlockInit(Uri url,long pos, long start, long blockSize)
    {
        Link = url.ToString();
        Position = pos;
        StartAddress = start;
        Size = blockSize;
    }
}

public enum BlockStatus {NotAssigned, Downloading, Downloaded, NotDownloaded}

public class PeerData
{
    //public string ID;
```

```
        public int NumberOfSessionsWith { get; set; }
        public long NumberOfBytes { get; set; }
        public long NumberOfFailedBlocks { get; set; }
        public long NumberOfTamperedBlocks { get; set; }
        public long NumberOfGoodBlocks { get; set; }
        public DateTime LastSuccessfulTime { get; set; }
        public double CurrentRI { get; set; }
        public double CurrentCNorm { get; set; }
    }

    public static class MyMethods
    {
        //Imitating Basic Table in Mathematica
        public static List<T> Table<T>(this T value, long count)
        {
            List<T> temp = new List<T>(1073741824);
            for (long i = 0; i < count; i++)
            {
                temp.Add(value);
            }
            return temp;
        }

        //http://stackoverflow.com/questions/1014005/how-to-populate-instantiate-a-c-sharp-array-with-
a-single-value
        public static void Populate<T>(this T[] arr, T value)
        {
            for (int i = 0; i < arr.Length; i++)
            {
                arr[i] = value;
            }
        }
    }
}

// Peer.CS

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ComponentModel;
using System.Threading;
using System.Net;
using System.Diagnostics;
using System.IO;
using System.Security.Cryptography;
using System.Xml.Linq;
using FRIENDS.Properties;

namespace FRIENDS
{
    class Peer
    {
        public string ID { get; private set; }
        //number of assigned blocks and number of successful blocks downloaded.
        //members, e.g. of class, are automatically assigned. Locals (of methods) aren't. rememeber...
        long nAssigned, nSuccessful, TotalBytesDownloaded; //sim=simulated.
        double simReliability, simBWFactor, simRiskFactor, TotalTimeTaken;//in milliseconds
        public int SessionCount { get; set; }
public BackgroundWorker BgWorker;
public static XElement PL;
public static Dictionary<string, Peer> PeerList;

public Peer()
{
   ID = Guid.NewGuid().ToString("n").Substring(0, 16);
   InitBgWorker();
   simReliability = SetReliability();
   simBWFactor= SetBandwidthFactor();
   simRiskFactor = SetRiskFactor();
}

public static void LoadPeerList()
{//try to get stored PeerList else return leaving PL null.
   if (Settings.Default.PeerList == "") return;
   PL = XElement.Parse(Settings.Default.PeerList);
   if (PL.Elements("Peer").Count() != 0)
   {
      PeerList = new Dictionary<string, Peer>();
      foreach (var l in PL.Elements("Peer"))
      {
         string s = l.Element("ID").Value;
         PeerList.Add(s, new Peer());
         PeerList[s].ID = s;
         PeerList[s].nAssigned = (long)l.Element("nAssigned");
         PeerList[s].nSuccessful = (long)l.Element("nSuccessful");
         PeerList[s].TotalBytesDownloaded = (long)l.Element("TotalBytesDownloaded");
         PeerList[s].TotalTimeTaken = (double)l.Element("TotalTimeTaken");
         PeerList[s].SessionCount = (int)l.Element("SessionCount");
         PeerList[s].simBWFactor = (double)l.Element("simBWFactor");
         PeerList[s].simReliability = (double)l.Element("simReliability");
         PeerList[s].simRiskFactor = (double)l.Element("simRiskFactor");
      }
   }
}

public static void CreatePeerList()
{
   PL = new XElement("PeerList",new XComment("List of Discovered Peers"));
   foreach (Peer p in PeerList.Values)
   {
      PL.Add(new XElement("Peer",
         new XElement("ID", p.ID),
         new XElement("nAssigned", p.nAssigned),
         new XElement("nSuccessful", p.nSuccessful),
         new XElement("TotalBytesDownloaded", p.TotalBytesDownloaded),
         new XElement("TotalTimeTaken", p.TotalTimeTaken),
         new XElement("SessionCount", p.SessionCount),
         new XElement("simBWFactor", p.simBWFactor),
         new XElement("simReliability", p.simReliability),
         new XElement("simRiskFactor", p.simRiskFactor)
         ));
   }
   Settings.Default.PeerList = PL.ToString();
   Settings.Default.Save();
}

public static void UpdatePLSettings(string id)
{
```

```
            foreach (var peer in PL.Elements("Peer"))
            {
                if (peer.Element("ID").Value == id)
                {
                    try
                    {
                        peer.ReplaceNodes(new XElement("ID", id),
                            new XElement("nAssigned", PeerList[id].nAssigned),
                    new XElement("nSuccessful", PeerList[id].nSuccessful),
                    new XElement("TotalBytesDownloaded", PeerList[id].TotalBytesDownloaded),
                    new XElement("TotalTimeTaken", PeerList[id].TotalTimeTaken),
                    new XElement("SessionCount", PeerList[id].SessionCount),
                    new XElement("simBWFactor", PeerList[id].simBWFactor),
                    new XElement("simReliability", PeerList[id].simReliability),
                    new XElement("simRiskFactor", PeerList[id].simRiskFactor)
                    );
                        Settings.Default.PeerList = PL.ToString();
                        Settings.Default.Save();
                    }
                    catch (Exception)
                    {
                        //Wait till next opportunity to try;
                    }
                }
            }
        }

        private static double SetBandwidthFactor()
        {
            using (RNGCryptoServiceProvider key = new RNGCryptoServiceProvider())
            {
                byte[] b = new byte[4];
                key.GetNonZeroBytes(b);
                Random rng = new Random(1 * b[1] * b[2] * b[3]);
                return (double)rng.Next(100, 1001) / 1000;
            }
        }

        private static double SetReliability()
        {
            using (RNGCryptoServiceProvider key = new RNGCryptoServiceProvider())
            {
                byte[] b = new byte[4];
                key.GetNonZeroBytes(b);
                Random rng = new Random(1 * b[1] * b[2] * b[3]);
                return (double)rng.Next(500, 1001) / 1000; //using bigger values for more space => better
randomness.
            }
        }

        private static double SetRiskFactor()
        {
            using (RNGCryptoServiceProvider key = new RNGCryptoServiceProvider())
            {
                byte[] b = new byte[4];
                key.GetNonZeroBytes(b);
                Random rng = new Random(1 * b[1] * b[2] * b[3]);
                return (double)rng.Next(0, 50) / 1000; //return is btw 0.000 and and 0.50.
            }
        }
```

```
    public void InitBgWorker()
    {
        if (this.simBWFactor == 0) this.BgWorker = new BackgroundWorker();
        this.BgWorker.WorkerReportsProgress = true;
        this.BgWorker.WorkerSupportsCancellation = true;
        BgWorker.DoWork += new DoWorkEventHandler(BgWorker_DoWork);
        BgWorker.RunWorkerCompleted                    +=                    new
RunWorkerCompletedEventHandler(BgWorker_RunWorkerCompleted);
        BgWorker.ProgressChanged                    +=                    new
ProgressChangedEventHandler(BgWorker_ProgressChanged);
    }
    object[] DownloadBlock(Block block, BackgroundWorker worker, DoWorkEventArgs e)
    {
        object[] obj = new object[2];
        obj[0] = block.Position;
        if (worker.CancellationPending)
        {
            e.Cancel = true;
        }
        else
        {
            try
            {
                Thread.Sleep(2000);
                HttpWebRequest                    myHttpWebRequest                    =
(HttpWebRequest)WebRequest.Create(block.Link);
                myHttpWebRequest.AddRange(block.StartAddress, block.StartAddress + block.Size - 1);
                myHttpWebRequest.Timeout = 30000;
                /*Debug.WriteLine("Call AddRange(50,150)");
                Debug.Write("Resulting Request Headers: ");
                Debug.WriteLine(myHttpWebRequest.Headers.ToString());*/
                using          (HttpWebResponse          myHttpWebResponse          =
(HttpWebResponse)myHttpWebRequest.GetResponse())
                /*Debug.Write("Resulting Response Headers: ");
                Debug.WriteLine(myHttpWebResponse.Headers.ToString());*/
                using (Stream streamResponse = myHttpWebResponse.GetResponseStream())
                using (MemoryStream ms = new MemoryStream())
                {
                    streamResponse.CopyTo(ms, 16384);
                    obj[1] = ms.ToArray();
                    myHttpWebResponse.Close();
                }
            }
            catch (Exception ex)
            {
                //notify peer of error
                Debug.WriteLine(ex.Message + " " + block.Position);
                //Thread.Sleep(2000);
                // e.Cancel = true;
                //MainForm.BlockList[block.Position].status = BlockStatus.NotDownloaded;
            }
        }
        return obj;
    }
    void BgWorker_DoWork(object sender, DoWorkEventArgs e)
    {
        // Get the BackgroundWorker that raised this event.
        BackgroundWorker worker = sender as BackgroundWorker;
        // Assign the result of the computation to the Result property of the DoWorkEventArgs object.
This will be available to the RunWorkerCompleted eventhandler.

        e.Result = DownloadBlock((Block)e.Argument, worker, e);
      }

      void BgWorker_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
      {
        nAssigned += 1;
        if (e.Cancelled)
        { /*echo to peer: "You cancelled the download"*/
          return;
        }
        object[] obj = e.Result as object[];
        long n = (long)obj[0];
        int sleep = (int)(((DateTime.UtcNow - MainForm.BlockList[n].StartTime).Milliseconds /
simBWFactor) * (1 - simBWFactor));
        Thread.Sleep(sleep);
        Debug.WriteLine("Position " + n);
        if (MainForm.BlockList[n].Assignee == ID)
        {
          if (e.Error != null || obj[1] == null || SimulateFailure(simReliability))
          {
            /*echo to peer: "Download failed"*/
            MainForm.BlockList[n].status = BlockStatus.NotDownloaded;
          }
          else
          {
            if (SimulateTampering(simRiskFactor)) { TamperWithBlock(ref obj[1]); }
            MainForm.BlockList[n].content = obj[1] as byte[];
            MainForm.BlockList[n].status = BlockStatus.Downloaded;
            Debug.WriteLine("Position " + n + " Completed");
            nSuccessful += 1;
            TotalBytesDownloaded += (obj[1] as byte[]).Length;
            TotalTimeTaken              +=              ((DateTime.UtcNow              -
MainForm.BlockList[n].StartTime).Milliseconds);
          }
        }
        else { /*echo to peer: "You took so long. Block has been reassigned"*/}
      }

      void BgWorker_ProgressChanged(object sender, ProgressChangedEventArgs e)
      {

      }

      private static bool SimulateFailure(double s)
      {
        using (RNGCryptoServiceProvider key = new RNGCryptoServiceProvider())
        {
          byte[] b = new byte[4];
          key.GetNonZeroBytes(b);
          Random rng = new Random(1 * b[1] * b[2] * b[3]);
          //reliability happens 100*s % of the time, so failure happens the rest of the time.
          return !(rng.NextDouble() < s);
        }
      }

      private static bool SimulateTampering(double r)
      {
            using (RNGCryptoServiceProvider key = new RNGCryptoServiceProvider())
            {
                byte[] b = new byte[4];
                key.GetNonZeroBytes(b);
                Random rng = new Random(1 * b[1] * b[2] * b[3]);
                //this happens 100*r % of the time, so do what happens 100*r % of the time.
                return (rng.NextDouble() < r);
            }
        }

        static void TamperWithBlock(ref object o)
        {
            using (RNGCryptoServiceProvider key = new RNGCryptoServiceProvider())
            {
                key.GetNonZeroBytes((o as byte[]));
            }
        }

        public double CBA
        {
            get
            {
                return (TotalBytesDownloaded == 0) ? 0 : (TotalBytesDownloaded * 1000) /
(TotalTimeTaken * 1024); //converting to kilobytes per second.
            }
        }

        public double SCR
        {
            get
            {
                return (nAssigned == 0) ? 0.5 : Math.Sqrt((0.25 + Math.Max(0, 3 * nSuccessful - 2 *
nAssigned)) / (nAssigned + 1));
            }
        }


    }

    public class Block
    {
        public string Assignee { get; set; }
        public string Link { get; private set; }
        public long Position { get; private set; }
        public long StartAddress { get; private set; }
        public long Size { get; private set; }
        public DateTime StartTime { get; private set; }
        private BlockStatus _status;
        public BlockStatus status
        {
            get
            {
                return _status;
            }
            set
            {
                _status = value;
                if (value == BlockStatus.Downloading) this.StartTime = DateTime.UtcNow;
                if (value == BlockStatus.NotDownloaded || value == BlockStatus.Downloaded)
                { OnStatusChanged(EventArgs.Empty); }
            }
        }
        public byte[] content;

        public event EventHandler BlockStatusChanged;

        void OnStatusChanged(EventArgs e)
        {
            // Make a temporary copy of the event to avoid possibility of
            // a race condition if the last subscriber unsubscribes
            // immediately after the null check and before the event is raised.
            // see https://stackoverflow.com/questions/1609430/copying-delegates
            EventHandler handler = BlockStatusChanged;
            if (handler != null) { handler(this, e); }
        }

        public void BlockInit(Uri url,long pos, long start, long blockSize)
        {
            Link = url.ToString();
            Position = pos;
            StartAddress = start;
            Size = blockSize;
        }
    }

    public enum BlockStatus {NotAssigned, Downloading, Downloaded, NotDownloaded}

    public class PeerData
    {
        //public string ID;
        public int NumberOfSessionsWith { get; set; }
        public long NumberOfBytes { get; set; }
        public long NumberOfFailedBlocks { get; set; }
        public long NumberOfTamperedBlocks { get; set; }
        public long NumberOfGoodBlocks { get; set; }
        public DateTime LastSuccessfulTime { get; set; }
        public double CurrentRI { get; set; }
        public double CurrentCNorm { get; set; }
    }
}

// Form.CS

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Security.Cryptography;
using System.IO;
using System.Diagnostics;
using System.Xml.Linq;
using FRIENDS.Properties;
using System.Threading;
using System.Media;
```

```
namespace FRIENDS
{
  public partial class MainForm : Form
  {
    public MainForm()
    {
      InitializeComponent();
      //Application.Idle += new System.EventHandler(this.RunTests);
    }

    private void RunTests(object sender, EventArgs e)
    {
      if (FileSize > 0 && OpenButton.Enabled) MessageBox.Show("Idle", "Yeah!!!");
    }


    Dictionary<string, PeerData> KnownPeerStore;
    Dictionary<string, bool> SessionPeers;
    static Dictionary<string, double> TrustRating;
    string[] PeerCDGroup;
    double WCBA, WSCR, WRI, CMax, CMin;
    Uri DownloadLink;
    string FileName, Method;
    long FileSize, MaxPeers = 3;
    long CompletedBlocks = 0, BlockSize = 400 * 1024;
    static public Block[] BlockList;
    byte[] FileAsByteArray, CheckFileAsByteArray; //consider using a file stream afterwards
    long[][] FileCheck;
    bool DownloadCancelled;
    XElement KPS;
    Stopwatch swc;

    private void Slider_Scroll(object sender, EventArgs e)
    {
      string txtName = (sender as TrackBar).Name.Replace("Slider", "Txt");
      (sender as Control).Parent.Controls[txtName].Text = Convert.ToString(((double)(sender as
TrackBar).Value) / 20);
      //this.Controls[txtName].Text = Convert.ToString(((double)(sender as TrackBar).Value) / 20);
    }

    private void InitialiseSliders()
    {
      SCRSlider.Value = (int)Math.Round(Double.Parse(SCRTxt.Text) * 20);
      CBASlider.Value = (int)Math.Round(Double.Parse(CBATxt.Text) * 20);
      RISlider.Value = (int)Math.Round(Double.Parse(RITxt.Text) * 20);
    }

    private void maskedTextBox_TextChanged(object sender, EventArgs e)
    {
      //MaskedTextBox mtxt = sender as MaskedTextBox;
      if (Double.Parse((sender as MaskedTextBox).Text) > 1.00) (sender as MaskedTextBox).Text =
"1.00";
      string sliderName = (sender as Control).Name.Replace("Txt", "Slider");
      ((sender as Control).Parent.Controls[sliderName] as TrackBar).Value =
(int)Math.Round(Double.Parse((sender as MaskedTextBox).Text) * 20);
      NormaliseWeights();
      RatePeers();
      RefreshDisplayBox();
    }
    private void NormaliseWeights()
    {
      double s = double.Parse(SCRTxt.Text);
      double c = double.Parse(CBATxt.Text);
      double r = double.Parse(RITxt.Text);
      WSCR = s / (s + c + r);
      WCBA = c / (s + c + r);
      WRI = r / (s + c + r);
      nwSCR.Text = WSCR.ToString("F4");
      nwCBA.Text = WCBA.ToString("F4");
      nwRI.Text = WRI.ToString("F4");
    }

    void RatePeers()
    {
      CMax = Peer.PeerList.Values.Max(c => c.CBA) * 4 / 3;
      CMin = (CMax == 0) ? 0 : Peer.PeerList.Values.Min(c => c.CBA) * 2 / 3;
      Dictionary<string, double> TR = new Dictionary<string, double>();
      foreach (Peer p in Peer.PeerList.Values)
      {
        double CNorm = (p.CBA == 0) ? 0.5 : (p.CBA - CMin) / (CMax - CMin);
        KnownPeerStore[p.ID].CurrentCNorm = CNorm;
        PeerData pr = KnownPeerStore[p.ID];
        double RI = (pr.NumberOfGoodBlocks + pr.NumberOfTamperedBlocks == 0) ? 0.5 :
(double)(pr.NumberOfGoodBlocks - pr.NumberOfTamperedBlocks) / (pr.NumberOfGoodBlocks +
pr.NumberOfTamperedBlocks);
        if (RI < 0) RI = 0;
        KnownPeerStore[p.ID].CurrentRI = RI;
        double t = (CNorm * WCBA) + (p.SCR * WSCR) + (RI * WRI);
        TR.Add(p.ID, t);
      }
      TrustRating = TR.OrderByDescending(key => key.Value).ToDictionary(k => k.Key, v =>
v.Value);
    }

    bool GetFileNameAndSize()
    {
      if (!Uri.TryCreate(AddressBox.Text, UriKind.Absolute, out DownloadLink) ||
(DownloadLink.Scheme != Uri.UriSchemeHttp && DownloadLink.Scheme != Uri.UriSchemeHttps))
      {
        MessageBox.Show("Error in url. Please input a valid http or https url.", "Error!",
MessageBoxButtons.OK, MessageBoxIcon.Error); return false;
      }
      HttpWebRequest req = (HttpWebRequest)WebRequest.Create(DownloadLink);
      req.Method = "HEAD";
      HttpWebResponse resp = (HttpWebResponse)(req.GetResponse());
      if (!resp.Headers.AllKeys.Contains("Accept-Ranges") || !resp.Headers["Accept-
Ranges"].Contains("bytes"))
      {
        MessageBox.Show("The specified server does not support byte range requests, sorry.",
"Error!", MessageBoxButtons.OK, MessageBoxIcon.Error); return false;
      }
      if (resp.ContentLength <= 400 * 1024)
      {
        MessageBox.Show("The specified file is less than 400KB. Please download it yourself",
"Error!", MessageBoxButtons.OK, MessageBoxIcon.Error); return false;
      }
      FileSize = resp.ContentLength;
      if (resp.Headers.AllKeys.Contains("content-disposition") && resp.Headers["content-
disposition"].ToLower().Contains("filename="))
```

```
        {
           FileName                    =@"C:\Friends\"                    +                    new
System.Net.Mime.ContentDisposition(resp.Headers["content-disposition"]).FileName;
        }
        else
        {
           int i = DownloadLink.AbsolutePath.LastIndexOf('/') + 1;
           FileName = @"C:\Friends\" + DownloadLink.AbsolutePath.Substring(i);
        }
        FileName = Uri.UnescapeDataString(FileName);
        string sb = FileName.Substring(11);
        CheckFileAsByteArray = File.ReadAllBytes(@"C:\Friends\Check\mp3s\" + sb);//in real life
we won't use this..It is only used to avoid Error 429 in the simulation.
        int n = 1;
        while (File.Exists(FileName))
        {
           FileName = string.Format(@"C:\Friends\{0}_{1}", n++, sb);
        }
        System.IO.Directory.CreateDirectory(@"C:\Friends");
        //MessageBox.Show("Check   Succeeded.   You   are   good   to   go!",   "Success",
MessageBoxButtons.OK);
        return true;
     }

     void FormPeerGroup()
     {
        PeerCDGroup = new string[MaxPeers];
        switch (Method)
        {
           case "Random":
              var rng = new Random();
              int c = -1;
              PeerSelectBox.ClearSelected();
              for (int r = 0; r < MaxPeers; r++)
              {
                 while (c == -1 || PeerSelectBox.GetSelected(c))
                 {
                    c = rng.Next(0, PeerSelectBox.Items.Count);
                 }
                 PeerSelectBox.SetSelected(c, true);
                 PeerCDGroup[r] = PeerSelectBox.Items[c].ToString();
              }
              break;
           case "Select":
              for (int s = 0; s < PeerSelectBox.SelectedItems.Count; s++)
              {
                 PeerCDGroup[s] = PeerSelectBox.SelectedItems[s].ToString();
              }
              break;
           case "Trust": //just to show that Trust in the default option..
           default:
              int n = 0;
              foreach (var s in SessionPeers.Keys)
              {
                 if (SessionPeers[s]) PeerCDGroup.SetValue(s, n++);
              }
              RatePeers();
              var tmp = TrustRating.Keys.ToArray();
              for (int i = 0; i < MaxPeers - n; i++)
              {
                PeerCDGroup[n + i] = tmp[i];
            }
            break;
        }
    }

    void AssignBlocks()
    {
        Peer p;
        int n = 0;
        double pCBAF; //peerCBAFactor; used to compute p.CBA * 1024, i.e CBA in bytes/s;
        foreach (string s in PeerCDGroup)
        {
            if (s == null) continue;
            p = Peer.PeerList[s];
            if (n == -1) return;
            if (!p.BgWorker.IsBusy)
            {
                pCBAF = p.CBA * 1024;//blocksize/pCBAF = maximum time spendable for block
download;
                n = Array.FindIndex(BlockList, b => b.status == BlockStatus.NotAssigned || (b.status ==
BlockStatus.Downloading && (DateTime.UtcNow - b.StartTime).Seconds - 2000 > b.Size / pCBAF));
                if (n > -1)
                {
                    FileIntegrityCheck(BlockList[n]);
                    BlockList[n].status = BlockStatus.Downloading;
                    BlockList[n].Assignee = p.ID;
                    p.BgWorker.RunWorkerAsync(BlockList[n]);
                    if (!SessionPeers.ContainsKey(p.ID))
                    {
                        KnownPeerStore[p.ID].NumberOfSessionsWith += 1;
                        Peer.PeerList[p.ID].SessionCount += 1;
                    }
                    SessionPeers[p.ID] = true;
                }
            }
        }
    }

    static void SimulateWLanDiscovery()
    {
        Peer.LoadPeerList();
        if (Peer.PeerList == null)
        {
            Peer.PeerList = new Dictionary<string, Peer>();
            for (int i = 0; i < 10; i++) //10 peers
            {
                Peer p = new Peer();
                while (Peer.PeerList.Keys.Contains(p.ID)) { p = new Peer(); }
                Peer.PeerList.Add(p.ID, p);
            }
            Peer.CreatePeerList();
        }
    }

    void InitializePeerStore()
    {
        LoadPeerStore();
        if (KnownPeerStore == null)
        {
            KnownPeerStore = new Dictionary<string, PeerData>();
            foreach (Peer peer in Peer.PeerList.Values)
            {
              KnownPeerStore.Add(peer.ID, new PeerData());
            }
          }
          else
          {
            foreach (Peer p in Peer.PeerList.Values)
            {
              if   (!KnownPeerStore.Keys.Contains(p.ID))   {   KnownPeerStore.Add(p.ID,   new
PeerData()); }
            }
          }
          KPS = new XElement("KnownPeers", new XComment("List of Known Peers"));
          foreach (var s in KnownPeerStore)
          {
            KPS.Add(new XElement("Peer",
              new XElement("ID", s.Key),
              new XElement("CurrentCNorm", s.Value.CurrentCNorm),
              new XElement("CurrentRI", s.Value.CurrentRI),
              new XElement("LastSuccessfulTime", s.Value.LastSuccessfulTime),
              new XElement("NumberOfBytes", s.Value.NumberOfBytes),
              new XElement("NumberOfFailedBlocks", s.Value.NumberOfFailedBlocks),
              new XElement("NumberOfGoodBlocks", s.Value.NumberOfGoodBlocks),
              new XElement("NumberOfSessionsWith", s.Value.NumberOfSessionsWith),
              new XElement("NumberOfTamperedBlocks", s.Value.NumberOfTamperedBlocks)
              ));
          }
          Settings.Default.KnownPeerStore = KPS.ToString();
          Settings.Default.Save();
          PeerComboBox.Items.AddRange(KnownPeerStore.Keys.ToArray());
          PeerSelectBox.Items.AddRange(KnownPeerStore.Keys.ToArray());

        }

        void LoadPeerStore()
        {//try to get stored Known Peers else return leaving KPS null
          if (Settings.Default.KnownPeerStore == "") return;
          KPS = XElement.Parse(Settings.Default.KnownPeerStore);
          if (KPS.Elements("Peer").Count() != 0)
          {
            KnownPeerStore = new Dictionary<string, PeerData>();
            foreach (var k in KPS.Elements("Peer"))
            {
              string s = k.Element("ID").Value;
              KnownPeerStore.Add(s, new PeerData());
              KnownPeerStore[s].CurrentCNorm = (double)k.Element("CurrentCNorm");
              KnownPeerStore[s].CurrentRI = (double)k.Element("CurrentRI");
              KnownPeerStore[s].LastSuccessfulTime = (DateTime)k.Element("LastSuccessfulTime");
              KnownPeerStore[s].NumberOfBytes = (long)k.Element("NumberOfBytes");
              KnownPeerStore[s].NumberOfFailedBlocks                                    =
(long)k.Element("NumberOfFailedBlocks");
              KnownPeerStore[s].NumberOfGoodBlocks                                      =
(long)k.Element("NumberOfGoodBlocks");
              KnownPeerStore[s].NumberOfSessionsWith                                    =
(int)k.Element("NumberOfSessionsWith");
              KnownPeerStore[s].NumberOfTamperedBlocks                                  =
(long)k.Element("NumberOfTamperedBlocks");
            }
            }
        }

        void InitializeBlockList()
        {
            FileAsByteArray = new byte[FileSize];
            long div = FileSize / BlockSize;
            long count = (FileSize % BlockSize < BlockSize / 2) ? div : div + 1;
            BlockList = new Block[count];
            FileCheck = new long[count][];
            for (int i = 0; i < count-1; i++)
            {
                BlockList[i] = new Block();
                BlockList[i].BlockInit(DownloadLink, i, i * BlockSize, BlockSize);
                BlockList[i].BlockStatusChanged+=new    EventHandler(Block_StatusChanged);    //C#1.0
syntax
            }
            //could have checked from end in the for loop rather than doing this, but it's computationally
wasteful.
            long rem =  (count - 1) * BlockSize;
            BlockList[count - 1] = new Block();
            BlockList[count - 1].BlockInit(DownloadLink,count - 1, rem, FileSize - rem);
            BlockList[count  -  1].BlockStatusChanged  +=  Block_StatusChanged;  //C#2.0  syntax.
equivalent to 1.0
        }

        void Block_StatusChanged(object sender, EventArgs e)
        {
            Block bl = (Block)sender;
            if (bl.status == BlockStatus.Downloading || bl.status == BlockStatus.NotAssigned) return;
            if (bl.status == BlockStatus.NotDownloaded || !VerifyDownload(bl))
            {
                if (bl.status == BlockStatus.NotDownloaded)
                { KnownPeerStore[bl.Assignee].NumberOfFailedBlocks += 1; }
                BlockList[bl.Position].status = BlockStatus.NotAssigned;
            }
            else if (!DownloadCancelled)
            {
                Array.Copy(bl.content, 0, FileAsByteArray, bl.StartAddress, bl.content.LongLength);
                CompletedBlocks += 1;
                DProgressBar.Value = (int)(CompletedBlocks * 100  / BlockList.LongLength);
                DProgressBar.ToolTipText  =  string.Format("Download  Progress:  {0}%  Completed",
DProgressBar.Value);
            }
            SessionPeers[bl.Assignee] = false;
            RatePeers();
            UpdateKPSSettings(bl.Assignee);
            Peer.UpdatePLSettings(bl.Assignee);
            if (CompletedBlocks != BlockList.LongLength)
            {
                if (Method == "Trust" && AdaptiveButton.Checked) FormPeerGroup();
                else AssignBlocks();
            }
            else
            {
                swc.Stop();
                File.WriteAllBytes(FileName, FileAsByteArray);
                statusLbl.Text     =     string.Format("Download     Completed     in     {0}     ms",
swc.ElapsedMilliseconds);
                OpenButton.Enabled = true;
                SystemSounds.Beep.Play();
                ResetForm();
            }
        }

        bool VerifyDownload(Block b)
        {
            if (b.content == null || b.content.LongLength != b.Size)
            {
                KnownPeerStore[b.Assignee].NumberOfTamperedBlocks += 1;
                return false;
            }
            foreach (long index in FileCheck[b.Position])
            {
                for (int i = 0; i < 2048; i++)
                {
                    if (CheckFileAsByteArray[index + i] != b.content[index - b.StartAddress + i])
                    {
                        KnownPeerStore[b.Assignee].NumberOfTamperedBlocks += 1;
                        return false;
                    }
                }
            }
            KnownPeerStore[b.Assignee].NumberOfGoodBlocks += 1;
            KnownPeerStore[b.Assignee].NumberOfBytes += b.content.LongLength;
            KnownPeerStore[b.Assignee].LastSuccessfulTime = DateTime.UtcNow;
            return true;
        }

        void FileIntegrityCheck(Block a)
        {
            int seed = 1;
            List<long> indices = new List<long>();
            using (RNGCryptoServiceProvider key = new RNGCryptoServiceProvider())
            {
                byte[] b = new byte[1];
                key.GetNonZeroBytes(b);
                Random rng = new Random(b[0]);
                b = new byte[rng.Next(5, 9)];//b can be 8 - 15 'bytes' (array length) long.
                for (int i = 0; i < a.Size; i += 40960)//40KB
                {
                    key.GetNonZeroBytes(b);
                    int mod = (a.Size - i >= 40960) ? 38913 : (int)a.Size - i - 2047;
                    //you can later modify this to select the last 5KB in the block.. duplicate downloads with
the last byte possible
                    if (mod < 128) break; //So currently if last piece of block is less than 383 (255+128) dont
download/check that part.
                    //e.g. if 5KB, 4864= 5*1024 - 256, max start index for a 256byte piece in a 5KB (5*1024)
block; 4865 = 4864 + 1 to do mod so you can also get 4864; 4865= 5*1024-255
                    seed = Math.Abs(b.Aggregate(1, (x, y) => x * y)) % mod;
                    indices.Add(a.StartAddress + i + seed);
                }
            }
            FileCheck[a.Position] = indices.ToArray();

            //Code below not used because of the high possibility of error 429 (too many requests) from
webserver. It is left because it will work in real life scenarios with actually different phones.
           /* while (this.InitiatorWorker.IsBusy)
            {
```

```
        //while waiting for the backgroundWorker to be free, inform the iniatiator and keep UI
reponsive
            statusLbl.Text += "\nOne more block check download task pooled.";
            Application.DoEvents();
        } */
        /* or use this instead of above to search only if the chance is there:
        if (!InitiatorWorker.IsBusy)
        {
            InitiatorWorker.RunWorkerAsync(FileCheck[a.Position]);
            statusLbl.Text = "Block check download in background.";
        } */

        //so this is used instead in the simulation: a CheckFileAsByteArray which is the original file
predownloaded to serve as a check. The reason is the Error 429 that will be received on trying to start
so many requests to the server from the same application.
    }

    object DownloadPiece(long start, BackgroundWorker worker, DoWorkEventArgs e)
    {
        object obj = new object();
        if (worker.CancellationPending)
        {
            e.Cancel = true;
        }
        else
        {
            try
            {
                HttpWebRequest                    myHttpWebRequest                    =
(HttpWebRequest)WebRequest.Create(DownloadLink);
                myHttpWebRequest.AddRange(start, start + 2047);
                HttpWebResponse                   myHttpWebResponse                   =
(HttpWebResponse)myHttpWebRequest.GetResponse();
                using (Stream streamResponse = myHttpWebResponse.GetResponseStream())
                using (MemoryStream ms = new MemoryStream())
                {
                    //we know the stream response should be 256 bytes long.
                    // default buffer is 4096 bytes. That's wastage for us here!
                    //so in copying, we use a buffer = 2*256 + 8 extra bytes (to avoid any unanticipated
issues). 2*256 + 8 = 520. OK. no magic number here. lol.
                    streamResponse.CopyTo(ms);
                    obj = ms.ToArray();
                }
            }
            catch (Exception)
            {
                e.Cancel = true;
            }
        }
        return obj;
    }

    private void InitiatorWorker_DoWork(object sender, DoWorkEventArgs e)
    {
        BackgroundWorker worker = sender as BackgroundWorker;
        long[] aIndices = (long[]) e.Argument;
        object[] aResult = new object[aIndices.Length + 1];
        aResult[0] = aIndices;
        for (int i = 0; i < aIndices.Length; i++)
        {
            aResult[i + 1] = DownloadPiece(aIndices[i], worker, e);
            Thread.Sleep(2000);
        }
        e.Result = aResult;
    }

    private void InitiatorWorker_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
    {
        if (statusLbl.Text.Contains("Block")) statusLbl.Text = "";
        if (e.Error != null)
        {
        }
        else if (e.Cancelled)
        {
        }
        else
        {
            object[] obj = e.Result as object[];
            for (int i = 0; i < obj.Length - 1; i++)
            {
                Array.Copy(obj[i + 1] as byte[], 0, FileAsByteArray, (obj[0] as long[])[i], (obj[i + 1] as
byte[]).Length);
            }
        }

    }

    private void ResetForm()
    {
        swc.Stop();
        CheckButton.Enabled = true;
        DownloadButton.Enabled = CancelDButton.Enabled = false;
        CompletedBlocks = 0;
        DProgressBar.Value = 0;
        DProgressBar.Visible = false;
        DProgressBar.ToolTipText = "Download Progress";
        RefreshDisplayBox();
        foreach (Peer p in Peer.PeerList.Values)
        {
            p.BgWorker.Dispose();
            p.BgWorker = null;
            p.BgWorker = new BackgroundWorker();
            p.InitBgWorker();
        }
        // will be necessary in real life situation where we have to actually download the checks.
        /*
        this.InitiatorWorker.Dispose();
        this.InitiatorWorker = new BackgroundWorker();
        this.InitiatorWorker.WorkerReportsProgress = true;
        this.InitiatorWorker.WorkerSupportsCancellation = true;
        this.InitiatorWorker.DoWork += new DoWorkEventHandler(this.InitiatorWorker_DoWork);
        this.InitiatorWorker.RunWorkerCompleted += new
RunWorkerCompletedEventHandler(this.InitiatorWorker_RunWorkerCompleted); */
    }

    private void DownloadButton_Click(object sender, EventArgs e)
    {
        DownloadCancelled = false;
        CheckButton.Enabled = OpenButton.Enabled = DownloadButton.Enabled = false;
        CancelDButton.Enabled = true;
        SessionPeers = new Dictionary<string, bool>();
        statusLbl.Text = "Downloading in progress...";
        DProgressBar.Visible = true;
        swc = Stopwatch.StartNew();
        FormPeerGroup();
        AssignBlocks();

    }

    private void MainForm_Load(object sender, EventArgs e)
    {
        this.Size = Settings.Default.MainFormSize;
        InitialiseSliders();
        SimulateWLanDiscovery();
        InitializePeerStore();
        NormaliseWeights();
        RatePeers();
        PeerSelectBox.SelectedIndex = PeerComboBox.SelectedIndex = 0;
    }

    private void CancelButton_Click(object sender, EventArgs e)
    {
        DownloadCancelled = true;
        statusLbl.Text = "You cancelled the download";
        ResetForm();
    }

    void radioButton_CheckedChanged(object sender, EventArgs e)
    {
        RadioButton rb = sender as RadioButton;
        Method = rb.Text;
        switch (Method)
        {
            case "Random":
            case "Trust": int n;
                MaxPeers = int.TryParse(MaxPeersTxt.Text, out n) ? Math.Min(n, Math.Min(FileSize /
BlockSize, KnownPeerStore.Count)) : 3;
                if (MaxPeers == 0) MaxPeers = 3;
                selPeers.Enabled = false;
                PeerSelectBox.Enabled = false;
                break;
            case "Select": MaxPeers = Math.Min(FileSize / BlockSize, KnownPeerStore.Count);
                selPeers.Text = string.Format("Select Peers below. Max = {0}", MaxPeers);
                selPeers.Enabled = true;
                PeerSelectBox.Enabled = true;
                break;
        }
        statusLbl.Text = string.Format("Number of peers being used: {0}", MaxPeers);
    }

    private void BlockSizeBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        BlockSize = (long)BlockSizeBox.SelectedItem * 1024;
    }

    private void PeerComboBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        RefreshDisplayBox();
    }
    private void RefreshDisplayBox()
    {
        RatePeers();
        string s = PeerComboBox.SelectedItem.ToString();
        DisplayBox.ResetText();
        DisplayBox.AppendText("OVERALL TRUST RATING:\t\t" + TrustRating[s].ToString("F4")
+ "\t(as per weights selected)\n");
        DisplayBox.AppendText("CBA:\t\t\t\t" + Peer.PeerList[s].CBA.ToString("F4") + "\n");
        DisplayBox.AppendText("CBA Normalised:\t\t\t" +
KnownPeerStore[s].CurrentCNorm.ToString("F4") + "\n");
        DisplayBox.AppendText("SCR:\t\t\t\t" + Peer.PeerList[s].SCR.ToString("F4") + "\n");
        DisplayBox.AppendText("1-RI (positive):\t\t\t" +
KnownPeerStore[s].CurrentRI.ToString("F4") + "\n");
        DisplayBox.AppendText("Last Successful Time:\t\t" +
((KnownPeerStore[s].LastSuccessfulTime.Year<2015)?"--------
\n":KnownPeerStore[s].LastSuccessfulTime.ToString("ddd, MMM. d, yyyy h:mm tt") + " (UTC)\n"));
        DisplayBox.AppendText("Total Bytes Downloaded:\t\t" +
KnownPeerStore[s].NumberOfBytes + "\n");
        DisplayBox.AppendText("Number of Sessions With:\t\t" +
KnownPeerStore[s].NumberOfSessionsWith + "\n");
        DisplayBox.AppendText("Number of Successful Blocks:\t\t" +
KnownPeerStore[s].NumberOfGoodBlocks + "\n");
        DisplayBox.AppendText("Number of failed Blocks:\t\t" +
KnownPeerStore[s].NumberOfFailedBlocks + "\n");
        DisplayBox.AppendText("Number of Blocks Tampered With:\t" +
KnownPeerStore[s].NumberOfTamperedBlocks + "\n");
        DisplayBox.AppendText("Total Number of Sessions for Peer:\t" +
Peer.PeerList[s].SessionCount + "\n");
    }

    private void OpenButton_Click(object sender, EventArgs e)
    {
        Process.Start("explorer.exe", "/select," + FileName);
    }

    void UpdateKPSSettings(string id)
    {
        foreach (var peer in KPS.Elements("Peer"))
        {
            if (peer.Element("ID").Value == id)
            {
                try
                {
                  peer.ReplaceNodes(new XElement("ID", id),
                new XElement("CurrentCNorm", KnownPeerStore[id].CurrentCNorm),
                new XElement("CurrentRI", KnownPeerStore[id].CurrentRI),
                new XElement("LastSuccessfulTime", KnownPeerStore[id].LastSuccessfulTime),
                new XElement("NumberOfBytes", KnownPeerStore[id].NumberOfBytes),
                new XElement("NumberOfFailedBlocks", KnownPeerStore[id].NumberOfFailedBlocks),
                new XElement("NumberOfGoodBlocks", KnownPeerStore[id].NumberOfGoodBlocks),
                new XElement("NumberOfSessionsWith",
KnownPeerStore[id].NumberOfSessionsWith),
                new XElement("NumberOfTamperedBlocks",
KnownPeerStore[id].NumberOfTamperedBlocks)
                );
                  Settings.Default.KnownPeerStore = KPS.ToString();
                  Settings.Default.Save();
                }
                catch (Exception)
```

```
                {
                    //Wait_till_next_opportunity;
                }
            }
        }
    }

    private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
    {
        foreach (string s in KnownPeerStore.Keys)
        {
            UpdateKPSSettings(s);
            Peer.UpdatePLSettings(s);
        }
        Settings.Default.MainFormSize = this.Size;
        Settings.Default.Save();
    }

    private void CheckButton_Click(object sender, EventArgs e)
    {
        if (GetFileNameAndSize()) DownloadButton.Enabled = true;
        InitializeBlockList();
    }

  }
}
```