University of Reading

Asynchronous Epidemic Algorithms for Consistency in Large-scale Systems

Mosab M Ayiad

January 2020
Reading, UK

University of Reading

Asynchronous Epidemic Algorithms for Consistency in Large-scale Systems

By:
Mosab M Ayiad

Supervisor:
Dr Giuseppe Di Fatta

THESIS

Submitted in partial fulfilment of the requirement for the degree of Doctor of Philosophy in the Department of Computer Science at the school of Mathematical, Physical and Computational Sciences (SMPCS)

January 2020
Reading, UK

Declaration

I, *Mosab M. Ayiad*, declare that this thesis titled, "*Asynchronous Epidemic Algorithms for Consistency in Large-scale Systems*" and the work presented in it is done wholly and mainly while in candidature for a research degree at the University of Reading, UK. I confirm that this is my own work and the use of all material from other sources has been properly and fully acknowledged. Also, parts of the work presented in this thesis have been published in peer-reviewed conferences.

Mosab M. Ayiad.

Acknowledgements

In the name of ALLAH, the Most Gracious and the Most Merciful. Alhamdulillah, all praises to ALLAH for His blessing, guidance, and vast giving in conducting this research work and completing this thesis. Also, I am sending the blessing and the peace upon His prophet MUHAMMAD, the mercy and the teacher for humanity.

The accomplishment of this research work and thesis would not have been possible without the help and support of many people and institutions who contributed mainly or subsidiary. Through their continuous and valuable assistance, the carryout and the completion of this PhD became a reality.

Firstly, I would like to express my sincere gratitude, appreciation and thanks to my supervisor Dr *Giuseppe Di Fatta* for the support of my study and research, for his motivation and broad knowledge. I appreciate him for empowering my research abilities and for assisting me in developing as a research scientist. Also, I much value his encouragement and guidance in carrying out publications which were not possible without his contribution and helpful feedback. Surely, Dr *Giuseppe Di Fatta* support was a substantial foundation for achieving this PhD.

I am also thankful to the monitoring committee: Prof *Xia Hong* and Dr *Frederic Stahl* for their reassurance, insightful comments, and great support.

An exceptional thanking goes out to my friends and study mates: Dr *Amogh katti* and Dr *Pasu Poonpakdee*. I am very grateful for their fabulous researches and their special support which have assisted me in embarking investigation and exploration work during the literature review stage. I am also thankful to them for their precious advice and feedback on my research work.

I gratefully acknowledge and much appreciate the study scholarship received towards my PhD from the Islamic Development Bank-Merit Scholarship Programme for High Technology (*IsDB-MSP*)[1]. The award has enabled the complete concentration on research and PhD activities. Although *IsDB-MSP* has funded the PhD study, they have no involvement in the research precise theme or topic.

I am also grateful to colleagues and staff at Al Azhar University-Gaza for their unfailing support and assistance before and during the sabbatical leave for the PhD.

I am thankful to everyone who supported and helped me during my PhD study.

[1]IsDB. *Islamic Development Bank*. [Online; accessed 5-August-2018]. 2018. URL: `https://www.isdb.org/`.

Dedications

To my Father Prof *Mostafa A. Ayiad*, despite his regretting absence since 13 years ago (passed away 2006), his keen directions and frequent encouragement have empowered the motivation for the PhD study;

To my great Mother, for her prayers for the sake of my success and to achieve my endeavours, for her being strong and patient, despite myself being miles away, despite illness and enormous sufferings;

To my gorgeous wife, lovely daughters, and sons, for their support, understanding, and patience during tedious and painful moments throughout the study duration;

To my mother-in-law, for her prayers, for her support, and her directions to my wife;

To my brothers and the wonderful sister, for their sympathetic and encouragement;

To my relatives and friends for their encouragement and continuous support;

I dedicate this work.

Abstract

Achieving and detecting a globally consistent state is essential to many services in the large and extreme-scale distributed systems, especially when the desired consistent state is critical for services operation. Centralised and deterministic approaches for synchronisation and distributed consistency are not scalable and not fault-tolerant. Alternatively, epidemic-based paradigms are decentralised computations based on randomised communications. They are scalable, resilient, fault-tolerant, and converge to the desired target in logarithmic time with respect to system size. Thus, many distributed services have adopted epidemic protocols to achieve the consensus and the consistent state, mainly due to scalability concerns. The convergence of epidemic protocols is stochastically guaranteed. However, the detection of the convergence is probabilistic and non-explicit. In a real-world environment, systems are unreliable, and epidemic protocols cannot converge to the desired state. Thus, achieving convergence by itself does not ensure making a system-wide consistent state under dynamic conditions.

The research work presented in this thesis introduces the Phase Transition Algorithm (PTA) to achieve distributed consistent state based on the explicit detection of convergence. Each phase in PTA is a decentralised decision-making process that implements epidemic data aggregation, in which the detection of convergence implies achieving a global agreement. The phases in PTA can be cascaded to achieve higher certainty as desired. Following the PTA, two epidemic protocols, namely PTP and ECP, are proposed to acquire of consensus, i.e. for the consistency in data dissemination and data aggregation. The protocols are examined through simulations, and experimental results have validated the protocols ability to achieve and explicitly detect the consensus among system nodes.

The research work has also studied the epidemic data aggregation under nodes churn and network failures, in which the analysis has identified three phases of the aggregation process. The investigations have shown a different impact of nodes churn on each phase. The phase that is critical for the aggregation process has been studied further, which led to propose new robust data aggregation protocols, REAP and REAP$^+$. Each protocol has a different decentralised replication method, and both implements distributed failure detection and instantaneous mass restoration mechanisms. Simulations have validated the protocols, and results have shown protocols ability to converge, detect convergence, and produce competitive accuracy under various levels of nodes churn.

Furthermore, distributed consistency in continuous systems is addressed in the research.

The work has proposed a novel continuous epidemic protocol with the adaptive restart mechanism. The protocol restarts either upon the detection of system convergence or upon the detection of divergence. Also, the protocol introduces the seed selection method for the peak data distribution in decentralised approaches, which was a challenge that requires single-point initialisation and leader-election step. The simulations validated the performance of the algorithm under static and dynamic conditions and approved that convergence and divergence detection accuracy can be tuned as desired.

Finally, the research work shows that combining and integrating of the proposed protocols enables extreme-scale distributed systems to achieve and detect global consistent states even under realistic and dynamical conditions.

Contents

List of Figures

List of Tables

List of Algorithms

List of Publications

2019 M. M. Ayiad and G. Di Fatta. ‘An Adaptive Restart Mechanism For Continuous Epidemic Systems’. In: *Internet and Distributed Computing Systems: 12th International Conference, IDCS 2019, Napoli, Italy, Ooctober 10-12, 2019, Proceedings* (10th Oct. 2019). Cham: Springer International Publishing, 2019. URL: `https://idcs2019.uniparthenope.it/`. Submitted.

2018 M. M. Ayiad and G. Di Fatta. ‘Robust Epidemic Aggregation Under Churn’. In: *Internet and Distributed Computing Systems*. Cham: Springer International Publishing, 2018, pp. 173–185. ISBN: 978-3-319-97795-9. DOI: `10.1007/978-3-319-97795-9_16`

2017 M. M. Ayiad and G. D. Fatta. ‘Agreement in Epidemic Data Aggregation’. In: *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*. Dec. 2017, pp. 738–746. DOI: `10.1109/ICPADS.2017.00099`

2016 M. Ayiad, A. Katti and G. Di Fatta. ‘Agreement in Epidemic Information Dissemination’. In: *Internet and Distributed Computing Systems: 9th International Conference, IDCS 2016, Wuhan, China, September 28-30, 2016, Proceedings*. Vol. 9864. Cham: Springer International Publishing, 2016. Chap. 9, pp. 95–106. ISBN: 978-3-319-45940-0. DOI: `10.1007/978-3-319-45940-0_9`

Glossary of Abbreviations

Abbreviation	Definition	Section
2PC	Two-Phase Commit	4.1
3PC	Three-Phase Commit	4.1
3PC-C	Three-Phase Commit-Convergecast	4.1
ADSL	Asymmetric Digital Subscriber Line	3.5
CDF	Cumulative Distribution Function	3.5
CTP	Collection Tree Protocol	4.1
CV	Coefficient of Variation	3.3
ECP	Epidemic Consensus Protocol	4.5
EMP^+	Epidemic Memebership Protocol-Plus	2.2
FLP	Michael J. Fischer, Nancy Lynch, and Mike Paterson	2.4
FU	Flow-Updating	2.1
GPL	General Public License	1.6
IDA	Information Dissemination Application	4.4
IP	Internet Protocol	2.5
IoT	Internet of Things	3.5
LAN	Local Area Netwrok	3.5
NCP	Node Cache Protocol	2.2
NCP^+	Node Cache Protocol-Plus	2.2.1
OSN	Online Social Network	5.1
P2P	Peer-to-Peer	1.2
PPG	Push-Pull Gossiping	2.1
PSP	Push-Sum Protocol	2.1
PTA	Phase Transition Algorithm	4.2
PTP	Phase Transition Protocol	4.4
PTP^+	Phase Transition Protocol-Plus	4.5
SE	Standard Error	3.3
SPSP	Symmetric Push-Sum Protocol	2.1
SSEP	System Size Estimation Protocol	4.3
REAP	Robust Epidemic Aggrgation Protocol	5.3
$REAP^+$	Robust Epidemic Aggrgation Protocol-Plus	5.4
RMSE	Root Mean Square Error	3.3
RTT	Round-Trip Time	3.5
TCP	Transmission Control Protocol	2.5
UUID	Universally Unique Identifier	6.1
WLAN	Wireless Local Area Netwrok	3.5
WSN	Wireless Sensor Network	3.5
WWW	World-Wide Web	5.1

Glossary of Variables

Variable	**Scope**	**Definition**
C	Local variable	Local cache of nodes information.
e	Local variable	Local estimate.
k	Global parameter	Number of neighbour nodes, or out-degree value.
$l^{\mathcal{Q}}$	Global parameter	Maximum length of queue $\mathcal{Q}$.
l^{C}	Global parameter	Maximum length of cache C.
$l^{\mathcal{P}}$	Global parameter	Maximum length of cache $\mathcal{P}$.
M	Global parameter	Number of agreement phases.
$\mathcal{M}v$	Notation	Total of initial values, or mass of values.
$\mathcal{M}v_p$	Notation	Total of initial values in nodes joined propagation phase in the aggregation process.
$\mathcal{M}w$	Notation	Total of initial weights, or mass of weights.
N	Global parameter	Number of nodes, or system size.
Np	Global parameter	Number of nodes joined the propagation phase in the aggregation process.
$\mathcal{P}$	Local variable	Local cache of aggregation information.
$\mathcal{Q}$	Local variable	Local queue of estimates.
t_{action}	Notation	Commit or action time.
tc	Notation	Convergence time.
t_{off}	Global parameter	Initial synchronisation offset.
$\dot{\mathcal{T}}$	Global parameter	Default cycle length, cycles are fully-asynchronous.
$\ddot{\mathcal{T}}$	Global parameter	Experimental cycle length, used for synchronising cycles.
$\check{\mathcal{T}}, \hat{\mathcal{T}}$	Global parameter	Maximum expiry time, or maximum timeout.
v or vd	Local variable	Aggregation value.
va, vc, vp	Local variable	Aggregation value for (*a*greement, *c*onvergence, *p*ropagation) phase.
$\mathcal{V}$	Notation	The true target value, or desired target.
w or wd	Local variable	Aggregation weight.
wa, wp	Local variable	Aggregation weight for (*a*greement, *p*ropagation) phase.
x	Local variable	Initial data value.
ε	Local variable	Local estimation error.
ϵ	Global parameter	Error tolerance threshold, or accuracy threshold.
Υ	Global parameter	Consecutive cycles threshold.
ς	Local variable	Seed identifier.
δ	Global parameter	Propagation delay of system network.

Chapter 1

Introduction

The discipline of distributed systems is a very active topic of scientific research for the last sixty years [1]. Recent communication technologies and computer networks have imposed further problems and challenges to the distributed systems, especially when scalability and synchronisation are a concern. The research presented in this thesis addresses the consistency problem in large and extreme-scale distributed systems. It proposes achieving and detecting of consistent states among system participants, although real-world systems are asynchronous and dynamic. Detecting a consistent state in realistic conditions is a non-trivial problem, and requires scalable, flexible, and robust solutions. This chapter gradually describes the consistency problem in extreme-scale distributed systems. It also outlines the aims, objectives and methodology of the research work. It summarises the main contributions and illustrates the thesis layout at the end.

1.1 Extreme-scale Distributed Systems

Over the last decade, the scale of distributed systems has expanded in size and space. The expansion of the distributed systems is a natural outcome to the enhancements in communication networks and the rising number of users and services. The advanced technology has enabled computing devices from different types and sizes and made them handy. Also, almost all computing devices these days are connected to the Internet and come supported with plenty of mobility capabilities and innovative functionalities. Furthermore, the emergence of widely deployed networks such as Wireless Sensor Networks (WSN), Internet of Things (IoT), and Cloud computing, together with the growth of users and services, have conveyed the scalability in distributed systems to additional dimensions.

In the present, each software application is usually relying on a middleware distributed system in one way or another. This reliance is promoted by the evolution of ubiquitous services and pervasive computation. For example, online services are accessible from many computing devices, whether a user is stationary or on the move [2]. Other types of services are adaptable to changes in the surrounding context, such as the case in smart appliances, vehicles, and spaces [3]. Recently, centralised services such as those in Cloud technology have been moved outside the computation center towards the edges for further processing capabilities (a.k.a Edge Technology) [4]. Also, some services are entirely decentralised and distributed, such as those in Blockchain technology [5]. Every service in the previous examples is built using a system of widely distributed participants (e.g., nodes, processes, agents) which are simultaneously running on several computing devices. The distribution of a system over a vast network that comprises thousands or millions of nodes imposes many difficulties and challenges in terms of scalability, synchronisation, and consistency.

In distributed systems, models of centralised algorithms and deterministic communications are controversial due to common problems such as the single point of failure and Bottlenecks [6]. Besides that, the unprecedented growth in systems scale arises further issues, in particular, distributed services that require system-wide operations. For example, synchronising distributed computational information among nodes or processes which are entirely asynchronous [7, 8], or attaining consistency and agreeing on a state that is critical for the operational continuity [9]. Moreover, acquiring a global control on nodes failures, which are the norm rather than the exception [10], or handling network dynamism and topology changes which may form an irregular structures [11]. Such issues are usually a concern for services designers and developers.

In response to challenges in centralised approaches, decentralised algorithms and Peer-to-Peer (P2P) communications have emerged as an alternative model of distributed systems [12, 13]. The general concept of the P2P model is the distribution of computations across system nodes such that they can collectively perform a task in a decentralised and scalable manner. Nodes in P2P networks usually interact in a peer style where a pair of nodes exchange information and each node potentially acting as both client and server, and hence, nodes run similar networking protocols and software. Perhaps the most popular classes of Internet applications nowadays is the P2P file sharing systems, conferencing applications, and cryptocurrencies [5, 14, 15].

One distinct property in P2P systems is the construction of an overlay topology on top of the physical network. It is vital for each node in a P2P system to retain a subset of peers through indexing and discovery [11, 16]. Typically, nodes can identify and communicate with each other using the local view of the overlay. However, pairwise information exchanging is performed via the underlying network. The overlay topology formulates the relationship among nodes, and therefore, overlay management is essential in the P2P systems [16, 17].

P2P overlays can be categorised to structured, unstructured and Hybrid systems. In structured systems, nodes are imposed in a special linking structure (e.g. Trees and Distributed Hash Tables) [18, 19]. In contrary, nodes in unstructured systems are not linked according to any particular or deterministic manner (e.g. Gnutella and Gossip) [11, 15]. Hybrid systems combine both structured and unstructured topologies to gain the benefits of both; for example, they are more resilient than tree-based systems and have less overhead than gossip-based systems [20]. However, unstructured topologies are also known as random topologies, and they gain attention due to supporting rapid data diffusion in dynamic distributed systems;

also, large distributed systems are usually assumed to have arbitrary topologies.

The P2P model is an attractive architecture for distributed systems due to the decentralisation feature. Mainly, the lack of a failure that can compromise or damage the system makes the model fault-tolerant, in addition to the ability to formulate unstructured overlays, which in consequence increases the model scalability. These characteristics of the P2P model have enabled an essential paradigm for decentralisation and scalability in distributed systems, the so-called *Epidemic*-based models. For instance, epidemic protocols are a typical P2P system that implements random pairwise data exchanging alongside decentralised computations over unstructured topologies, for which they form very flexible designs that is considerably more scalable, fault-tolerant and resilient.

1.2 Epidemic-based Protocols

A typical epidemic-based paradigm consists of decentralised computing models based on randomised communication models. The terminology and concept of epidemic protocols are inspired by the phenomenon of infection in biological systems [21, 22]. Likewise, the term '*Gossip Protocols*' is a synonym concept as both social gossip and virus infection shares the same diffusion properties. Epidemic models are known for a decade, primarily, for maintaining replicas in distributed databases [23] and for decentralised routing across the Internet, i.e., P2P and ad-hoc networks [12]. However, modern distributed systems have applied epidemic-based solutions for sophisticated problems on the contrary to other approaches which have failed, either due to extreme system sizes or due to inconvenient network structures [22]. Mainly, epidemic protocols have inherited the capabilities of P2P model, and hence, they have inspired the design of novel distributed protocols for large systems taking advantage of their intrinsic properties such as natural diffusion, resilience, scalability and fault-tolerance.

Epidemic protocols have acquired the popularity due to the ability to accomplish two main tasks: (1) *information dissemination* and (2) *data aggregation*. Information dissemination protocols merely employ the diffusion process of epidemics or gossip to disseminate data or queries from a participant to another, while data aggregation protocols use the diffusion process to compute a synopsis value or more for a distributed set of data in a system. For additional details on epidemic models for information dissemination a reader can refer to these pivotal studies [12, 22, 23, 24, 25]. Also, key models for data aggregation are presented in the following studies [26, 27, 28, 29, 30]. Also, Chapter 2 presents a review on additional epidemic protocols for data dissemination and aggregation.

One essential feature of epidemic protocols is the ability to eventually converge to a desired state. The convergence among nodes is critical for distributed services and it is excessively studied in the literature [28]. The term '*convergence*' is a temporal concept describes the reach to the desired state at each node in a system. In information dissemination tasks, convergence consists when all nodes in a system have received a copy of a particular information or update [31]. Also, the desired state in data aggregation tasks is typically a numeric value that each node in a system attempts to compute. Technically, an epidemic protocol achieves convergence when a locally computed result in each node gets arbitrary close to the target value [26, 27]. In stable systems, the theoretical analysis of epidemic protocols provided stochastic guarantees to the convergence of all nodes. In addition, practical studies proved the ability of each node to detect convergence using only its local state [28, 32].

In real-world systems, asynchrony and dynamism have a detrimental effect on the efficiency and the correctness of epidemic protocols [33, 34]. The inherent properties of epidemic models cannot be ascertained and may vanish due to the unpredictable behaviour of nodes. Epidemic-based services performance becomes questionable, and the tasks may not converge or may converge to wrong states. Therefore, it is essential to maintain the robustness of epidemic protocols under dynamical conditions to keep the dependent services operational. Further research on achieving and detecting of distributed consistent states using robust epidemic models can lead to adequate solutions for services in extreme-scale systems. The following sections demonstrate the importance of evolving epidemic protocols to achieve a robust consistent state for systems in a realistic environment.

As a note of terminology, an epidemic-based application is a software that incorporates one or more of epidemic protocols to provide a designated service. In such an application, internal components associated with a service (e.g., protocols) are located in each participant forming a distributed system for that particular service. In the thesis, the term '*epidemic system*' refers to an internal system of an epidemic-based service.

Definition 1.2.1. An *epidemic system* is a middleware distributed system that implements one or more epidemic protocols to accomplish global services and operations.

Also, the term '*task*' refers to a complete mission which includes at least one process. For instance, an aggregation task may consist of several parallel or sequent aggregation processes and likewise, a dissemination task. In the other hand, the desired state that is targeted by the computation in data aggregation tasks is denoted in the rest of the thesis as the '*target value*'.

1.3 Towards Consistency of Epidemic Systems

In extreme-scale distributed systems, global operations often utilise decentralised solutions to monitor and maintain a system-wide consistent state. For example, an online service where participants join and leave independently of the service may need to track the number of active participants to successfully complete a specific task [27]. Also, distributed applications for unmanned vehicles may repeatedly attempt to coordinate a universal speed limit to optimise system performance or to avoid catastrophic scenarios [35]. In WSN, devices frequently collect data from their sensors and require to have an up-to-date view of the network with aggregated information about fresh data in all devices [36, 37]. Recent trends in Edge computing have considered moving Cloud services away from centralised computation and data centres towards the edges of the network. Edge computing can benefit from and may even require decentralised monitoring and processing capabilities [4]. Also, Blockchain platforms require certainty on data immutability in the distributed ledger, so, it is necessary to continually keep a general agreement among nodes about all submitted transactions [5, 38].

The consistency of distributed systems refers to achieving a common object or state among system participants [39, 40]. Traditionally, a consistent state indicates a replica data of a distributed database, an outcome of a distributed transaction, a storage state under simultaneous operations, etc. Also, consistency can be strong or weak, achieved directly or eventually. Typically, distributed consistency is attainable through distributed algorithms that ultimately converge to a global state (a.k.a *Eventual Consistency*). In similar manner, algorithms for distributed consensus exhibit characteristics of those for distributed consistency. The consensus is critical to many distributed services as nodes or processes are often required to coordinate some action or agree on some value [1, 41]. In general, consensus algorithms are effective techniques for achieving consistency. Through coordination among participants, a distributed algorithm can achieve and synchronise a globally recognised state, which allows the accomplishment of system-wide tasks such as termination, event ordering, and decision-making.

Commitment protocols (e.g., Three-Phase Commit (3PC)) are typical examples for services that make consistency of distributed systems, normally, by using decentralised collective decision-making (or *voting*). The decision-making process is performed among participants targeting the agreement on an output that is in a common interest of all participants. The agreement target is usually a distributed state such as number of failures, commitment of a transaction, formed opinion, elected leader or consent on replicas [42, 43]. The key fact is that all participants must come to the same conclusion regardless of the problem they are trying to agree on [9].

In a typical agreement process, each node proposes or starts with an initial value, nodes exchange their values, and each node decide on one of the values. The agreement process ends when all nodes have decided a common value [44]. For a correct agreement, i.e. globally consistent state [43, 45], all nodes should eventually decide the same value and that value should be one of initial or proposed values. On the other hand, gossip-based models are widely used to achieve the distributed consensus and eventual consistency in distributed systems [40]. In particular, gossip-based averaging is the conventional process for acquiring agreement among nodes (a.k.a *Approximate Consensus* and *Asymptotic Consensus*). These studies: [40, 46, 47, 48, 49] are a sample related work for the gossip-based consensus.

Definition 1.3.1. An *agreement process* is a distributed process that coordinates among system participants to achieve consensus.

The consensus in 3PC is usually detected by a coordinator which exhibit same centralised approaches pitfalls. Also, agreement process in distributed consensus and eventual consistency models is probabilistic and it is approved to be achievable in stable and synchronised systems. Large systems are more open, asynchronous and dynamic. Conventional agreement methods either do not scale or are not adequate for such systems. Epidemic systems are scalable, however, achieving and detecting a globally consistent state in epidemic systems is non-trivial problem. It requires not only decentralised and resilient solutions but also robust and continuous protocols that handle changes in a dynamic environment.

In epidemic systems, consistency supports the reliability and predictability of the system by achieving a common state at asynchronously acting nodes. Consistency of epidemic systems requires all nodes to achieve a particular state (or value) at the end of dissemination or aggregation tasks [22, 46, 48]. The agreement process in epidemic systems aims to compute a synopsis among initial system values and eventually, all system nodes converge to the target value. The similarity of agreement processes in distributed consensus and data aggregation process is an unsurprising finding due to the well-known guarantees on the convergence of decentralised models. The ultimate convergence of decentralised models has motivated many researchers to use achieving convergence as a synonym to reaching agreement in distributed systems, primarily, when the target value is the interest of the consensus. This motivation also points out the essential role of convergence in the consistency of epidemic systems.

The research work in this project studies the problem of achieving and detecting consistency of epidemic systems. An epidemic system that regularly makes a consistent state can achieve reliable performance and predictable behaviour. Investigations have addressed eventual consistency and consensus problems in distributed systems to achieve the research objectives. The pivotal proposition is that reaching a consensus among system nodes makes a consistent state, and the process of agreement is the way to make and synchronise the global state. The study mainly considers the decentralised algorithms for distributed consensus and the distributed data aggregation problems in the sake of exploring potential techniques for robust and continuous epidemic protocols.

1.4 Consistency Problem in Epidemic Systems

The convergence of decentralised models is the foundation block for achieving distributed consistency in epidemic systems. However, acquiring of globally consistent state not only requires achieving convergence, but also the detection of convergence. Convergence detection in decentralised models is typically heuristic, and it is not guaranteed in real-world systems. Moreover, recent applications are demanding efficient and reliable consistency, for which the certainty on the system convergence requires explicit detection methods. The explicit detection method aims to build awareness among participants about the convergence of the system.

In stable epidemic systems, the ultimate reach to an agreement is supported by the ultimate convergence as proven by theoretical and empirical analysis. Each node in a synchronous system can directly discover the agreement by detecting the convergence, and it can instantly decide the consensus. Also, models for asynchronous systems have used the same assumption and rely on the inevitability of convergence in stable systems [27, 45, 48, 49, 50]. Asynchronous algorithms for consensus assume that a system will eventually converge in a finite time and hence, the detection of convergence implies the agreement, and a node can decide the consensus after bypassing a predefined interval. In general, the agreement in previous models is entirely heuristic, and a node has no apparent certainty on the convergence of other system nodes.

System convergence in real-world environments is neither can be ensured nor can be detected. The problem of achieving and detecting convergence in epidemic systems which are asynchronous and dynamic is described in the following points:

- Nodes achieve convergence at different times, and each node has no awareness about other nodes' state of convergence [51, 52]. Although a node may locally detect convergence, it is not sure about other nodes which may achieve convergence as well, or fail, or abort the agreement process.

- The grace interval before deciding the consensus in asynchronous models requires global information about the system, and the interval has to be long enough for a correct consensus, which is impractical in extreme-scale systems and challenging under the presence of dynamical conditions.

- Computations in epidemic models are decentralised by nature, and the detection of system convergence can only be obtained through the absolute reliance on local state at each node. Moreover, the certainty of global detection of the convergence cannot be directly presumed by local detection due to nodes asynchrony. In the lack of a central authority like in commitment protocols (e.g., 3PC), it is a challenge to use only locally available information at each node in acquiring the awareness of other nodes' convergence.

In the context of the previous explanation, the consistency of epidemic systems is defined as follows,

Definition 1.4.1. *Consistency* of an epidemic system is a model in which each participant of the system achieves a globally desired state and achieves an explicit awareness about other participants that have made the same state.

In contrary to conventional distributed and eventual consistency models, which typically define the global target of consistency by the convergence to a common state, the prior definition requires the acquiring of global awareness on the convergence of the system, i.e., the convergence of other participants. The proposed consistency model of epidemic systems preserves the following general properties:

1) *Termination*, all non-faulty nodes should eventually decide or compute some value.

2) *Agreement*, the target is the same for all non-faulty nodes.

3) *Validity*, the target is among, or an aggregate of the set of proposed and exchanged values.

The proposed model of consistency for epidemic systems also exhibits the liveness and safety properties of eventually converging distributed models [40]. Both properties are guaranteed by the theoretical and empirical analysis of convergence in epidemic paradigms, particularly, under stable conditions [22, 26, 27, 48, 50].

Definition 1.4.2. The *liveness* of the consistency model consists of the ability to converge in a finite time. It is an intrinsic reflection of the diffusion process in the epidemic paradigms.

The safety property is also an outcome of the convergence feature of the epidemic models, and it is defined as follows:

Definition 1.4.3. The model is *safe*, if all participants ultimately converge to the same desired state in a finite time.

The liveness and safety properties are preserved in stable epidemic systems; however, these properties are questionable in real-world systems where the convergence is detrimentally influenced. One substantial aim of this research work is to achieve a global awareness among system participants for a particular state, e.g., the convergence to the desired target. Making system-wide awareness ensures the safety and liveness of the epidemic systems, even in dynamical and unreliable environments.

In essence, the problem of consistency in epidemic systems is an extension of the distributed consensus to the distributed data aggregation where the agreement is an epidemic aggregation process. In order to achieve agreement on a particular matter, nodes in the system have to (1) receive a particular data item or compute a local estimate of the target value using a specified data aggregation function, (2) detect local convergence on the target, (3) acquire explicit awareness on the convergence of the system. A typical example of the proposed agreement process in actual epidemic systems is the local estimate of a data synopsis function, where a node may need to achieve three different levels of information, the target value with some approximation, awareness of local convergence to the target and certainty of a global convergence on the target.

The decentralised data aggregation is the core of the agreement process and therefore, the robustness of the aggregation process is important for making a correct agreement. In dynamic conditions, aggregation process may not converge or results may deviate of the target value, which invalidates the aggregation process and the agreement process as well. Achieving a globally consistent state in real-world epidemic systems requires robust and continuous models that adapt to new system conditions. Adaptive epidemic models make correct consensus and consistency on the most recent state of the system in spite of the asynchrony and dynamical conditions.

1.5 Research Objectives

The pivotal hypothesis of the research work in this project is to utilise epidemic paradigms as adequate solutions for the extreme-scale systems, which are asynchronous and dynamic.

We address attaining the distributed consensus and making consistent states as a practical application for the proposed epidemic protocols in this project, mainly due to the vital roles that consistency and consensus services have in the large distributed systems nowadays. We argue that current decentralised models for distributed consensus and eventual consistency are inefficient for real-world epidemic systems, although theoretical and practical analysis of decentralised models have provided guarantees on the convergence to a global state. We propose that the consideration of realistic environments fundamentally needs to satisfy two main conditions:

1) Each participant of an epidemic system should achieve and detect the desired state, and

2) Each participant should detect convergence of the system to the desired state, not depending on probabilistic assumptions only but also using an explicit detection method that builds global awareness of the system state.

We also propose that epidemic models need to be continuous and adaptive to handle the changes in the system due to dynamical conditions. The research project had the following general aims:

1) To review and study decentralised and epidemic protocols for data dissemination and aggregation in the literature. The analysis of existing epidemic models provides the fundamentals for introducing novel protocols.

2) To investigate the principles and implications of systems synchrony, which are essential for the proposal of new protocols. The traditional assumption of a proper synchronisation in the system is impractical in the real environment. The project aims to ustilse a practical system model that is asynchronous and realistic. On the other hand, it is vital to understand the impossibilities of distributed systems. The consideration of the impossible cases helps to adopt the typical synchrony model, in which epidemic systems can operate.

3) To explore models of churn and dynamism in P2P networks to identify the most practical model to use in validating the proposed epidemic protocols. The proposed protocols should not be penalised by examining them under extreme scenarios that are uncommon or rarely occurs.

4) To propose epidemic protocols that can achieve and detect globally consistent states in epidemic systems and can allow services to operate predictably and reliably. The proposed protocols are required making explicit awareness in the system, although it is dynamic and asynchronous.

In the course to achieve the project aims, the following objectives are proposed:

1) To define the level of synchrony in the system model. This includes using appropriate synchronisation setting for cycles start times and length. It also includes determining the density distribution of network propagation delays, typical messages size and transport layer reliability.

2) To introduce innovative epidemic protocols for the correct making of a globally consistent state under realistic conditions. The roadmap to achieve this objective starts by investigating the convergence feature of the epidemic and decentralised models. The investigations include determining the appropriate methods for convergence detection. Then use the ideal detection method in the novel protocols. Additionally, it is an objective to introduce an innovative approach for making and detecting global awareness in the epidemic systems.

3) To study the performance of epidemic protocols in dynamic systems. The study addresses the dynamic behaviour of nodes and examines the impact of nodes churn on epidemic protocols, particularly, on data aggregation protocols. The research target is also to introduce robust epidemic protocols that can converge under the dynamical conditions.

4) To apply adaptive behaviour to the proposed epidemic protocols and make them applicable for dynamic systems. The research target is to study potential methods to discover the detrimental impact of the nodes' churn on epidemic protocols, and upon detecting any consequences of churn, the protocol adapts it self and takes appropriate actions.

Finally, the research project involves the incorporation of proposed methods, mechanisms, and protocols to introduce generic paradigms for robust and consistent epidemic systems that can operate in spite of vast scale, asynchrony, and dynamism.

1.6 Methodology

The research study in this project is simulation-based. The utilisation of simulation is a typical choice due to wide system sizes for which the epidemic models are designed. The direct deployment of epidemic protocols in real-world systems for the purpose of examinations is a complicated, expensive and impractical strategy. In contrast, computer modelling and simulation provide adequate tools for the experimentation and enable the studying of alternative designs and architectures.

The opted methodology follows a gradual research technique to facilitate investigations and experimentations. At the start, the research has addressed epidemic models in steady systems. After that, the study has involved the impact of churn and dynamic conditions. In general, the research consists of the analysis of both theoretical concepts and empirical results of epidemic protocols. For this purpose, some epidemic protocols from the literature in addition to the novel protocols are implemented and examined using simulations. The experimental results are inspected and verified in the shade of analytics and proofs in the previous research studies. Also, the performance of the novel epidemic protocols is evaluated and compared to other existing protocols.

The simulations are carried out using PeerSim [53] a Java-based discrete-event P2P simulation tool that is provided under the (GPL) open source licence. The simulator is developed primarily for evaluating decentralised protocols which are designed for extreme-scale and dynamic distributed systems. Also, PeerSim can simulate different network topologies such as structured and unstructured overlays.

There are two separate simulation engines in PeerSim: (1) *Cycle-based* and (2) *Event-based*. In the cycle-based model, a protocol in each node is called upon at each cycle and protocols communicate in a randomised manner. It is a simplified and well documented simulation engine [54]. In the event-based model, events (e.g. message dispatching) are scheduled based on their delivery times and a node protocol is called upon according to events reception order. The usage of the event-based engine in PeerSim is not a straightforward task due to substandard documentation [54, 55].

In another perspective, PeerSim has no graphical user interface but provides lightweight Java packages for the simulation and basic statistical calculations and data collections. The packages include many simple, extendable, and pluggable support components [53]. The entry point to the simulator is command based, where a group of predefined instructions, protocols, and parameters are inputted through a text configuration file.

The simulation of various conditions of real-world epidemic systems demanded additional simulating components. In particular, the transport layer component in PeerSim is extended to generate random delays following a specified statistical distribution. The new transport component implements a Weibull distribution with a scale η, a shape β, and a location γ. The Weibull distribution is flexible and can simulate probability density function against the latency of various networks. Mainly, the parameter β can control the upper bound of delays and can make short or long-tailed distribution according to the requirements of the experiments. Also, the parameter η defines the average value or the distribution peak. The parameter γ bounds the lowest value.

In this project, the event-based model is chosen for the implementation of the protocols, because it is adequate for simulating actual systems. For this purpose, a programming framework for epidemic protocols is implemented for PeerSim. The framework consists of process-alike structure and events to control the life-cycle of Protocol component in PeerSim, which is extended too to accommodate the new framework. Services in the framework are accessible but not compulsory to protocols implements, thanks to default methods in Java 8. The framework makes concurrency implementation and protocols structures more controllable and understandable. The following describes the basic services of the framework:

1) Initialise Service sets the initial state of a protocol. PeerSim initialisation

components usually use this service before the simulations start. This service can also schedule the Start Event or Run Event for a calling protocol.

2) Start Service begins a protocol execution. This service schedules the Run Event for a calling protocol at the beginning of the simulations. The service enables different start times for protocol instances in a system.

3) Sleep Service reschedules the Run Event for the calling protocol during the simulations.

4) Send Service passes information and data to the transport component in PeerSim where the Message Event is scheduled for a random delivery time.

Also, the next list involves a description of the common events in the framework:

1) Start Event is an active thread of a protocol that runs only once at start time.

2) Run Event is an active thread that is usually called at each cycle. In this event, a protocol should perform main computation and communication activities.

3) Message Event is a passive thread that runs upon the occurrence of the event only. In this thread, a protocol processes a message from a peer and may sends a response.

Moreover, there are data collection and statistics components are developed for PeerSim to ease the analysis and evaluation of experimental results. For example, a group of integrated components to produce visual results such as figures and graphs are implemented. Additional effort is also applied to tune the performance of examined protocols and optimise the usage of memory and processor.

In summary, the lightweight package of PeerSim and command-based configuration were ideally flexible for running several simulations in parallel. Also, PeerSim was able to simulate up to 10^6 of nodes in mid-performance computer. Experiments durations were ranging from several minutes for thousands of nodes to a couple of hours for one million nodes. Generally, the simulation methodology has enabled this research to achieve intended objectives and aims and present vital contributions to the epidemic systems and to the discipline of decentralisation.

1.7 Contribution

The research work in this project has studied epidemic models of data dissemination and aggregation. Additionally, it explored epidemic protocols for distributed consensus and consistency in the context of achieving the project aims and objectives. It presents a distinct combination of decentralised models for extreme-scale systems with those to distributed commitment. The work has introduced innovative epidemic protocols for achieving and detecting systems consistency under stable and dynamical conditions. It has also proposed mechanisms to improve epidemic protocols' robustness, which, when incorporated to epidemic protocols for consensus, make modern epidemic paradigms that are scalable, asynchronous, and robust for the real-world systems. In essence, the work presented in this thesis makes the following novel contributions:

1) It defined the partial-synchrony model of epidemic systems. The model adopts asynchronous communication and processing cycles. It also adopts a non-atomic pairwise message exchange, and hence, messages interleaving are present with high probability. Cycles of different nodes may overlap, and messages may deliver in different cycles. However, it assumes a reliable underlying network. Cycle length is also defined and analysed. The recommended cycle length is the RTT on the diameter of the system network. The utilised cycle length allows non-atomic pairwise exchange transactions to complete within the cycle with high probability. As a sub-contribution, the adopted partial-synchrony setting has demonstrated the applicability of the proposed protocols for real-world deployment.

2) It improved the heuristic methods for local detection of convergence in epidemic protocols. The analysis has shown that conventional detection methods require global information on the system, which hard to obtain, especially in dynamic environments. The work in this project has introduced an evolved detection method that requires no prior information about the system. Also, it has approved that although some controlling parameters are needed, the setting of those parameters can have standard values, and thresholds can be set according to application preference.

3) It introduced the Phase Transition Algorithm (PTA) for making globally consistent states in epidemic systems. The PTA makes the transition into phases to acquiring more certainty and global awareness on the desired states. Each phase of PTA is a decentralised decision-making process that aims to attain explicit agreement on the system state. The algorithm is further implemented in two protocols for reaching and detecting agreement in data dissemination and aggregation, PTP and ECP respectively. The algorithm is flexible and can achieve system awareness with as certainty as required.

4) It provided a survey on the nodes churn models, particularly in P2P networks and systems. The survey aimed to adopt the typical churn model for studying epidemic protocols. With some general assumptions on nodes departure, estimation for the expected departure rates was obtainable from the statistical distributions of session duration provided in the surveyed studies. The survey has shown that under normal system conditions, 30% of nodes churn in the average should be expected. This result implies that churn rates are usually moderate, and epidemic protocols should provide the desired accuracy at the mid-level of churn.

5) It analysed the data aggregation process in which three implicit phases are identified. The phase that is critical to the aggregation process is further studied, leading to

introducing two robust epidemic aggregation protocols: REAP and REAP$^+$. The protocols implement an instantaneous failure detector and recovery mechanism to compensate for the disturbance in the mass conservation property. The prompt compensation method supports the aggregation process and makes it converge to a more accurate approximation of the target value. The protocols produce results with high accuracy even under severe nodes churn. The achieved accuracy is competitive, and levels of errors under different churn rates have been specified.

6) It introduced a novel protocol for continuous epidemic services with the adaptive restart mechanism. The protocol either restarts upon acquiring consensus on the global convergence or upon the detection of divergence. Moreover, the mechanism generates optimised communications overhead that can be piggybacked with regular message exchanging. The protocol also implemented a decentralised selection method for data aggregation tasks that require single-point initialisation, which was a challenge that needs the leader election step. Furthermore, the detection accuracy of the protocol can be tuned according to the application preference for good quick results or accurate estimates that take longer to compute.

In general, the research work in this project has introduced several methods, mechanisms, and protocols to the discipline of epidemic computing. The PTA protocols have provided the explicit method for achieving and detecting consistent global states in the epidemic systems. The method allows distributed services to accomplish any designated global action or decision. The proposed robust data aggregation protocols and when integrated to PTA protocols improve the convergence and the results under dynamic conditions. This makes the acquiring of consistency achievable in dynamic systems. Furthermore, by combining the adaptive restart mechanism, epidemic protocols can adapt to the most recent state and can make correct system-wide decisions and actions. The introduced contributions provide fundamental solutions for many services in modern distributed systems, especially when global synchronisation, coordination, and consistency are required for services operations.

1.8 Scope of The Research Work

The research presented in this thesis is targeting distributed services in the large and extreme-scale systems. Thus, decentralised solutions are adopted due to their scalability and fault-tolerant properties. However, utilising of decentralised models exhibit higher overall communications. Structured and centralised models can be adequate for small systems and provide better performance and communications loads.

There are two popular applications for the decentralised models, including the epidemic models: information dissemination and data aggregation. Any distributed service that applies information dissemination and data aggregation as its primary operation or task can use any of the proposed protocols in this project. The main objective of the project is to achieve a globally consistent state in large systems under stable and dynamical conditions. Distributed services that require achieving and detecting global consistency can merely implement one of the PTA protocols. For instance, a service that consists of information or data dissemination tasks can use the protocol PTP. Service that computes a synopsis of distributed data or for system proprieties can use the protocol ECP for achieving consistency.

All previous services that desire to achieve consistency under dynamical conditions need to consider combining or solely using protocols such as REAP$^+$ and the continuous epidemic protocol with the adaptive restart to make robust epidemic operations and tasks.

1.9 Thesis Outline

This thesis comprises seven chapters. Apart from this one, each chapter describes a stage of the research work in this project. Chapters include sections facilitates the investigations and protocols proposed at each stage. This thesis is organised as follows.

The next chapter describes the literature review stage and provides information on the most relevant research which is required to explain the work conducted in the project. The impossibilities in real-world distributed systems are specified in the chapter. The chapter also describes the model that identifies the minimum requirements needed for epidemic protocols to operate in the asynchronous environment.

Chapter 3 discusses the convergence of epidemic protocols. It explains heuristic and explicit detection methods in the context of the given system model. The developed detection methods presented in the chapter are the main foundation blocks in the proposed protocols. The chapter ends, illustrating the convergence time from a practical point of view and discusses mapping convergence time to true-time units.

Chapter 4 introduces the Phase Transition Algorithm (PTA), for achieving and detecting consistent states in epidemic systems. In the chapter, the agreement process in centralised distributed systems is explained emphasising the need for acquiring global awareness in the distributed consistency problem. Also, a complete description for the PTA and the epidemic agreement protocols is provided. Further discussions on the PTA are presented at the end of the chapter.

Two protocols for robust data aggregation with a distributed failure detection and mass restoration mechanisms are introduced in Chapter 5. In the chapter, research studies on the node churn are surveyed to specify the typical churn model in P2P networks. The chapter includes the analysis of epidemic data aggregation process where three distinct phases of the process have been identified. The study has shown that node churn at each phase has a different effect on the estimation error, resulting in a novel mechanism to address the violation of mass conservation invariant.

The work in Chapter 6 introduces a novel epidemic protocol with an adaptive restart mechanism. The mechanism restarts an epidemic task upon the detection of convergence or divergence in autonomous and variant epochs. Also, the mechanism ensures correct convergence to the target for all nodes through aggregating nodes decisions and acquiring consensus on the restart action. Moreover, the mechanism is producing small communication overhead, which can be piggybacked on existing protocol messages.

The thesis ends with Chapter 7 that presents a summary of the work presented in the thesis and the aims and objectives which have been achieved. Also, further research ideas are provided.

Chapter 2

Literature Review

The work in this project extends distributed consensus problem to epidemic data aggregation problem, in which both problems need to be studied. This chapter provides an exploratory review of the literature that formed the initial stage of the project. In the chapter, the most relevant research work and studies are described emphasising answers they provide for the research questions of this project. In the beginning, studies and paradigms of the epidemic information dissemination and data aggregation are presented. Mainly, the review was questioning existing protocols that have been identified as decentralised and asynchronous. Communication and computation models of epidemic protocols are carefully inspected, and some essential techniques have been implemented and examined through simulations. Distributed system models for which the explored epidemic protocols are proposed are investigated too, targeting the ideal model of synchrony for the real-world systems.

The review also involved research studies on coordination, consensus and consistency of distributed systems. At this stage, the review aimed to identify the key solutions for the consistency problem, particularly when decentralisation and asynchrony are requirements. Solutions to distributed consensus and consistency that adopt gossip-based or epidemic algorithms are also considered. General assumptions, design and implementation decisions, and results are analysed and investigated.

Topology management and peer-sampling services have a considerable impact on the robustness of the epidemic protocols. Thus, epidemic protocols for managing overlay topologies and peer-sampling are presented and review in this chapter. In dynamic systems where topologies may change or disconnect, peer-sampling services typically have a vital role in supporting epidemic protocols efficiency and accuracy, particularly, data aggregation protocols which are highly influenced by the dynamical conditions.

The impossibilities of asynchronous and dynamic distributed systems are also discussed in the chapter. Principally, the review investigates the suitability of epidemic protocols to deal with the impossibility cases, and under what assumptions, can epidemic protocols operate. The chapter describes the partial-synchrony model adopted throughout the research project and demonstrates the level of asynchrony in the computations and communications.

2.1 Data Dissemination and Aggregation Protocols

Demers et al. [23] introduced the earliest well-known epidemic algorithm for updating database replicas on the Internet for Xerox Corporation. Mainly, the work aimed to introduce a reliable and rapid algorithm that copes with the increasing size of the corporate network. The work has proved the efficiency of randomised data exchange as an alternative to the deterministic model. Also, it showed that randomised communication algorithms are straightforward and simple to implement, and achieve information convergence in logarithmic time concerning the network diameter. The work of Demers et al. has also proposed randomise pairwise exchange that distinguishes *anti-entropy* and *rumor-mongering* modes. Typically, anti-entropy implies each replica regularly contacts a random server and resolves differences, whereas in rumour-mongering, a node which had or received a new update, disseminate the update to a random peer, and after some time this update encounters less interest and so can be removed. Demers et al. algorithm is a randomised information dissemination to make the eventual agreement on replicas.

Rumours dissimulation is investigated in [56]. Authors have analysed the problem of spreading rumours in a distributed environment using synchronised rounds and randomised communications. They considered the lifetime of the rumour and analysed the transmission and time complexities of PUSH and PULL communications, which led to introducing a PUSH-PULL model as the more efficient and provides lower complexity. The major issue that has been addressed by the work is how nodes decide whether the rumour shall be forwarded to a peer or not, and the global termination mechanism. They prove that the number of transmissions can be reduced significantly when rumours are sent in both directions. They also mentioned examples of theoretical work for data consistency in the persistence of failures. The work was precocious notice for the need of global convergence detection in epidemic information dissemination.

The basic mechanism for periodic information dissemination, which is decentralised and based on local information was introduced by [12, 57]. Formally, all nodes are involved in the information dissemination, and every node stores every piece of information it receives into a buffer, and continuously forwards all information to a peer or set of peers that are randomly selected at every time. The work defined several variants, e.g. length of the buffer that may limit inbound information, number of cycles needed to spread information over a system, and set of peers for the random information exchange. Typically, information dissemination quality and efficiency may vary base on these parameters and the network size.

An optimisation to epidemic information dissemination was *Spatial Gossip* where, instead of selecting a random peer, nodes select peers based on a distance metric [24]. This method noticeably reduces the load on bottleneck links which connecting communities of nodes. In Spatial Gossip, nodes choose peers that are closer in the topology, thereby reducing the load on long-distance links, but pay the cost of slower spreading speed. In the same respect, a comprehensive systematic survey of many recent epidemic algorithms for information dissemination is provided in [22]. The algorithms described in this work are interdisciplinary tools from Markov chain theory, Optimization, Percolation, Random graphs, Spectral graph theory, and Coding.

A known study for exchanging information for computing in an arbitrarily connected network of nodes, e.g. WSN is presented in [25]. Constraints such as dynamic topology and limited computational, communication, and energy resources have motivated the work to address epidemic algorithm for WSN systems. The work has analysed the problem from many aspects, and provided guarantees on the convergence of a network with arbitrary matrix of values derived from natural random work and without any central coordination.

Montresor A. and Jelasity M. studies are common and recent source of epidemic data aggregation and peer-sampling protocols. Mainly, the authors have proposed aggregation protocols to compute global aggregates, e.g. average and count, inspired by the anti-entropy protocols [58]. They showed that date aggregation generalises the differences resolving to a numeric form of computation. Their work is proactive and hence, has no performance bottlenecks, and the aggregate results are present continuously at all nodes. They indicate the adequacy of aggregate protocols for collective decision making and automatic system consistency based on global information in a fully decentralised fashion. Márk was also interested in the dynamics of the diffusion of information and has defined the terminology of epidemiology whereas each node can be in one of three states, *Suscepitble*, *Infcted*, or *Removed* [59]. The author introduced these models, which are derived from node states and based on Push-Pull model. Mainly, the author aim was to update all nodes while minimising the redundant message transmissions and keeping the algorithms simple. In addition to introduce a general framework for solving the security and malicious behaviour problem in epidemic systems [60], demonstrate the efficiency and robustness of epidemic protocols theoretically and experimentally under stable and dynamic scenarios [27], proposed topology management and peer-sampling services for robust and efficient epidemic data dissemination and aggregation [13, 61], and they have implemented and provided the simulation tool, i.e. PeerSim, which has been adopted for work in this project [53].

In the literature, distributed data aggregation is widely popular, and essential for a broad range of services because it is independent of the overlay topology and the underlying network which gives the data aggregation broad flexibility for various applications. The following sample of studies, which have been reviewed, highlights the popularity of data aggregation algorithms in large-scale distributed systems.

The work in [62] introduced a Tiny AGgregation service (TAG) for low-power WSN. TAG allows the spread and execution of queries over an ad-hoc network of sensors connected by radio. In [63], the Generic Aggregation Protocol (GAP) is introduced to collect data from devices for network management purposes in a scalable and robust manner. Authors in [64] proposed a Gossip Time Protocol (GTP) for decentralised time synchronisation in the P2P network. The protocol is self-management, and hence, time samples evaluation and gossiping frequency are entirely a local decision of nodes. Also, the work of [65] presented a monitoring and global load estimating epidemic algorithm. Information about CPU load and memory usage are exchanged from a peer to another peer chosen from the network. From another perspective, authors in [47] showed that epidemic broadcast algorithms indeed almost converge to a consensus with less overhead in contrast to standard retransmission mechanisms in lossy networks.

Furthermore, epidemic protocols in data mining is introduced in [66]. Authors have proposed a fully distributed k-means based on an epidemic algorithm. The algorithm is a clustering solution that can approximate the centroid over the aggregated data as closely as desired without global communications and is intrinsically scalable and fault-tolerant. An example of decentralised reputation computation is presented in [67]. A differential gossip-based algorithm is proposed to exchange votes among a node and its neighbours. The authors in [68] proposed a set of algorithms for failure detection inclusive of an algorithm based on gossiping with an additional component to achieve consensus. After the detection of a failure, the epidemic algorithm disseminates failure information to all non-faulty processes in the system. A consensus on a failure is achieved by utilising the same disseminating process to compute the number of the informed nodes. In another perspective, epidemic algorithms for peer-sampling services and topology construction and maintenance are also proposed [11, 69, 70], for instance, authors in [33] introduced an epidemic protocol to maintain connectivity in dynamic networks of low degree distribution.

The work in [71] has found two hurdles in designing algorithms for the data aggregation problem, the wide distribution due to the vast scale of the system, and the continuous changes due to a network or a nodes churn. Changes to the set of distributed data can take place even while the aggregate is being computed. Hence, the validity of the aggregate function becomes unclear [71]. Authors also define the nations of validity for data aggregation algorithms. They show that data aggregation approaches in P2P networks can practically achieve approximated validity rather than computing strong valid aggregates. Additionally, authors in [28] have evaluated a range of data aggregation algorithms. They prove that data aggregation robustness and accuracy are compromised in the real-world environment. They found that in practice, data aggregation is a non-trivial problem in realistic scenarios when requesting solutions for a distributed environment where no single node holds a global view of the system.

In additions to the previously reviewed work for data dissemination and aggregation, several main data aggregation algorithms are further implemented and examined through simulations. The experimental study aimed to make a deep understanding of the data aggregation problem. Mainly, the practical review addressed two properties of the data aggregation protocols, convergence when a uniform gossiping with dynamic topologies is adopted, and when system asynchrony and dynamism is present. The outcomes have adjusted the direction of the research work in the project and helped to make critical design and implementation decisions. The following sections describe the practical review work.

2.1.1 Push-Sum Protocol

Asynchronous epidemic data aggregation protocols that based on PUSH only model is introduced in the work of [26]. Authors conducted a theoretical and practical analysis and proposed several extend aggregate functions beyond regular summation and average functions. The protocol is fully decentralised and can be used to achieve global tasks, for example, random samples, quantiles, and answer several data aggregate queries. Small and randomised data PUSH messages are adopted for simplicity and fault-tolerance. The analysis provides in work [26] has provided stochastic guarantees on the convergence of the protocol results to the correct answer exponentially fast. Primarily, the PUSH-SUM protocol is a simple and natural protocol to compute the summation and average. Algorithm 1 illustrates the protocol for a system of N nodes. For the aggregate summation, a small change need to be applied, instead of all nodes starting with weight $w_{i,t0} = 1$, only one node starts with weight 1, while all others start with weight 0.

Algorithm 1: Push-Sum Protocol

```
Require: a peer-sampling service, e.g. NCP$^+$.
Initialisation: $v_i = x_i$; for average, $w_i = 1$; for summation $w_{\hat{\imath}} = 0$ at a single node $\hat{\imath}$
    and $w_i = 0$ at all other nodes; $0 < i \leq N$; where $v_i, w_i$ are
    aggregation parameters at each node $i$, and $x_i$ is the initial value of $i$.
-----------------------------------------------------------------
1 At each cycle at node $i$:
2 $j \longleftarrow getRandomPeer()$
3 $v = \frac{v}{2}$                               // divide aggregation pair
4 $w = \frac{w}{2}$
5 send $\langle v, w \rangle$ to $j$                     // a PUSH message to node $j$
-----------------------------------------------------------------
6 At event 'receive message $m$ from $j$' at node $i$ and time $t$:
7 $v = v + m.v$                                   // update local pair
8 $w = w + m.w$
9 $e = \frac{v}{w}$                               // local estimate of this step
```

At every cycle, each node divides the local sum values by two and sends the pair of values to a random peer. In the peer node, the local values are updated. The aggregate result e can be estimated at every node by $e_{i,t} = \frac{v_{i,t}}{w_{i,t}}$. The accuracy of the produced estimate will tend to increase along each cycle, converging to the correct value.

The authors in [26] have defined a vital system property, so called, *mass conservation* invariant. System mass is the aggregate of all initial set of distributed data, where the average of all sum values $v_{i,t}$ is always the correct average, and the sum of all weights $w_{i,t}$ is always N, this property is further described in Section 3.2. Aggregation protocols that violate the conservation invariant cannot converge to the true results. The authors have assumed the existence of a failure detection mechanism, that allows nodes to detect when a PUSH message did not reach its destination. In this situation, the mass can be restored by sending the undelivered message to the node itself. They also stated that the assumption of synchronous cycles is not truly necessary for the Push-Sum protocols. Instead, nodes may simply follow their own clocks in deciding when to contact a peer and the mass conservation is still ensured.

Figure 2.1 is a sample of the experimental review in the project. The figure illustrates the performance of the Push-Sum protocol. The protocol is simulated under asynchronous setting: random dynamic overlay, cycles among different nodes may overlap, and messages typically arrive at different cycle.

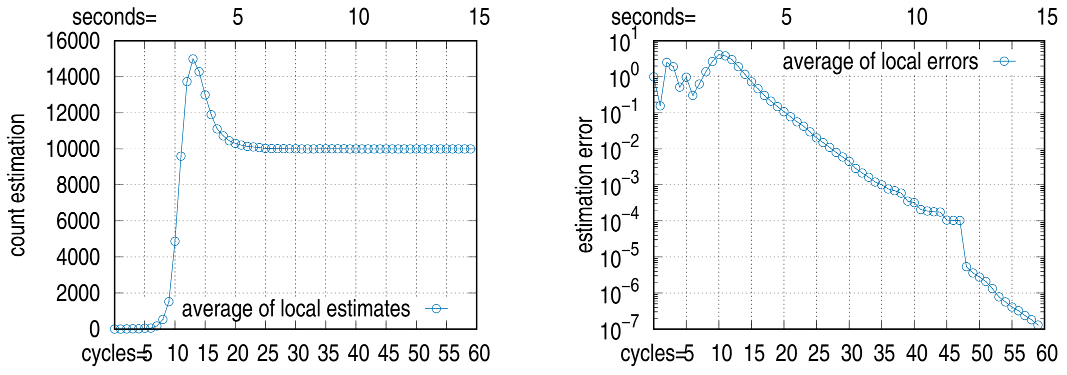


Figure 2.1: Push-Sum Protocol perform nodes counting under asynchronous setting, $N = 10^4$, $k = 30$, $\dot{\mathfrak{T}} = 250ms$, $t_{off} = 250ms$.

2.1.2 Push-Pull Protocol

There are several algorithms for the Push-Pull protocol, e.g. [28, 29, 58, 72]. However, the algorithm that is illustrated in this section is the simple form of the protocol introduced by Montresor A. and Jelasity M. [58]. The authors have proven that the protocol converges independent of network size and the increase in system size will not slow convergence down and will not increase resource requirements on particular nodes. They presented the smooth distribution of computation and communication over the system. However, the overall traffic increases linearly concerning system size. Push-Pull aggregation is efficient and provide results very quickly at all nodes due to the exponential convergence. In [27], authors have extended the protocol to propose practical solutions for proactive aggregation in dynamic environments accompanied by robust topology management. Algorithm 2 illustrates the protocol.

Push-Pull protocols exhibit better convergence due to the pairwise data exchange. Periodically, each node sends its current aggregated value to a random peer and waits for the response with the remote aggregate value. The aggregation function is applied to both values in sender and receiver nodes. Unlike Push-Sum protocols, this protocol does not use weight sum, imposing greater atomicity requirements on the interaction between node pairs. However, the version that adopts atomic-pairwise exchange of the protocol is inefficient for real-world systems because it requires synchronising cycles among nodes. The protocol does not converge under asynchrony. Hence, no results have been presented in this section.

Algorithm 2: Push-Pull Protocol

```
Require: a peer-sampling service, e.g. NCP$^+$.
Initialisation: $v_i = x_i$, $0 < i \leq N$.

1 At each cycle at node $i$:
2 $j \longleftarrow getRandomPeer()$
3 send $\langle v \rangle$ to $j$                    // a PUSH message to node $j$
4 receive $\langle v \rangle$ from $j$               // a PULL message from node $j$
5 $v_i = \frac{v_i+v_j}{2}$                          // update local pair

6 At event 'receive message $m$ from $j$' at node $i$:
7 send $\langle v \rangle$ to $j$                    // a PULL message to node $j$
8 $v = \frac{v+m.v}{2}$                              // update local pair
```

2.1.3 Symmetric Push-Sum Protocol

Symmetric Push-Sum Protocol (SPSP) is firstly introduced in [30]. It is asynchronous, non-atomic Push-Pull protocol that preserves the accuracy of Push-Sum protocols, achieves double convergence speed as any Push-Pull protocol, and provides means of configuration for various data aggregation functions. SPSP computes global aggregates in a decentralised manner and require no synchronous pairwise data exchange. The protocol SPSP used simple Node Cache Protocol (NCP) to select a peer from the system uniformly at random in every cycle (*uniform gossip*). NCP and its extended version NCP$^+$ are described in Section 2.2.1. The authors have adopted a fixed length cycle structure for the simulations. The cycle structure ensures no overlapping messages among different cycles and provides a simple mechanism for varying simulation parameters. Algorithm 3 shows the asynchronous pairwise data exchange in the protocol. A complete analysis of using the symmetric push-sum model in the global decentralised aggregation is provided in Section 3.2.

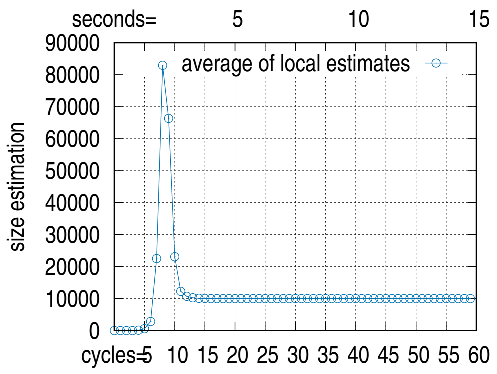
SPSP is a novel epidemic data aggregation protocol that can aggregate data on a system of any scale. Also, it is robust and preserves the mass conservation invariant in the absence of node or network failures. The protocol has been experimentally examined in the project, and results have validated the protocol, and it has outperformed other protocols for different aggregates and under asynchrony. However, further, inspection is required to study the performance of the protocol in dynamic and faulty networks. Due to the intrinsic features of SPSP, it has been used, and the symmetric push-sum style has been adopted in other research studies, for example, epidemic membership protocol proposed in the work of [68], and failure detection and consensus protocol in the work of [33], and it is adopted for this project as well. Figure 2.2 is a sample of the experiments on SPSP in the project. The figure illustrates the performance of the SPSP algorithm. The protocol is simulated under asynchronous setting: random dynamic overlay, cycles among different nodes may overlap, and messages typically arrive at different cycle.

Algorithm 3: Symmetric Push-Sum Protocol

```
Require: a peer-sampling service, e.g. NCP$^+$.
Initialisation: $v_i = x_i$; for average, $w_i = 1$; for summation $w_{\hat{\imath}} = 0$ at a single node $\hat{\imath}$
                and $w_i = 0$ at all other nodes; $0 < i \leq N$.

1  At each cycle at node $i$:
2  $j \longleftarrow getRandomPeer()$
3  $v = \frac{v}{2}$                                   // divide aggregation pair
4  $w = \frac{w}{2}$
5  send $\langle v, w, reply = true \rangle$ to $j$    // a Push message to node $j$

6  At event 'receive message $m$ from $j$' at node $i$ and time $t$:
7  if $m.reply$ then
8      $v = \frac{v}{2}$                               // divide aggregation pair
9      $w = \frac{w}{2}$
10     send $\langle v, w, reply = false \rangle$ to $j$   // a Pull message to node $j$
11 $v = v + m.v$                                       // update local pair
12 $w = w + m.w$
13 $e = \frac{v}{w}$                                   // local estimate of this step
```



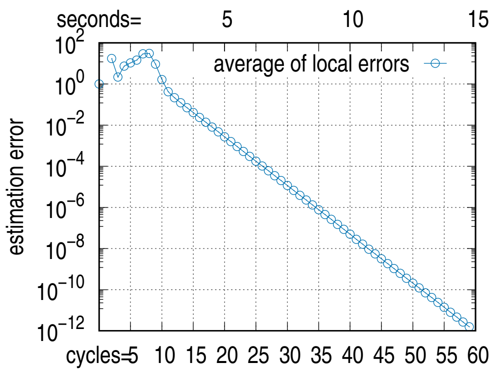


Figure 2.2: Symmetric Push-Sum Protocol, size estimation under asynchronous setting, $N = 10^4$, $k = 30$, $\dot{\mathfrak{T}} = 250ms$, $t_{off} = 250ms$.

2.1.4 Flow Updating Protocol

Flow Updating (FU) is an averaging based aggregation algorithm introduced by [73]. The algorithm tolerates high levels of message loss; a feature that was lacking in previous approaches. Message loss often violates the mass conservation invariant. The algorithm achieves faster convergence speed compared to other algorithms. Also, it is simple and can be executed in a synchronous model. In principle, the averaging Flow Updating protocol has a unique way of preserving the mass invariant; it keeps the distributed mass at every node in a variable rather than sending it in messages. The key idea is to use the exchange of the flow to perform update operations, which is the key to its unique robustness capabilities. Flow Updating makes accurate aggregations at all nodes and converges to the correct result.

The work in [74] brought to attention the vulnerabilities of averaging techniques when experiencing failures and dynamic networks. This limitation is not easy to fix in regular approaches. The authors introduce a dynamic version of Flow Updating algorithm were entries for neighbours are added or removed according to the current node participation. This simple design adapts to dynamic changes in a network with a good level of accuracy. Flow Updating outperforms previous strategies, and it operates continuously without requiring restarts.

Algorithm 4: Flow Updating Protocol-(*Unicast*)

```
Require: $\mathcal{D}_i$ set of adjacent neighbour nodes, $0 < i \leq N$.
Initialisation: flows: $f_{ij}, \forall j \in \mathcal{D}_i$, initially $f_{ij} = 0$; estimates: $e_{ij}, \forall j \in \mathcal{D}_i$, initially
        $e_{ij} = 0$; $v_i$ local value.

1  At each cycle at node $i$:
2  send $\langle f_{ik}, e_{ik} \rangle$ to $k$                         // a flow message to node $k$

3  At event 'receive message $m$ from $j$' at node $i$:
4  $f_{ij} \leftarrow -f_{ji}$                                      // state-transition
5  $e_{ij} \leftarrow e_{ji}$
6  $e_i = \dfrac{(v_i - \sum_{j \in \mathcal{D}_i} f_{ij}) + \sum_{j \in \mathcal{D}_i} e_{ij}}{|\mathcal{D}_i| + 1}$
7  $k \longleftarrow chooseRandomNeighbor(\mathcal{D}_i)$
8  $f_{ik} \leftarrow f_{ik} + (e_i - e_{ik})$                          // update
9  $e_{ik} \leftarrow e_i$
```

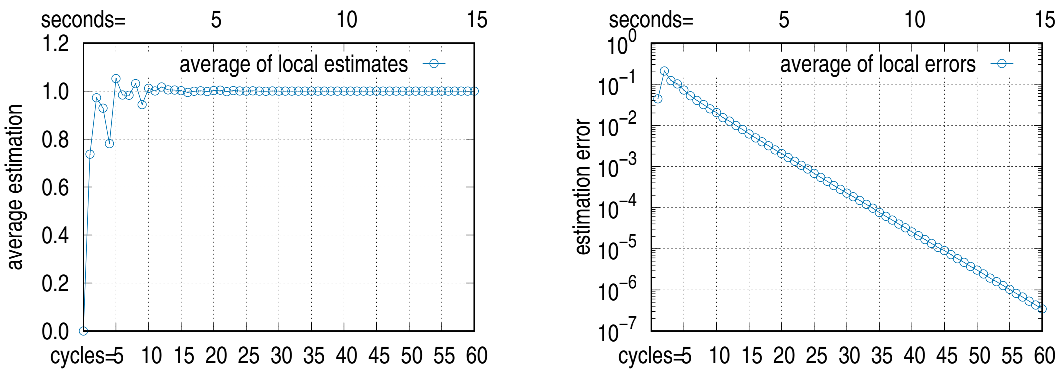


Figure 2.3: Flow Updating Protocol (*Unicast*), peek distribution averaging under asynchronous setting, peak value is N, $N = 10^4$, $k = 5$, $\dot{\mathcal{T}} = 250ms$, $t_{off} = 250ms$.

In the project, we investigated the protocol Flow Updating, Algorithm 4 presents the *Unicast*' version of the protocol. Figure 2.2 shows a sample of the experiments. The figure illustrates the performance of the FU algorithm when simulated under asynchronous setting: cycles among different nodes may overlap, and messages typically arrive at different cycle. However, it was not possible to simulate the protocol in dynamic topologies as it adopts static overlays.

FU algorithm is base on message exchanging with neighbour nodes (*neighbour gossip*). It requires local flows and estimates to be symmetrically pushed to every pair of neighbours, which requires cycle synchronisation in addition to maintaining symmetric links among neighbours. Global knowledge of the topology is assumed beforehand for Flow Updating to operate, and the protocol is limited to static topologies. Also, the work [75] has validated the protocol for achieving the aggregate average, and it is not clear how to apply other aggregation function.

2.2 Topology Management and Pear-Sampling Protocols

Peer-sampling services are critical to distributed systems and P2P networks. In large and extreme-scale systems, it is impractical for each node to maintain a complete list of membership for all other nodes in the system. Also, such a list is unachievable in dynamic systems where nodes may join or depart. The needed for synchronisation and maintenance makes a complete membership list at each node unrealistic approach [11, 33]. In epidemic systems, membership protocols usually form and maintain a partial view of the system at each node, which is used to provide the random node selection service. For applicability, membership protocols are typically epidemic services that use decentralised algorithms to process the partial views, and randomised communications to exchange local views. Another aspect of the distributed set of local views in a system is the formation of an overlay topology, which is periodically and randomly changes at every protocol cycle. The formation of overlay topologies independently of the physical networks is an efficient decentralised method for distributed nodes to identify and communicate with other nodes. Nodes can identify other peers using the links in the local view but communicate with peers via the underlying network.

Peer-sampling services and membership protocols are topology managers, and they can have a significant impact on the accuracy and robustness of the epidemic services. A topology manager can improve the quality of an epidemic service by forming adequate topology for the service. They can also recover and maintain the connectivity of the overlay topology [13, 16, 33]. Moreover, peer-sampling protocols can boost the quality of the service by detecting and excluding crashed nodes, and membership protocols can manage the impact of nodes joining or leaving a system for the interest of the epidemic service. It is impractical to assume an epidemic system without a robust peer-sampling service or membership protocol that helps the system services in some way or another.

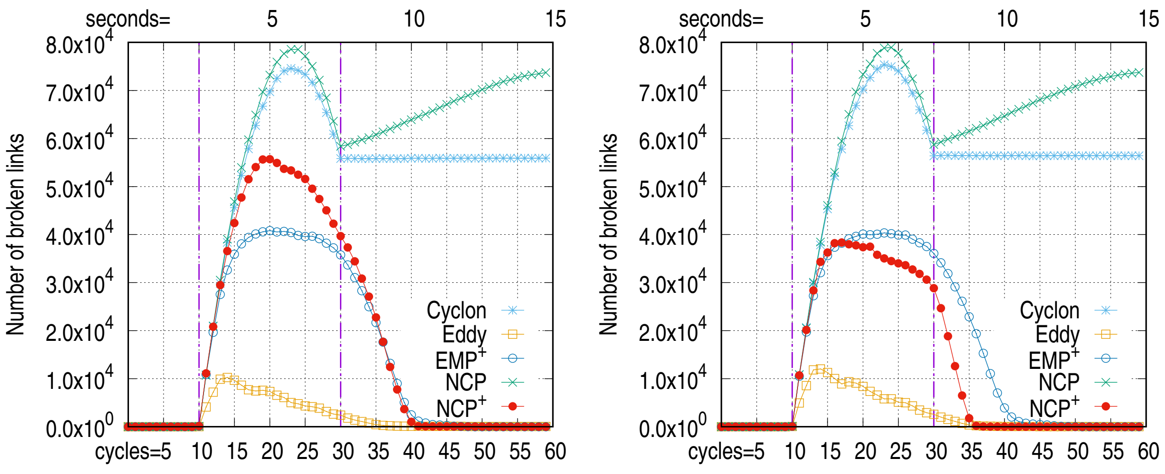
The design space of membership protocols were proposed by [11]. The author aimed to establish abstract model for peer sampling services to enable simple implementation of other membership protocols. There are many membership protocols for maintaining unstructured topologies in distributed and epidemic systems, For example, Cyclon [16], Eddy [76], Epidemic Membership Protocol-Plus (EMP$^+$) [33], and the protocol that has been improved and adapted in this work is the Node Cache Protocol-Plus (NCP$^+$). The protocol NCP$^+$ is a simple and lightweight overlay manager that produces random topologies. The protocol has been implemented to provide broken-links removal in addition to the inherited features of the NCP as introduced in [30, 33]. NCP$^+$ runs in parallel with other epidemic protocols and attains three main functionalities, forming unstructured and dynamic overlay, making a random selection of peers from the system (*uniform gossip*), and finally reducing redundant broken links in the overlay topology. However, apart from the randomised selection, NCP$^+$ has no particular mechanism for recovering or maintaining connectivity. For systems where connectivity is a concern, the protocol EMP$^+$ implements an innovative technique that can recover topologies after partitioning. In general, any membership protocol that generates unstructured topologies can be utilised for the epidemic protocols and services proposed in this project.

2.2.1 Node Cache Protocol-Plus (NCP$^+$)

Algorithm 5: NodeCache Protocol-Plus (NCP$^+$)

```
Require: maximum cache size $l^C$; default expiry time $\check{\mathfrak{T}}$; initial set of physical
         neighbour nodes $\mathcal{D}_i$.
Initialisation: at time $t0$ and at each node $i$:
         $C_i \longleftarrow \{\ell_1, \ldots, \ell_{l^C}\}$, where $\kappa_a \in \mathcal{D}_i, \ell_a = \langle \kappa_a, t0 + \check{\mathfrak{T}} \rangle$ and $0 < a \leq l^C$.
----------------------------------------------------------------
1  At each cycle at node $i$:
2  $j \longleftarrow getRandomPeer()$
3  send $\langle C, reply = true \rangle$ to $j$                  // send a copy of $C_i$ to node $j$
----------------------------------------------------------------
4  At event 'receive message $m$ from $j$' at node $i$ and time $t$:
5  if $m.reply$ then
6      send $\langle C, reply = false \rangle$ to $j$             // send a copy of $C_i$ to node $j$
7  $C_{temp} = C_i \cup C_j$
8  $C_i \longleftarrow \emptyset$
9  $C_i \cup \{\langle j, t + \check{\mathfrak{T}} \rangle\}$
10 while $|C_i| < l^C$ and $C_{temp} \neq \emptyset$ do
11     Randomly select and remove link $\ell$ from $C_{temp}$
12     if $\ell$ has not expired then
13         $C_i \longleftarrow C_i \cup \{\ell\}$
----------------------------------------------------------------
14 function getRandomPeer()          // public service to select a ranodm peer
15     Select a random link $\ell$ from $C_i$
16     return $\ell$                               // return random node information
```

The NCP$^+$ maintains a local cache C_i of fixed length l^C in each node i. The cache C_i is the local subset of the overlay and the local source for information about other peers, e.g. peers' identifiers. Typically, l^C is an application parameter presenting the overlay graph of k-regular initialisation. Links in C_i are uniformly distributed, and the protocol converges in $O(\log N)$ steps to uniform independent samples of the well-connected initial set of nodes [30]. Also, the NCP$^+$ provides the service '$getRadomPeer()$' to other epidemic protocols in the



(a) $\check{\mathfrak{T}} = 10$ cycles for all protocols.

(b) $\check{\mathfrak{T}} = 5$ cycles for all protocols.

Figure 2.4: NCP$^+$ and other membership protocols broken-links removing under churn. For all protocols, $N = 10^4$, $k = 30$, $\dot{\mathfrak{T}} = 250ms$, $t_{off} = 250ms$. Shuffling length in Cyclon, Eddy and EMP$^+$ is 24 entries. The minimum disparity among caches in Eddy is 3 entries.

same node such as SSEP and ECP to request a random link from the local cache for a peer node from the system. The protocol NCP$^+$ is illustrated in Algorithm 5.

In each cycle, each node exchanges C_i with a randomly selected peer. On the reception of a remote cache C_j in the node i, it merges the two caches into C_{tmp} such that $C_{tmp} = C_i \cup C_j$, and then it refreshes all entries in C_i by replacing them with other entries removed randomly from C_{tmp} until $|C_i| = l^C - 1$ or $|C_{tmp}| = \emptyset$. Also, NCP$^+$ merges an entry for the sender node j hence $C_i \cup \{j\}$ as a *refreshing mechanism* [13, 33], which principally mean that node sending a PUSH message is an alive node. This way, active nodes in the system constantly get their identifiers injected in the cache C_i with a fixed expiry time ($\check{\mathfrak{T}}$). Also, expired identifiers in the merged cache C_{tmp} are detected and removed. Thus, broken links are gradually removed from the system. However, broken-links removal is not instantaneous in NCP$^+$, and under severe churn, some broken links may remain in the system despite their expiry.

The NCP$^+$ and some other protocols are simulated, and their performance in removing broken-links is examined[1]. In the experiments, 60% of system nodes are enforced to fail between cycles 10 and 30, and using an external observer, the number of broken links is aggregated in every cycle. Figure 2.4 shows the number of broken links in the system during and after the dynamic conditions. In Figure 2.4.a, the NCP$^+$ outperforms the classical NCP and Cyclon; however, broken-links removing mechanisms in EMP$^+$ and Eddy protocols show better results. The results in Figure 2.4.b presents an improvement in NCP$^+$ performance due to using shorter expiry time. In general, EMP$^+$ and Eddy are very complex protocols and require special configurations, whereas NCP$^+$ is simple to implement and use.

Although the peer-sampling services and membership protocols can improve epidemic tasks quality and maintain overlays connectivity, they cannot guarantee optimum robustness

[1]Credits to Dr *Pasu Poonpakdee* for providing prototypes for Cyclon, Eddy, and EMP$^+$ protocols from his work in [33].

or eliminate faulty links. Epidemic tasks such as data aggregation are typically got damage due to the violation of the mass-conservation invariant caused by nodes failure and churn. Furthermore, some epidemic services may wish to use lightweight membership protocols that is simple rather than robust for the sake of optimisation, e.g. decentralised services in OSN, IoT, and agent-based systems. Therefore, the robustness of the data dissemination and aggregation is still essential and require further research, and with the help of robust peer-sampling services or membership protocols, epidemic services may achieve the optimum robustness.

2.3 Eventual Consistency and Distributed Consensus

The work of Demmer [23] mentioned earlier in this chapter might be among the earliest epidemic algorithms for a consensus among replicas, for which the agreement process is based on randomised information dissemination. Also, authors in [56] presented a comprehensive analysis and provided transmission and computation complexities of the randomised information dissemination. The shared key point is the convergence, which is guaranteed by the theoretical analysis under a few assumptions. The work of [46] proposed gossiping algorithm to achieve consensus, and presented a theoretical analysis of the performance of the algorithm under crash failures. The study shows that the gossip consensus algorithm can tolerate f crashes and achieve consensus in $O(f)$ time and $O(n \log f)$ messages. Authors in [77] proposed an averaging protocol for the consensus. The protocol is based on the data propagation among neighbour nodes in WSN. Through analytics, the paper characterised the convergence of the broadcast averaging protocol, which approved to be scaling better than pairwise averaging. Authors in [78] presented theoretical results for the consensus under dynamic topology and variable communication in multi-vehicle cooperative control.

In [79], authors reviewed the average consensus problem and presented probabilistic consensus over large scale network. The work considers algorithms which do not converge to the average, but, under certain circumstances, to some good approximation of it. In the analysis, assumptions such as symmetric rounds and communications are released. However, this led to asymmetric gossip that does not preserve the exact global average. The analysis in [48] studied a variety of consensus and averaging algorithms, taking into consideration topology changing. It proves that averaging algorithm (agreement algorithm) guarantees asymptotic consensus under semi-static topology, but, the agreement and its convergence may suffer severely in a fully dynamic topology.

Authors in [39] proposed gossip-based protocol to achieve eventual consistency in the total ordering of replica updates. However, their work interestingly differs because they do not perform consensus or explicit agreement. In essence, they proposed a gossip-base protocol that disseminates and delivers the update information without ensuring the global order of the updates. Thus, some old updates may arrive after having subsequent updates applied. In this case, the replica has to roll-back, apply the past update, and re-apply all followed updates. The authors showed that reaching consistency without storing all updates is possible with high probability. A direct advantage is replicas need not too aware of the existence of some old updates as they are undelivered messages. Also, nodes can achieve eventual consistency in an asynchronous system due to releasing the need for consciousness on the global ordering.

The work in [40] facilitates the eventual consistency paradigm as an alternative solution for distributed systems, although safety is not assured. The paper describes the eventual consistency as achieving convergence by exchanging information that can take any form, e.g. broadcast, flooding or pairwise. It also lists characteristics of the systems that adopt eventual consistency such as simplicity and relaxed constraints, e.g. communication latency consideration. The paper has explained that with eventuality, the *safety* property of systems cannot be guaranteed. For instance, algorithms and protocols which eventually converge to a consistent state, or requests that ultimately receive a response; have no certainty on what may finally happen. This property supports the hypothesis of the project, in which more steps are needed to acquire assurance and maintain systems safety. Moreover, the paper illustrates several examples of the real-world systems where eventual consistency has been applied.

The approximate consensus is introduced in [80], and it is a paradigm of achieving efficient rather than exact common distributed state, aiming for optimisation of the agreement process in the large-scale systems. In a recent study [49], the analysis proposed the agreement through averaging to achieve the approximate consensus in structured dynamic networks, i.e. *trees*. The authors concluded that achieving approximate consensus under dynamic conditions is more suitable than the exact consensus because it does not require reliable connectivity. Another practical work addressed the approximate consensus is proposed in [81]. The work proposed epidemic algorithm for the consensus on failure detection, in which communication is traded for consensus detection accuracy. In the algorithm, the awareness of failure detection is tolerated to reduce communication for the consensus.

The selected studies presented in this section demonstrated the wide spread of using the global averaging as a synonym of the global agreement. Many of the studies analysed and inspected the averaging process under different assumption and conditions while considering the agreement process. It might be a standard style of research to represent distributed consensus in the form of the aggregation of global average. However, it is vital to manifest that the achieved consensus is probabilistic, and additional steps are needed to attain the explicit consensus. Further review on the agreement process through distributed coordination is presented in Section 4.1.

2.4 Impossibilities in Real-world Systems

The correct addressing of the environment for which epidemic algorithms are implemented is essential for the efficiency and the applicability of those algorithms. In realistic conditions, epidemic systems are asynchronous and fault-prone similar to any other distributed system. Typically, systems asynchrony refers to the absence of bounds in processes execution times and communication delays [1]. Also, the potential failure of a node or a network is a norm rather than an exception. In consequence, the real-world model of distributed systems adopts no assumptions about time intervals of any execution or transmission, which eventually limit or prevent algorithms performance in such systems.

The common behaviour in distributed systems that nodes may fail or lose communication. In consequence, the impossibility result presented in [82] (a.k.a. the FLP result) refers to the impossibility of detecting consensus (or agree on a consistent state) in fully asynchronous and unreliable distributed systems. Although the FLP result was presented for deterministic asynchronous systems, it has formed a notable matter for the researchers [83]. The result showed that the termination condition in the consensus problem is unachievable even with the presence of a single node failure under full asynchrony. The unachievable termination occurs due to the inability to distinguish a failed node from an undelivered response as a result of communication delays.

The FLP result has motivated the identification of the minimal properties of distributed systems that are necessary to solve the consensus problem [84]. In particular, the adoption of randomised communication reduces the potential causes of the impossible case [85]. Also, the global agreement can be achieved in a partially synchronous model with prior defined bounds on the communication delays and process execution times [44, 86]. The systems supported by a failure detection service can also attain the consensus [86, 87, 88]. Interestingly, asynchronous protocols that can be adapted to terminate efficiently in practice can agree on a consistent state too [44, 86, 89].

Epidemic systems naturally exhibit fault tolerant and randomisation properties. They consist of decentralised computations with randomised P2P communications, and thus, they lack the disastrous single point failure and implement significantly efficient diffusion process [22, 26]. Moreover, epidemic protocols eventually compute a common value with high probability which implies the ability to satisfy the termination condition in consensus problem [27, 90].

It is apparent that algorithms and protocols in epidemic systems are by nature capable of attaining the consistency mainly due to their ultimate convergence property. However, the convergence property is attainable in ideal system conditions [22, 26]. In dynamic conditions, a system cannot converge to the true target but probably converge to an approximation of the target [26, 27, 34]. This behaviour is due to the disturbance in '*mass conservation*' property which is the aggregate of all initial values in a system. The quality of the convergence depends on the level of damage or loss in the initial system mass. Because the detection of convergence is essential in acquiring consistency, it is also vital to ensure the quality of the convergence, especially, in dynamic conditions. However, it is impossible for a system to converge to the exact target under dynamic conditions.

In general, epidemic protocols are required to tolerate a small error value in the local estimates to be able to detect convergence locally [32, 91]. The detection methods are though heuristic and the tolerated error value is a tuning parameter which can be adjusted as desired. However, the remaining problem is how to improve the performance of epidemic protocols to attain good approximation of the target in dynamic conditions? The answer prepares the way towards the reliable achieving of the consistency in unreliable systems.

The research work in this thesis adopts partial-synchrony settings following the work in [44], in which process execution times have a fixed bounding, and an upper bound on the communication delays is defined but it is unknown to the system. Furthermore, the research project has proposed heuristic and explicit convergence detection methods which satisfy the termination condition in the consensus problem. In addition to proposing robust aggregation algorithms that can converge under dynamic conditions to a good approximation of the target.

2.5 Model of Epidemic Systems

The model of the epidemic system consists of a large number of nodes N. Nodes are connected to the Internet as part of an application or a service. Each node is assigned a unique identifier (e.g. IP address) and communicates using pairwise message exchanging with another peer node selected uniformly at random from the system (a.k.a *uniform gossip* [56]). Epidemic protocols in the system adopt the asynchronous model of SPSP, which is described in Section 2.1.3. The pairwise exchange model of SPSP is non-atomic Push-Pull scheme that does not lock for a response after sending a Push message. Messages order is not guaranteed and a node may receive multiple Pull messages in one cycle, and hence, messages interleaving is present at all times [28, 33].

Protocols in each node generate a Push message in each '*Cycle*' (a.k.a *gossip period* [50]). A cycle start is entirely a local node decision and each node follows its own clock. Time drift among nodes clocks is assumed to be small and negligible. Cycles are time intervals of fixed length denoted ($\dot{\mathfrak{T}}$). Subsequent cycles do not overlap. However, among various nodes, cycles may overlap and global cycle synchronisation is not enforced. The parameter $\dot{\mathfrak{T}}$ is typically set to be greater than the Round Trip Time (RTT) on the diameter of the network (e.g. the Internet), and the interval between two consecutive cycles is sufficient for the delivery of most messages with high probability [13, 29]. Although the parameter $\dot{\mathfrak{T}}$ is relatively long, messages may experience longer delays and may cross time boundaries among cycles (a.k.a out-of-cycle messages [29, 92]), and some messages may have infinite delays. More details on cycle length settings are provided in Section 3.5.

In addition to the Internet physical topology, the system comprises a dynamic overlay topology that is maintained for the decentralised identification and pairwise communication among nodes. The overlay network is formed by a peer-sampling service that provides each node with a subset of peers from the system (a.k.a *neighbours*). Nodes can identify other peers using the local links in the subset but still communicate via the underlying network.

The model assumes a random, dynamic, and directed overlay topology with adequate uniform degree distribution and sufficient connectivity among nodes. The presumed peer-sampling service should perform three general tasks: (1) forming an unstructured and dynamic overlay, (2) providing a communication peer from the system independently and uniformly at random (*uniform gossip*), (3) reducing redundant broken links in local views of the overlay in the presence of dynamic conditions. Practically, the Node Cache Protocol-Plus (NCP$^+$) is adopted as an epidemic peer-sampling protocol that runs in parallel with other epidemic protocols in the system. Any other peer-sampling service and membership protocol can also be utilised, e.g. [33]. The NCP$^+$ is further explained in Section 2.2.1.

From another perspective, the underlying network is assumed connected and reliable unless otherwise stated in the text. In this regard, nodes of the studied system are connected to the Internet through a reliable transport protocol (e.g. TCP) [27, 33]. In consequence, all exchanged messages are eventually delivered, and the reliable delivery of messages limits the expected types of failures in the system to nodes failure and nodes churn.

The research in the project addresses dynamic epidemic systems that comprises N nodes in a particular time t. As time evolves, nodes simultaneously join and depart the system in a collective behaviour composing the churn. Nodes departure is not specified either being voluntary or adversary and both cases are treated uniformly. The fractions of joined and departed nodes in a specific time interval Δt define the churn rates in the system. Distribution of nodes arrivals or departures over Δt varies from a network type to another. The rates of nodes churn in P2P is discussed in Section 5.1. Malicious behaviour of nodes and Byzantine failure are not considered in the present research, however, the reader can refer to [60] for detecting and removing misbehaviour in the epidemic systems.

2.6 Summary

This chapter has presented several studies from the literature reviewed in the early stage of the research project. Initial studies on epidemic algorithms and protocols for the distributed systems are mentioned. The review gradually listed the related work from the oldest up to the most recent and relevant. Also, practical review for epidemic protocols is provided to demonstrate the foundations of the following work of the research. A number of selected studies on the eventual consistency and the distributed consensus are briefly explained. The chapter ended describing the impossibilities in real-world distributed systems and the model of epidemic systems.

In essence, epidemic algorithms and protocols proposed in this project adopts the asynchronous model of SPSP, and take advantage of its intrinsic properties. The protocol NCP$^+$ has been developed and improved to provide peer-sampling service to the epidemic system in the project. In theory, the convergence in consensus and consistency is guaranteed; in practice, the achieved agreement is probabilistic and lack of global awareness on the convergence. The safety property can not be assured in systems adopting eventual consistency and approximate consensus techniques. Undoubtedly, further work is needed to attain explicit system consistency. Epidemic algorithms and protocols are randomised, fault-tolerant and guaranteed to converge in stable systems, and thus, they satisfy the termination condition for the consensus in asynchronous distributed systems. However, the performance of the epidemic algorithms under node churn and network failure require additional research. Epidemic systems in this project adopt a partial-synchrony model with reliable underlying network and bounded delays. The model does not impose messages to arrive in the same processing cycle and does not enforce cycles synchronisation. The model adopts equivalent asynchrony to real-world systems.

Next chapter describes in details the convergence of asynchronous epidemic data dissemination and aggregation. Also, the chapter discusses the convergence detection methods.

Chapter 3

Convergence Detection in Epidemic Systems

The convergence of epidemic protocols is a descriptive term to define the moment in which local states of nodes approaches the desired target [31, 93]. Although the convergence is guaranteed through the theoretical analysis, the practical consideration of epidemic protocols requires each node to detect the convergence locally. The detection of convergence is critical in epidemic systems; for instance, it is needed to measure total communication overhead during an aggregation task [51]; also, it can be used to minimise the aggregation time via adjusting the accuracy threshold [94]. Typically, a good quick estimation might be in favour than an accurate one that takes longer to compute. It is also a prerequisite for the execution of a designated action or decision such as termination or restart [27, 93], and it has a substantial role in achieving system convergence [32], and achieving global agreement [52, 92], i.e., the global consistent states. Furthermore, it is a fundamental element for acquiring consistency in distributed systems [40]. In general, the decentralised detection of convergence can significantly improve the overall performance and efficiency of epidemic algorithms and systems.

This chapter analyses epidemic data dissemination and aggregation processes, and characterises the convergence model of each process. It discusses the conventional heuristic detection methods in the context of real-world epidemic systems, which are asynchronous and dynamic. It also introduces and describes our developed method for local convergence detection, which does not require any system information in advance. Additionally, it discusses the need for an explicit detection method for the system convergence. Detection methods presented in this chapter are the main foundation's blocks in the novel protocols of the project. The chapter ends, illustrating the convergence time from a practical point of view and discusses mapping convergence time to true-time units, especially in the presence of asynchronous data exchanging and processing.

3.1 Convergence of Dissemination Process

In information dissemination process, convergence to the desired state is a convergence of content and is accomplished when all nodes in a system have received a copy of an information [25, 31]. Information can take any form of data that can be in the interest of others, for example, news, updates, failure notification or failures count, etc. Basically, a node generates an information item and starts a diffusion process into the system for that item. The diffusion process may use one of the common propagation models such as *broadcast*, *flooding*, or *gossiping*. Generally, convergence speed and time are usually affected by the propagation model; however, the convergence eventually consists of the reception of information in each node. The convergence in the dissemination process is found as promising and efficient as nature diffusion of epidemics [23].

Definition 3.1.1. The convergence of epidemic data dissemination consists of receiving particular information content by each of the system nodes.

On the other hand, the detection of local convergence in information dissemination process is straightforward. Each node considers the reception of an information item as achieving convergence on that particular item [23, 56]. However, one essential problem is to make global convergence of an information item in epidemic systems [52, 95]. For example, consider the total ordering of updates in a system of distributed replicas, in which it requires each update information to be applied at most once at each replica and any two updates to be applied in the same order at all replicas. In practice, formulating a universal conclusion on the dissemination of particular update information or its global order is a complicated problem and theoretically impossible [82]. It is not instantly feasible to system nodes in which order an update has been generated until each node receives a new update that is in prior order than the one it already has.

However, eventual consistency algorithms apply weaker assumptions and provide efficient methods for ordering updates [23, 39]. They provide guarantees that the contents of distributed replicas eventually converge if they can independently communicate for a long enough period. Despite the assurance of eventual consistency algorithms, they have a similar downside of the heuristic determination of convergence as described earlier in this thesis. The attaining of consistent state on an information item requires the explicit detection of convergence and agreement on that particular item. Section 4.2 introduces an innovative epidemic protocol to achieve global convergence and agreement in information dissemination.

3.2 Convergence of Aggregation Process

The common fundamental characteristic of distributed data aggregation tasks is the ultimate convergence of the resulting aggregates to the desired target. The convergence feature formed the baseline of many decentralised services such as the computing of system-wide properties, reaching consensus, and acquiring consistency [27, 39, 49]. In the literature, data aggregation algorithms are extensively studied and the theoretical and practical analysis in the studies have provided guarantees on the convergence of the aggregation results [26, 28, 48, 50, 58]. Although the achieving of convergence is independent of the overlay topology and the underlying routing, the convergence speed, robustness and quality are usually influenced and may differ from a network to another. Therefore, the determination of the moment of achieving convergence is essential to identify convergence properties. In general, the formation of convergence expands along the aggregation process which starts from the initialisation of distributed data in each participating node to the moment of which each node computes a close approximation of the desired target.

In a data aggregation process, a numeric value is usually representing the desired target which is the intended numeric outcome of the computation in each node, and it is denoted in the thesis as $\mathcal{V}$. In general, a data aggregation process ultimately converges to the target value $\mathcal{V}$. However, $\mathcal{V}$ usually corresponds to the range of initial data values in a system and to the applied synopsis function (e.g., *average*, *sum*, *max*, *sample*, etc.). For example, in the leader election problem, a protocol may wish to compute the maximum of initially distributed values or the largest value among nodes identifiers [51, 92]. Thereby, the target is the leader node value or leader node identifier. Also, the intended $\mathcal{V}$ for an aggregation process in a network of climate sensors differs from a target of the aggregation process for a system size. There are many other data aggregation protocols that eventually converge to the intended target in the literature, for example calculating system size, resource capacity, and average uptime [26, 27].

Each node i participates in a data aggregation process, typically starts with an initial value $x_{i,t0}$ where $0 < i \leq N$ and $t0$ is the start time. As time evolves, each node performs a pairwise data exchange with one or more peer nodes using a particular communication model. Each node then updates a local estimate $e_{i,t}$ with a fresh value computed by a specific synopsis. Formally, an aggregation process achieves convergence when $e_{i,t}$ in each node i get arbitrary close to $\mathcal{V}$.

Definition 3.2.1. The convergence of epidemic data aggregation consists of computing $\mathcal{V}$ by the local estimate $e_{i,t} = f()$ at each node i. The convergence typically consists after some evolved time t when $e_{i,t} = \mathcal{V} \pm \varepsilon_{i,t}$, where $\mathcal{V}$ is the desired target value, $e_{i,t}$ is the local estimate at node i, $f()$ is the global aggregate function (e.g., *average*, *sum*, etc.), and $\varepsilon_{i,t}$ is a local small error that can be tolerated.

The model of pairwise exchange that is adopted for protocols in this project is the symmetric push-sum, which can be initialised to accomplish several data synopsis functions, specifically, for the averaging and summation tasks [30]. This analysis describes the formation of convergence in the adopted aggregation model. Initially, each node i starts with a local value x_i where $0 < i \leq N$. Also, each node maintains a tuple for a pair of aggregates $(v_{i,t}, w_{i,t})$. The value $v_{i,t}$ is the data aggregate and initially, it is set to $v_{i,t0} = x_i$, whilst, the *weight* $w_{i,t}$ is a determinant that depends on the required aggregate function [26, 30]. For averaging problems, the initial weight is ideally set to $w_{i,t0} = 1$ in all participating nodes and so by the design each node should converge to $w_{i,t} \approx 1$. However, in summation problems, the weight is initially set to $w_{\hat{\imath},t0} = 1$ at a single designated node $\hat{\imath}$ that can be predetermined, elected or randomly selected [26, 30], and the weight in other nodes is set to $w_{i,t0} = 0$ where $0 < i \leq N$ and $i \neq \hat{\imath}$. The weight at each node in a summation problem ultimately converges to $w_{i,t} \approx \frac{1}{N}$.

At each cycle, a node i divides its aggregate pair into to halves, retains the half $(\frac{v_{i,t}}{2}, \frac{w_{i,t}}{2})$, and sends the other half in a Push message to a random peer j. Upon the reception of a Push message from node i, node j divides its aggregate pair by 2, retains a half, and responds to node i by sending the other half in a Pull message (or symmetric push message). Next, both nodes i and j update their local aggregate tuple, for instance, the node i performs $(v_{i,t}, w_{i,t}) = (v_{i,t} + v_{j,t}, w_{i,t} + w_{j,t})$, and computes a fresh local estimate $e_{i,t}$ using $e_{i,t} = \frac{v_{i,t}}{w_{i,t}}$. Eventually, a node achieves local convergence when $e_{i,t}$ get arbitrary close to $\mathcal{V}$ and an aggregation process converges if local estimates in the participating nodes get close to $\mathcal{V}$.

Another fundamental property for a consistent aggregation process and for a robust convergence is the '*mass conservation*' invariant [26]. This property must remain constant during the aggregation process, and merely it is the aggregate of all initial values of participating nodes,

$$\mathcal{M}v = \sum_{i=0}^{N} v_{i,t0}, \qquad \mathcal{M}w = \sum_{i=0}^{N} w_{i,t0}, \tag{3.1}$$

however, during the aggregation process, the pairwise data exchanging diffuses and aggregates the initial system mass $\mathcal{M}v, \mathcal{M}w$. After a sufficient number of cycles at time tc, the system mass is distributed evenly over all participating nodes,

$$(v_{i,tc}, w_{i,tc}) = (\frac{\mathcal{M}v}{N}, \frac{\mathcal{M}w}{N}), \qquad 0 < i \leq N, \tag{3.2}$$

and hence, tc is the moment of achieving convergence in the aggregation process. Thereafter, each node can approximate the target value through the local estimate as follows $e_{i,\hat{t}} = \frac{v_{i,\hat{t}}}{w_{i,\hat{t}}}$ where $\hat{t} \geq tc$. After achieving convergence, the exchanging of aggregate pairs and the computation of local estimates at all nodes always give an approximation of $\mathcal{V}$ and this behaviour remains the same as long as system mass invariant is not violated [26, 34, 51].

In principle, the typical aggregation of the target value can be computed by $\mathcal{V} = \frac{\mathcal{M}v}{\mathcal{M}w}$. From that, the target value for the averaging is computed by $\mathcal{V} = \frac{\sum_{i=0}^{N} v_{i,t0}}{N}$ and the target in the summation process is given by $\mathcal{V} = \frac{\sum_{i=0}^{N} v_{i,t0}}{1}$. Also, the *count* aggregate function is a special summation process in which initial data values are set to $v_{i,t0} = 1, \forall i, 0 < i \leq N$ and hence, the target of the counting function is $\mathcal{V} = \frac{\sum_{i=0}^{N} v_{i,t0}}{1} = N$. From this perspective, the counting aggregate function is fundamental in the decentralised estimation of system size. The System Size Estimation Protocol (SSEP) is a typical application of the counting aggregate function and it is explained in Section 4.3.

The analysis of convergence in the data aggregation process indicates that decentralised detection of convergence is achievable through monitoring of local estimates. Every node can detect the convergence locally by verifying how close the estimate $e_{i,t}$ is to the target $\mathcal{V}$. However, in more realistic scenarios, global information about system initialisation such as the target $\mathcal{V}$ or the size N may not be available and thus obstructing the detection of the convergence. The remaining of this section inspects the convergence detection methods taking into consideration some practical issues encountered during the research project.

3.3 Local Convergence Detection

The detection of local convergence in data aggregation process is usually heuristic and requires some application-specific parameters. In a general method [51, 92, 94], a *tolerance threshold* (a.k.a *accuracy threshold*) is used to evaluate local estimates accuracy with respect to $\mathcal{V}$. Tolerance threshold is a global parameter that is determined according to application preference, and it is used to tolerate or control the *estimation error* (a.k.a *residual error*) in the local estimate. The convergence detection should consist only upon verifying the estimation error became as small as desired. The method also includes the counting of the number of consecutive cycles in which an estimation error is under the tolerance threshold. This is principally important due to the fluctuation in local estimates during the aggregation process that may cause precocious detection of the convergence [51, 92].

In the general method for local convergence detection, each node i usually (1) computes the estimation error $\varepsilon_{i,t}$ in the estimate $e_{i,t}$ at every cycle t using the typical formula

$$\varepsilon_{i,t} = \frac{|e_{i,t} - \mathcal{V}|}{\mathcal{V}}, \tag{3.3}$$

and the method monitors the error $\varepsilon_{i,t}$ approaching the given tolerance threshold ϵ as a result of variance reduction in the aggregation process, and (2) it verifies the criterion $\varepsilon_{i,t} < \epsilon$ until it becomes *true* for a presumed number of consecutive cycles Υ, which after confirming it the detection of convergence consists.

Theorem 3.3.1. *There is time tc such that for all $t \geq tc$, the local estimation error $\varepsilon_{i,t}$ at each node i is at most ϵ for a number of consecutive cycles greater than or equal to Υ.*

At the time tc, each node can locally detect the convergence by verifying the local estimation error to the given thresholds ϵ and Υ. The complete proof of this theorem presented in these studies [26, 51].

The parameters ϵ and Υ are used in the general method and they are application-specific parameters, which are provided beforehand to data aggregation protocols. However, the practical consideration of the detection method and the control parameters reveal some issues that may limit the applicability of epidemic protocol in real-world systems. There are two primary issues to address here, first, the evaluation formula that calculates the error $\varepsilon_{i,t}$ in the detection method requires the knowledge of the target $\mathcal{V}$ in advance. Second, the determination of correct values for the parameters ϵ and Υ also demands prior information about the initialisation and the size of the system. Notably, the parameter Υ is tricky; if it is set to a smaller value than the necessary number of cycles for a particular system, a protocol may quickly detect a false convergence. Whilst a larger value of the parameter Υ may introduce extra delays and hence penalising the protocol performance [51, 92].

Although control parameters are application-specific, they have a vital role in the convergence detection in data aggregation protocols. Therefore, the utilisation of the correct settings for a particular system is essential for the efficiency and the robustness of protocols in that system. However, system-wide settings should not request any global information because usually, this information is unavailable or hard to obtain, especially in realistic conditions. For example, it is very challenging to provide beforehand a precise size value in a large dynamic system, which in consequence may force using approximated values for the control parameters and so influence protocols performance. In general, the reliance on any global information that might restrict or limit epidemic protocols' performance should be minimised. One way to achieve this aim is to use system-independent values or scales for the control parameters.

The work of Poonpakdee et al. [94] proposed a novel method for local convergence detection which has been adopted and improved in our work in [92]. The technique in Poonpakdee's method adopts a moving-average mechanism to analyse the latest estimate through a sliding subset of running estimates. Technically, each node i maintains a history queue $\mathcal{Q}_i$, upon the reception of a message from node j, the receiver node i enqueues its local estimate and the received estimate, so $\mathcal{Q}_i = \mathcal{Q}_i \cup \{e_{i,t}, e_{j,t}\}$. The queue $\mathcal{Q}_i$ has a fixed length $l^{\mathcal{Q}}$ and hence, it dequeues two elements when $|\mathcal{Q}_i| = l^{\mathcal{Q}}$. In each cycle t, the node i calculates a statistical synopsis for the estimates in $\mathcal{Q}_i$, in particular, the Root Mean Square Error (RMSE) is used in the original version of Poonpakdee's method. However, the utilisation of the RMSE requires a reference to the target value $\mathcal{V}$, for which requesting global information about the system. Poonpakdee has attempted to substitute the reference to $\mathcal{V}$ by using the local estimate at each node, based on the assumption that all nodes will eventually converge to the target value. It is apparent that the previous assumption is not valid for dynamic systems as described earlier in the thesis.

$$\varepsilon_{i,t} = \sqrt{\frac{1}{|\mathcal{Q}_i|} \sum_{e \in \mathcal{Q}_i} (e - e_{i,t})^2}, \tag{3.4}$$

We developed Poonpakdee et al. method and proposed a new version that implements the Coefficient of Variation (CV) instead of the RMSE. The CV is computed using the following formula:

$$\varepsilon_{i,t} = \frac{\mathcal{Q}_i.s}{\mathcal{Q}_i.\bar{e}}, \tag{3.5}$$

where $\mathcal{Q}_i.\bar{e}$ is the average of estimates in $\mathcal{Q}_i$ given by

$$\mathcal{Q}_i.\bar{e} = \frac{1}{|\mathcal{Q}_i|} \sum_{e \in \mathcal{Q}_i} e \tag{3.6}$$

and $\mathcal{Q}_i.s$ is the standard deviation calculated by

$$\mathcal{Q}_i.s = \sqrt{\frac{1}{|\mathcal{Q}_i - 1|} \sum_{e \in \mathcal{Q}_i} (e - \mathcal{Q}_i.\bar{e})^2}. \tag{3.7}$$

The new method for convergence detection has eliminated the need for prior knowledge about the target $\mathcal{V}$ in the evaluation formulas in contrast to the general formula in 3.3. The implementation of CV in the method has improved the detection accuracy especially in data averaging problems. Also, the practical analysis of the protocols that implemented the new method has provided useful information about convergence properties especially in terms of speed and quality. Moreover, the CV typically delivers outputs in the relative measurement units which provides a unified setting for the proportional and unit-less comparisons. For instance, a constant percentage can be applied as a tolerance threshold to analyse convergence quality among different epidemic protocols or to analyse aggregation problems in various system sizes or under different data initialisations. The unified settings determine the default parameters values that can be applied in the majority of epidemic systems upon the actual implementation.

Although the utilisation of the CV in the convergence detection methods is preferable on the usage of the RMSE, some aggregation problems may impose the use of absolute metrics in the detection methods, particularly, the aggregate of counting in dynamic epidemic systems [34]. For example, the correctness of the system size estimation protocol requires a minimum tolerance threshold since the lower the residual error, the higher the size estimation accuracy. Practically, setting of the threshold ϵ with the universal relative value may tolerate a substantial error in the equivalent quantity, and thus the protocol may achieve an incorrect convergence detection.

To clarify, we had a case in validating the proposed protocols in this project, in which the CV was inappropriate. Assume a system size aggregation task, also, assume the task wishes to measure estimation error caused by system dynamics. In the task, the convergence detection method uses the CV and sets the parameter ϵ to the ratio 1%. Upon the detection of local convergence, size estimation in each node will tolerate 1% of actual system size due to the tolerance threshold. In this case, it is ambiguous to decide whether the source of error in local estimates is detection method or nodes churn. Consequently, the parameter ϵ should be very precise to minimise the tolerated error. In similar scenarios, the detection of local convergence with an accurate size estimation demands a correct adjustment for the tolerance threshold ϵ for that particular system, which means requesting size information beforehand. Also, dynamic systems may require reasonably continuous adjustment for the threshold ϵ to cope with the changes in the system size.

To overcome the inadequacy of using the CV in the validation of epidemic protocols under dynamical conditions, and to propose the option of using absolute values in threshold settings, the Standard Error (SE) is used instead [34]. Technically, the new method reuses the history queue $\mathcal{Q}_i$ in each node i and replaces the computation of the CV with the calculation of the

SE for the buffered estimates, and the local estimation error is given by:

$$\varepsilon_{i,t} = \frac{\mathcal{Q}_i.s}{\sqrt{l^{\mathcal{Q}}}}. \tag{3.8}$$

where the $\mathcal{Q}_i.s$ is the standard deviation provided by the formula 3.7 and $l^{\mathcal{Q}}$ is the fixed length of the queue $\mathcal{Q}_i$ as described earlier.

Like the CV, the calculation of the SE does not require any reference to the correct target value in its formula. Also, it gives less uncertainty in error measurement around the mean in comparison to variance and standard deviation. Moreover, the utilisation of the SE in the convergence detection method has allowed the settings of the tolerance threshold ϵ to the absolute values, which is an advantage as it can generally be the minimum well-known value. In practice, the detection method with the SE was sufficient for summation aggregation protocols in general, and system size estimation in particular [34]. The detection method is used with constant control parameters across several experimentations of various system sizes. For instance, the tolerance threshold is set to $\epsilon = 1$ which is the minimal error to tolerate concerning counting nodes in a system and the criterion $\varepsilon_{i,t} < \epsilon$, and the parameter Υ is set to $\Upsilon = 3$ for all system sizes, limiting its purpose to the prevention of early convergence detection.

Mainly, the issue is the determination of universal settings for the control parameters ϵ and Υ such that they can smoothly be applied to different aggregation problems without requiring any global information. A generic solution for the determination of correct parameters settings in dynamic systems could be establishing a dedicated aggregation protocol or process inline with the main aggregation task to set the parameters according to a system situation. Although this is achievable, we chose to adopt a simpler technique in the research project, which is the usage of CV and SE in the detection methods.

To demonstrate the performance of the new detection methods, particularly, the method with CV and the one with SE, several experiments are carried out using the size estimation protocol SSEP which is described in Section 4.3. Methods are applied to the protocol and examined with various settings for the tolerance threshold ϵ, while the parameter Υ is set to a constant. Figure 3.1 illustrates the local convergence detection in opposite to the true convergence, whereas the latter is detected using a dedicated oracle observer using the general error formula 3.3. The figure shows the percentage of nodes which have locally detected convergence using the new methods and the percentage of nodes which actually achieved local convergence. Figures 3.1.a and 3.1.b show the impact of the parameter ϵ on the method with the CV formula. Whilst Figures 3.1.e and 3.1.f show that setting the parameter ϵ to the minimum value that can be tolerated made the method with the SE to achieve the correct detection. It apparent in the Figures 3.1.a and 3.1.e that by tolerating more error, i.e., accepting less accuracy may lead to detecting local convergence before nodes actually converge.

Figures 3.1.c, 3.1.g, 3.1.d, 3.1.h present statistical summery for experimenting the protocol

SSEP with each detection method for 30 runs. The figures show minimum, average and maximum percentage of nodes that have converged. Generally, there are three findings extracted from the figures: (1) each figure illustrates the range percentage of nodes that converge at a particular time. (2) they also show the minimum and maximum time needed to achieve a specific percentage. (3) The figures clearly validates the thresholds settings for the correct detection of convergence, as in all figures no overlapping is present between true convergence curves and local convergence detection. The detection of local convergence always follows the true convergence with a small-time difference.

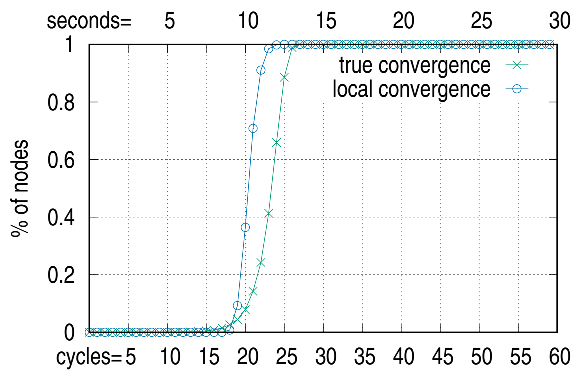
In summary, Poonpakdee's method for local detection of convergence is altered to compute the CV and SE because these statistical formulas do not require any prior information about the system, and provide more accurate convergence detection. The method with CV requires a ratio-based accuracy threshold such as setting ϵ to a common percentage. The method with SE can be used for problems that require absolute accuracy threshold, and the threshold ϵ can universally be set to the minimum possible value. Moreover, the utilisation of the history queue and the settings for CV and SE in the detection methods have bounded the purpose of parameter Υ to the prevention of precocious detection of convergence during the aggregation process, and it becomes independent of system size. The parameter Υ can be set to a constant for all problems, e.g., $\Upsilon = 3$. In general, the adopted methods and settings for local convergence detection require no information about the system, and can be used as default settings for many epidemic systems.

3.4 Global Convergence Detection

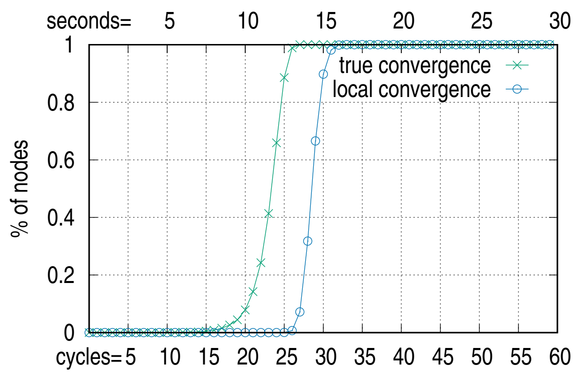
In epidemic systems, protocols can locally detect convergence by monitoring the residual error in local estimates as explained in the former section. However, achieving a globally consistent states requires the explicit detection of system convergence, namely the *Global Convergence*, that describes the moment in which all system nodes have detected local convergence. In real-world systems, nodes achieve local convergence at different times and each node has no awareness about other nodes' state of convergence [51]. Therefore, although a node may detect local convergence, it requires an additional method to detect the convergence of other nodes which may achieve convergence as well, or make divergence or adversely fail. Achieving global convergence in a system makes a consistent state about that system, which is considered a very useful property for many decentralised services [25, 94, 96], especially, for services which require global agreement among nodes to perform an action or to accomplish a task [52, 92]. This section inspects conventional methods for the detection of global convergence in distributed systems and introduces the bases for which the novel detection methods is proposed in this project.

Definition 3.4.1. *Global Convergence* is a term that describes the moment on which all system nodes have achieved and detected local convergence, and implies convergence of the system.

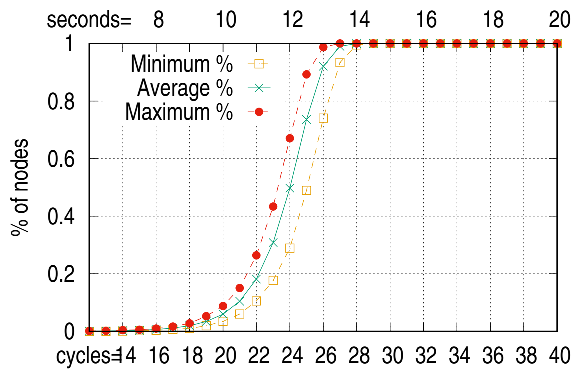
Local convergence detection using CV



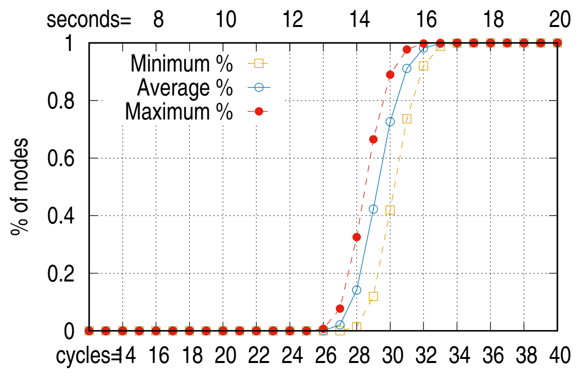
(a) $\epsilon = 0.01, \Upsilon = 3$



(b) $\epsilon = 0.0001, \Upsilon = 3$

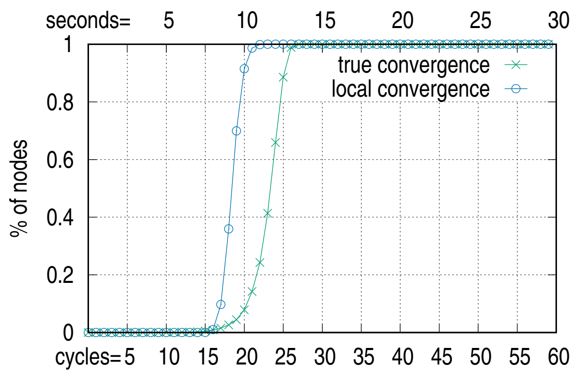


(c) True convergence, $\epsilon = 0.0001, \Upsilon = 3$

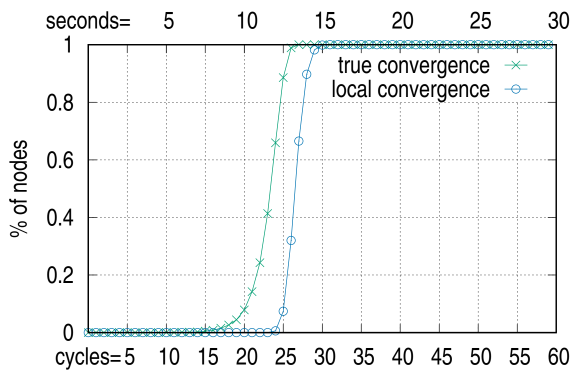


(d) Local convergence, $\epsilon = 0.0001, \Upsilon = 3$

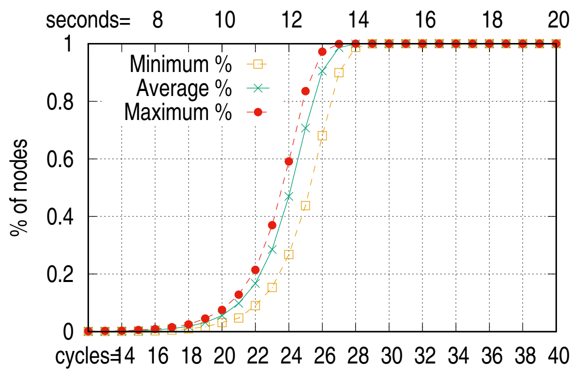
Local convergence detection using SE



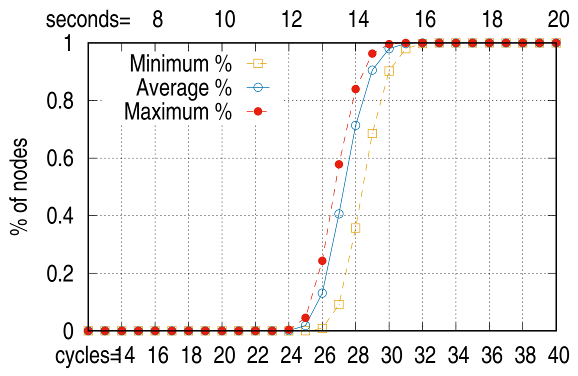
(e) $\epsilon = 100, \Upsilon = 3$



(f) $\epsilon = 1, \Upsilon = 3$



(g) True convergence, $\epsilon = 1, \Upsilon = 3$



(h) Local convergence, $\epsilon = 1, \Upsilon = 3$

Figure 3.1: Performance of different methods for local convergence detection in SSEP, $\mathcal{V} = N = 10^4$, $k = 30$, $l^{\Omega} = 10$, $\ddot{\mathfrak{T}} = 500ms$.

In distributed systems, methods for global convergence detection are usually centralised [42, 45]. Typically, a predefined coordinator takes the mission of collecting information about local convergence from system nodes and the broadcasting of information about global convergence detection. This technique is usually used in general protocols for achieving coordination and commitment. A typical example is the Three-Phase Commit protocol (3PC), which applies the transition into phases mainly to acquire certainty among nodes before the implementation of commitment. In the 3PC protocol, a coordinator starts a commitment process by sending a request to all nodes, and this phase ends upon the collection of all nodes' acknowledgements. In the second phase, the protocol processes information collected in the previous phase and sends the result back to the system, nodes receive results and respond (*Local Convergence*), and nodes responses declare their explicit acceptance or agreement. The second phase provides the coordinator with certainty on nodes readiness for the commitment (*Global Convergence*). In the third phase, the coordinator instructs system nodes to commit. However, epidemic systems require decentralised protocols that can only rely on the local estimates for detecting global convergence [52, 92].

Traditionally, researchers dealt with local convergence in epidemic protocols as a promising outcome that eventually occurs in a finite time. This principle has been applied to synchronous and asynchronous protocols [8, 39, 93, 96, 97]. Consequently, studies attention was analysing local convergence time and speed to define an upper bound of time for attaining a convergence in high probability regardless of system initialisation or scale. The upper time bound is then used to determine a designated action or event (e.g. termination) [25, 96]. In theory, this reduces the need to detect global convergence because system nodes will eventually converge in finite time. However, this principle is based on assumptions that are impractical and inapplicable for many applications in real-world dynamical systems, for example, assuming synchronous or static systems.

Decentralised methods for achieving and detecting global convergence in distributed systems are also proposed [51, 98, 99]. The work of Bahi et al. [32, 51] is one of the most relevant studies. Bahi's study presented analytical proof and practical consideration for an efficient method to achieve and detect global convergence in the algorithm of asynchronous iterations. The detection method does not require any additional information on the system but only the local states of nodes, while always ensuring correct detections. In Bahi's work, the general method of local detection of convergence described in Section 3.3 is used as a prerequisite for detecting global convergence. Bahi's method for the detection of global convergence consists of running a leader-election process in parallel with the main iterative process. After a leader node identifies itself, the leader node checks to detect local convergence, and upon the correct detection, it broadcasts a verification message into the system. Non-leader nodes detects local convergence too and acknowledge the leader node. The leader node then gathers nodes acknowledgements and sends verdict message into the system to confirm or deny the detection of global convergence.

The algorithm presented in Bahi et al. work applies additional steps (leader-election) after detecting local convergence for the detection of global convergence which implies that global convergence is not directly linked to local convergence and the two cannot be assumed at the same time. Also, it collects nodes responses to build a local awareness on the convergence of other nodes at the leader node. It then diffuses the awareness information to those nodes for their self-awareness. In the result, the algorithm formulates a global system awareness about the local convergence of all nodes which principally is the global convergence. However, the reliance on a leader or coordinator in decentralised algorithms is subject to the single point failure problem.

Another relevant study to achieve global convergence in epidemic systems is presented in the work of Poonpakdee et al. [94]. The study presents a novel epidemic algorithm to achieve and detect global convergence from only local state of nodes. In the algorithm, each node i runs multiple independent instances of a desired epidemic aggregation protocol. Let assume that m is the number of protocol instances in each node. Each protocol instance is initialised with the same local value $v_{i,j,t0} = x_i$ where $0 < j \leq m$ and $0 < i \leq N$. As time evolves, each protocol j performs an independent random peer selection and pairwise message exchanging. Eventually, each protocol j in a node i converge to the target value at time tc, and therefore, a sample mean $\bar{e}_{i,tc}$ can be computed form the local m protocols as follows $\bar{e}_{i,tc} = \frac{1}{m}\sum_{j=1}^{m} e_{i,j,tc}$ where $e_{i,j,tc}$ is the local estimate in each protocol j. Also, a standard deviation $s_{i,tc}$ can be calculated by $s_{i,tc} = \sqrt{\frac{1}{m-1}\sum_{j=1}^{m}(e_{i,j,tc} - \bar{e}_{i,tc})^2}$.

In Poonpakdee's method for global convergence detection, the criterion $s_{i,tc} < \epsilon$ is defined as a trigger for the local convergence detection. The global convergence detection is anticipated heuristically by determining the lower bound of time tc at which the criterion $s_{i,t} < \epsilon$, $\forall t \geq tc$ is true. For the sake of the typical tc value, Poonpakdee's study carried out many practical experimentations for various settings of the algorithm and with several system sizes.

Although Poonpakdee's method can detect global convergence without dedicated communications to acquire global awareness in contrast to Bahi's method, the utilisation of the typical tc comes with additional communication costs, which is $m - 1$ times the cost of a single aggregation protocol. Also, the method is entirely heuristic and does not provide explicit awareness of convergence among nodes. From another perspective, the implementation of multiple protocol instances has revealed a possible research direction for global convergence detection in dynamic systems. Further explanation is given in Section 6.2.

Theorem 3.4.1. *There is time tc such that for all $t \geq tc$, all nodes verify their local estimation error such that $\varepsilon_{i,t} \leq \epsilon$ for all cycles greater than Υ.*

The theorem implies that at time tc all system nodes has achieved and detected the local convergence. However, the theorem assumes an oracle that can observe local states of all system nodes to define the correct value of tc. Also, the theorem lacks the certainty on the convergence of the system. There is no awareness available at any node about the convergence of other nodes in the system. The empirical and theoretical proofs of the theorem are presented in the studies [51, 94].

Generally, the detection of global convergence in information dissemination is equivalent to the detection of global convergence in data aggregation. In both tasks, the detection aims to acquire awareness about the system convergence on specific information or on a target value. Interestingly, the awareness strategy applied in Bahi's algorithm [32, 51] is also applied in the centralised protocols for commitment [42, 45]. It is thus clear the significance of building awareness among participants in distributed systems to achieve the desired level of certainty and consistency. The formation of global awareness is attained through a decision-making or a poll-alike process, where every node places a vote after the occurrence of a local event (i.e., convergence). Upon the completion of the poll, results are disseminated to participating nodes which then have local certainty on the occurrence of the event in other nodes, and therefore the global awareness has been consisted.

The formation of global awareness of a convergence detection in epidemic systems is challenging. Protocols cannot rely on a single coordinator to perform awareness acquiring process although a coordinator is determined using a decentralised method (e.g. leader election). On the other hand, the lack of global awareness in epidemic protocols limits their suitability for demanding services such as achieving consensus and making consistency. Consequently, the acquiring of global awareness in epidemic systems may require additional stages of decentralised computations at which a local detection of convergence implies the detection of global convergence. This principle of global convergence detection comes with extra cost of communications.

In epidemic systems, the detection of global convergence is critically required to perform system-wide actions or accomplish global tasks and to put a system in a consistent state. Technically, the detection of global convergence in each node requires two mandatory stages, (1) the local detection of convergence to the desired state, which can be achieved using one of the detection methods described in this chapter. (2) The local detection of convergence to the desired state in other nodes, which can be achieved using explicit detection methods. Chapter 4 introduces two innovative epidemic protocols to achieve convergence and agreement using explicit detection methods without the reliance on any form of centralised coordination.

3.5 Convergence Speed and Detection Time

Convergence speed (a.k.a *Convergence Rate*) is a crucial performance factor that characterises how fast a particular model can attain convergence. Achieving convergence is essential for decentralised applications, and it is in the core of many epidemic services as described earlier. Also, convergence properties such as speed and duration are vital indicators for efficiency and applicability of epidemic services. This section addresses general convergence properties of epidemic models within the context of the system model of this project. Mainly, it highlights the speed of convergence in the symmetric push-sum model which has been described earlier in Section 3.2.

Models that implements asynchronous Push-pull model such as SPSP can achieve convergence faster than those using other schemes such as Push-Sum or atomic Push-

Pull [30, 59]. The non-atomic pairwise data exchanging and the uniform gossiping result in exponential diffusion speed, in which the convergence can be achieved in a logarithmic number of cycles with a time complexity $O(\log N)$. In general, convergence rate of dissemination and aggregation tasks usually corresponds to the characteristics of the diffusion process. Therefore, the convergence rate in epidemic tasks is typically equivalent to the exponential diffusion rate.

The convergence rate in data aggregation tasks depends on the variance reduction rate towards the synopsis target [27]. An aggregation process usually starts a diffusion process for each value in the set of initially distributed values. During the diffusion of values, the distributed computation reduces the variance among initial values by a uniform factor [13]. Eventually, the variance among local estimates reaches very high precision and composes the convergence. Also, the reduction of variance is a direct consequence of the redistribution of initial system mass, which is in the peak case, a distribution of a single value over a whole system. Therefore, the minimum number of cycles required to achieve convergence is generally logarithmic concerning the system size. However, the convergence speed of data aggregation tasks may vary due to system initialisation and the desired synopsis.

In practice, the convergence speed is defined by the cycle length $\dot{\mathfrak{T}}$ [13, 29, 72]. The parameter $\dot{\mathfrak{T}}$ is pivotal in determining convergence duration and convergence detection moment in unit time. The choice of $\dot{\mathfrak{T}}$ value can be an application-specific; however, it is essential to the epidemic task to select the adequate cycle length considering the latency of the underlying network. Choosing a short $\dot{\mathfrak{T}}$ may cause a faster convergence due to the increase in communication rate per unit time [13]. Also, a $\dot{\mathfrak{T}}$ that is too short will flood the underlying network with messages, and the convergence speed will be susceptible to communication delays e.g. message queuing and congestion. The increase in transmitted messages will make them more vulnerable to network faults and may cause severe damage to epidemic tasks due to losing some system mass in the network [29, 72]. On the other hand, a large $\dot{\mathfrak{T}}$ value makes reaching convergence slower and may expose epidemic tasks unnecessarily longer to system dynamics. In summary, the sufficient length of the parameter $\dot{\mathfrak{T}}$ should allow a system to converge reasonably fast and with very high probability.

There are two views for the determination of the typical cycle length. The first view claims that cycle length should be ideally short to achieve faster convergence [72]. The other view adopts a cycle length that is just long enough for most messages to deliver before the next cycle starts [13, 29]. The delivery of all exchange messages within the same cycle allows each node to complete the exchange transaction and achieve definitive local estimates. From the perspective of some applications, it is more sensible to use short cycle length as epidemic tasks will eventually converge, and take advantage of the fast convergence in reliable and stable networks. Other services may prefer using longer cycle length to support epidemic tasks stability and robustness.

Let $\Delta_{i,t}$ be the aggregate length of all delays that node i may experience to complete a pairwise exchange transaction. Also, let Δ_t be the maximum delay among all system nodes $\Delta_t = max(\{\Delta_{i,t}\})$, where $0 < i \leq N$. An epidemic protocol will require cycles to be greater of Δ_t to conserve the system mass in every cycle assuming a stable system and reliable underlying network, so $\dot{\mathfrak{T}} \geq \Delta_t$. It is apparent that finding the correct length of Δ_t requires global information about all system nodes [29]. In real-world systems, assuming a global synchronisation among cycles is not trivial to achieve due to the inherent problems in distributed systems such as clock drifting and coordination, e.g. not all nodes join and start an epidemic task at the same time. Thus, synchronising cycles boundaries among nodes is impractical in actual systems, and using long cycles will not prevent some messages from crossing boundaries among cycles [29, 72].

The symmetric push-sum communication model makes two data messages on average in every cycle [30, 52, 92], the Push message is sent at each cycle, and the Pull message is sent in response to the Push message. In principle, each node sends a Push message completes its exchange transaction when it receives the Pull message as a response. Due to the non-atomicity and asynchrony of the model, choices for the parameter $\dot{\mathfrak{T}}$ are flexible and can have any value that is compatible with the propagation delay of the system network. Therefore, the length of $\dot{\mathfrak{T}}$ in the system model is defined as twice as typical propagation delay, which corresponds to the time needed for a pairwise exchange transaction to complete. In essence, let δ denotes a relatively large propagation delay, so

$$\dot{\mathfrak{T}} = RTT = 2 \times \delta \tag{3.9}$$

Although the previous definition of the cycle length does not guarantee the delivery of all messages within $\dot{\mathfrak{T}}$ in real-world systems, it is still recommended due to the following; it is adequate for most system nodes to complete their pairwise exchange transactions and compose reliable estimates in each cycle; also, it is compatible with the system network, and it sensibly achieves rapid convergence. Therefore, for a realistic system in which a portion of messages may deliver late or not deliver, setting the parameter $\dot{\mathfrak{T}}$ to the round-trip time and utilising a reliable transport layer are the most practical choices for epidemic services.

From another perspective, implementing epidemic models in real-world systems requires expressive specification of the RTT in the system network, not only in relative time units such as cycles but also in absolute units, e.g. seconds. Using classical units in specifying the parameter $\dot{\mathfrak{T}}$ makes the analysis of performance more comparative and convenient. For instance, in dynamic systems, churn rates are usually specified in actual time measurements; to make a valid matching and determine the exact range of churn during a typical epidemic task, it is essential to specify convergence detection times in actual units too. Moreover, describing performance in absolute time units provides a valuable scale for system configuration, and it is an additional attribute for analytical comparisons among research findings.

Internet is the presumed underlying network of the systems in this project. The typical round-trip time of the Internet is the ideal cycle length, and the correct value for the parameter $\dot{\mathfrak{T}}$. Table 3.1 provides RTT measurements of the TCP traffic over different physical networks. The measurements are summaries of research studies from the literature, and the RTT values are carefully collected inclusive of connection setup to fit pairwise communications in epidemic systems. In the table, short RTT values are the median of the shortest measurements, and long values are the median among the longest. The average is the arithmetic mean of all measures of a particular network. As expected, RTT of local-area networks is considerably shorter than wide-area networks. Also, the average RTT in wired networks is shorter than wireless networks, and this applies to both local and wide networks. In general, the RTT measurements of wide-area networks is adequate for epidemic systems, unless an epidemic system is specially made for serving local-area networks. Noticeably, the long RTT in wired networks and the average RTT in cellular networks are approximately equivalent.

Physical network	RTT in *milliseconds*		
	Short	**Long**	**Average**
ADSL & Broadband [100, 101, 102, 103, 104]	48	258	172
Cellular [101, 105]	57	1135	289
LAN [101, 106]	1.3	5.1	3.8
IoT, WSN, & WLAN [107, 108] (One-hop delay)	1.7	310	14.2

Table 3.1: RTT of TCP traffic over various networks.

In the context of previous measures of RTT, it is feasible to define a default setting for the protocols in this project and epidemic systems in general. Although the setting is not precise and may vary in practice, it is useful in specifying convergence speed and detection time in absolute units. Table 3.2 presents the adopted values for the round-trip time and the associated propagation delay.

Parameter	Values in *milliseconds*	
	Min	**Max**
RTT	50	250
δ	25	125

Table 3.2: The adopted values for RTT and the propagation delay δ.

The project is simulation based, hence, communications are simulated through the transport layer component as described in Section 1.6. The settings of simulations have been adjusted to the values in Table 3.2. Mainly, propagation delays in the simulation network are randomly generated using a Poisson process that follows the Gaussian distribution. In particular, the transport layer implements a Weibull distribution of three parameters: a scale η, a shape β, and a location γ. Values from Table 3.2 are used in settings of the Weibull parameters as shown in the next Table 3.3.

Experimental results have demonstrated that propagation delays followed a normal distribution around the desired mean and within the intended spread. Figure 3.2 shows

Parameter	Value	Description
γ	$25ms$	minimum delay, $\gamma = \delta_{min}$
η	$50ms$	$\eta = \frac{\delta_{max} - \delta_{min}}{2}$, and the average delay $\delta_{avg} = \delta_{min} + \eta$
β	4	bounds the delay δ_{max} to $125ms$, $\delta_{max} = \delta_{min} + 2 \times \eta$

Table 3.3: Values of the parameters η, β, and γ for simulating propagation delays.

the results from a dedicated observation component, which is developed for PeerSim to capture and analyse traffic in the transport layer. The results summerise the outcome of 30 experiments, each experiment is executed for 60 cycles using different random seeds. Figure 3.2.a illustrates the density of messages against the propagation delays. In Figure 3.2.b presents the Cumulative Distribution Function (CDF) of the generated delays.

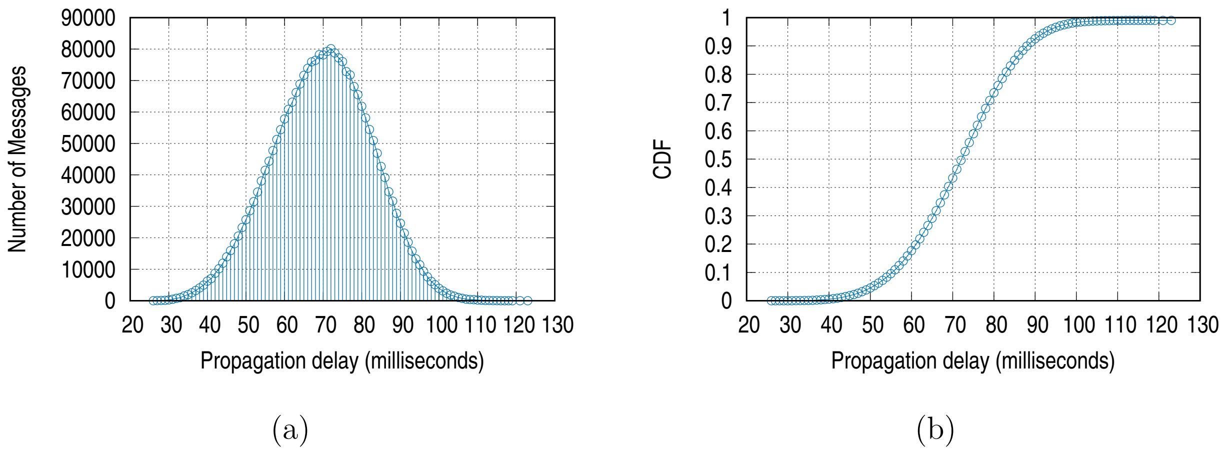


Figure 3.2: Distribution of propagation delays in simulations, $N = 10^4$, $k = 30$, $\delta = [25, 125]$.

Another important implication of expressing RTT in absolute time units is the determination of actual cycle length. There are two cycle length definitions in this project, the cycle length $\dot{\mathfrak{T}}$, which is the default and the recommended for epidemic systems. The parameter $\dot{\mathfrak{T}}$ is previously defined in Equation 3.9. The second cycle length is denoted ($\ddot{\mathfrak{T}}$), and the setting of the parameter assumes that it is long enough for all pairwise exchange transactions to complete within the cycle boundaries. The parameter $\ddot{\mathfrak{T}}$ is primarily used for experimental and simulation purposes. In simulations, some experiments are targeting the mass conservation invariant, and therefore, the portion of mass transmitted through the underlying network in every cycle needs to be stabilised to make a correct assessment of protocols performance. In those experiments, the cycle length is set to the parameter $\ddot{\mathfrak{T}}$, and the experimental results are described in term of the adopted cycle length. The parameter $\ddot{\mathfrak{T}}$ is defined as follows,

$$\ddot{\mathfrak{T}} = RTT + t_{off} \tag{3.10}$$

The parameter t_{off} is a synchronisation offset, and it is defined as the maximum time offset between any pair of nodes in the system [30]. The parameter t_{off} is applied to simulate non-simultaneous nodes starting and execution times in the real-world systems. It also includes time skew and drifts in the local clocks of nodes. Assume asynchronous system where all pairwise exchange transactions complete within the RTT. For any two nodes, i and j, assume that each node has a different cycle sequence and both nodes have started a new cycle at t_i and t_j respectively, where $t_i \leq t_j$. The offset between the starting times of the two arbitrary cycles is less or equal the typical cycle length, $t_j - t_i \leq RTT$, otherwise, t_j will be large to coincide the next cycle in node i, $t_j \geq RTT + t_i$. In consequence, adding the artificial interval t_{off} to the cycle length ensures that pairwise exchanging transactions in two different cycles do not overlap and allows system mass to stabilise before the start of the next cycle. The following table describes the adopted cycle length parameters in this project,

Parameter	**Definition**	**Description**
$\dot{\mathfrak{T}}$	$\dot{\mathfrak{T}} = RTT = 2 \times \delta_{max} = 250ms$	Default cycle length, cycles are fully-asynchronous and messages may deliver in different cycles.
t_{off}	$t_{off} = 250ms$	Synchronisation offset
$\ddot{\mathfrak{T}}$	$\ddot{\mathfrak{T}} = RTT + t_{off} = 500ms$	Experimental cycle length, used to synchronise cycles and ensure messages delivery within the cycle.

Table 3.4: Parameters of the cycle length.

It is presently possible to describe epidemic tasks performance in expressive way that is compatible with the real-world systems. Performance attributes such as convergence speed and duration, detection moment, and task termination can be specified in relative and absolute time units, which provide useful information about the behaviour of epidemic systems. In simulations, experiments description will state the adopted cycle length, and experimental results will be illustrated using cycles and actual time units. Figure 3.3 presents a sample of the experimental results of this project. The figure shows the results of simulating the SSEP protocol from Section 4.3. In the left column of the figure, the simulation of the protocol has adopted cycle length to the typical value of $\dot{\mathfrak{T}}$ as described in Table 3.4. Also, the protocol is simulated using the cycle length $\ddot{\mathfrak{T}}$ and the results are shown in the right column.

In Figures 3.3.a and 3.3.b, the aggregates of initial system mass, $\mathcal{M}v$ and $\mathcal{M}w$ are correct at cycle 0; however, the drop in system mass in the subsequent cycles is due to the portion of mass in the exchanged messages that are not yet delivered. The figures also show that under stable conditions, the portion of mass that leaves the system due to pairwise communications is the same as the portion that returns to the system. In opposite, results in Figures 3.3.e and 3.3.f show the correct aggregate of mass at all cycles because all exchanged messages are delivered within the boundaries of every cycle. Figures 3.3.c,d,g and h, illustrate the convergence of the epidemic task to the correct size in the same number of cycles. The figures also show the corresponding actual time based on the adopted cycle length.

An interesting finding which can be observed in Figures 3.3.c and Figures 3.3.g is that some nodes have achieved convergence to the target value at the cycle 10, and the number of converged nodes increases in the subsequent cycles. Also, Figures 3.3.d and Figures 3.3.h confirms that local detection of convergence under the desired accuracy is actually performed between the cycles 15 and 30.

Further experiments have targeted the parameter β of the delay distribution. Varying of the parameter β makes the distribution function to generate some very long delays, which exceed the defined upper bound in the system, i.e. *unbounded delays*. The system model adopts a reliable network, and hence, all messages are eventually delivered. The results showed no major impact on the performance of the SSEP neither on the convergence or on the detection method. The only nearly noticeable observation was a small delay in achieving the convergence. Such delay is expected as some messages will arrive in late cycles. However, these experiments approved that adopted the convergence detection method, and the proposed protocols are operating in a fully asynchronous environment. The results are removed as they make no significant contribution.

In conclusion, describing epidemic task performance and behaviour in different time units such as cycles and seconds is an effective way of comparing research findings and helpful guide for implementation in real-world systems. Mainly, convergence properties such as speed and duration are essential indicators for the applicability of epidemic services. This section has explained a practical procedure to specify convergence properties using relative and actual time units. Also, the section has demonstrated analytically and through simulations, the relation among pairwise exchange, cycle length and task duration.

The analysis of an epidemic task performance is mainly based on the convergence speed and detection in the task. Convergence speed is subject to the diffusion rate and variance reduction of data dissemination and aggregation processes. The detection of convergence is achieved using methods that need application-specific parameters. Detection methods and parameters determine convergence duration and detection moment. The analysis of convergence of different epidemic tasks under various system conditions is investigated in the following chapters.

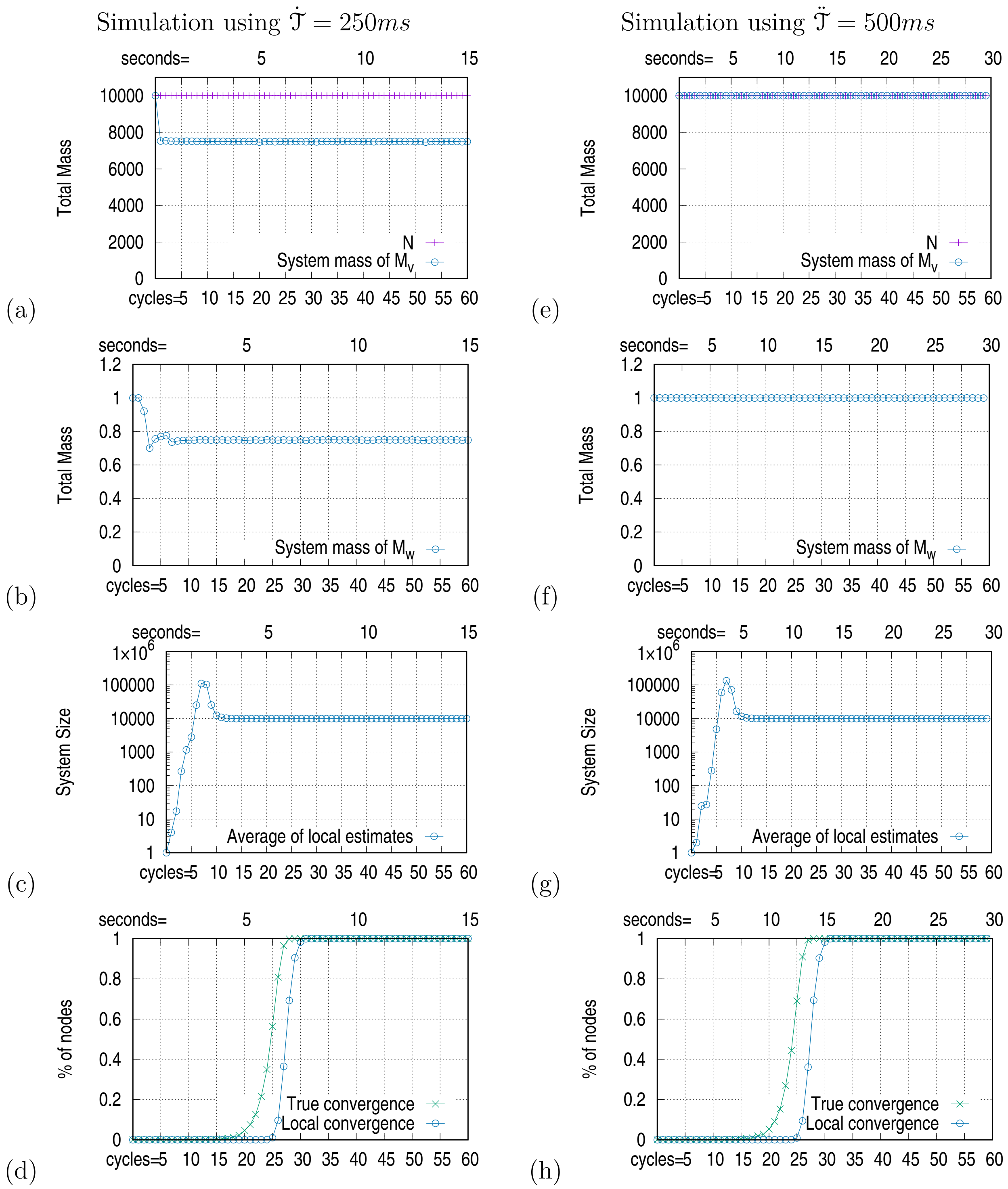


Figure 3.3: The impact of cycle length on protocols' performance, $N = 10^4$, $k = 30$, $\delta = [25, 125]$. Convergence is detected using the SE method in 3.8, $l^{\Omega} = 10$, $\epsilon = 1$, $\Upsilon = 3$.

Chapter 4

Agreement in Epidemic Systems

Decentralised coordination is the way to achieve consistency in epidemic systems. Typically, coordination is an agreement process to acquire consensus among system nodes on a particular target or matter. Although there are many algorithms for distributed consensus in the literature, they are not adequate for epidemic systems. Achieving agreement in epidemic systems is challenging due to decentralisation, asynchrony, and complete reliance on local states of nodes. The formation of the global agreement requires the detection of global convergence using a decentralised decision-making process. The detection of global convergence is described in Section 3.4.

This chapter introduces the first innovative contribution of the research work in this project. It proposes the Phase Transition Algorithm (PTA). The algorithm achieves global convergence and agreement using a decision-making method that is explicit and decentralised. The PTA is a generalised model of transition into subsequent phases, which aims to acquire global awareness in epidemic systems that are asynchronous and stable. It extends the style of coordination in centralised approaches, e.g., commitment protocols, and the model of epidemic consensus in synchronised systems as presented in the work [68]. The algorithm is entirely decentralised and does not require any coordinator or leadership. Also, it inherits all the intrinsic properties of epidemic models such as scalability, resiliency, and fault-tolerance. Two novel protocols that implements PTA are also proposed and presented in this chapter. The Phase Transition Protocol (PTP) and Epidemic Consensus Protocol (ECP) are developed for making distributed consistent states in epidemic information dissemination and epidemic data aggregation respectively.

In the chapter, models of agreement in centralised and distributed systems are discussed in the next section emphasising the need for acquiring global awareness among system nodes. Section 4.2 describes the PTA and its general formation. The System Size Estimation Protocol (SSEP) is presented in section 4.3. Agreement protocols, PTP and ECP are introduced in sections 4.4 and 4.5 respectively. Further discussions on the PTA is presented in Section 4.6. The work presented in this chapter has been published in the papers [52, 92].

4.1 Agreement in Centralised Distributed Systems

The coordination among participants in distributed systems is typically targeting the consensus [42, 43]. The consensus is the eventual result of an agreement process, which has different formations. In the general formation, each participant proposes or starts with an initial value and exchanges it with other participants. All participants follow the exact process to decide on a common output or value [44]. Principally, all participants should achieve the same outcome regardless of the problem they are trying to agree on [9]. The distributed formation of the agreement process is usually based on distributed data aggregation algorithms, for instance, the computation of averaging [46, 48, 49]. Also, decentralised data aggregation algorithms are the conventional method for acquiring an agreement in gossip-based applications. This section explains the agreement process based on distributed data aggregation and presents examples of the agreement protocols for the structured and centralised systems.

Distributed data aggregation protocols generally provide global information about a system by computing a synopsis function, e.g. {*average*, *sum*, *sample*, etc.} over a distributed set of data values [26, 58]. In a system of N nodes where each node i has a set of neighbours and holds a numeric value $v_i = x_i$ that describes a property in node i or in its environment. The aggregation protocol at node i exchanges v_i with neighbours to compute a particular global function. Upon receiving a message from node j, the node i updates its local value $v_i = f(v_i, v_j)$. After a sufficient number of data exchanges, all nodes in a system converge to the same output $\mathcal{V}$, which is the aggregation target.

The optimum performance of distributed data aggregation is obtained when the computation of the synopsis function is distributed over a structured topology such as *trees* [19, 20]. There are two communication models to perform data aggregation over trees, Broadcast-Convergecast and Convergecast only. In the Broadcast-Convergecast, a root node disseminates a request for the aggregation over system tree and collects responses that carry data values from non-root nodes. In the Convergecast model, the data aggregation starts from the deepest nodes and results are sent towards the root node. A typical example of the Convergecast model is the Collection Tree Protocol (CTP) [19, 109] that provides a faster gathering of results at the root node. Each non-root node i in CTP computes $\grave{v}_i = f(v_i, \grave{v}_1, \ldots, \grave{v}_{m_i})$ where m_i is the number of child nodes at the node i and $\grave{v}_1, \ldots, \grave{v}_{m_i}$ are the aggregate results at child nodes. Thereafter, node i sends $\grave{v}_i$ to its parent node. After a sufficient number of steps, the root node receives $\grave{v}_1, \ldots, \grave{v}_{m_i}$ from child nodes and calculates the final target value $\grave{v}_{root} \approx \mathcal{V}$.

In CTP and similar schemes, the aggregate result $\grave{v}_{root}$ is eventually known to the root node only. In the consensus problem, $\grave{v}_{root}$ is the final result, and it is in the interest of all participants and hence, the root node needs to disseminate the final result back to the tree. After disseminating the final result, the agreement is ostensibly achieved, principally, because the final result becomes known to all nodes [43, 110]. However, in unreliable systems, the root node is no more certain about the reception of the final result at all non-root nodes that have sent their local aggregate results formerly. A failed node in the tree topology may prevent a whole branch from receiving the final result. For instance, in the tree Two-Phase Commitment (2PC) protocol [42], the root node collects acknowledgements from all nodes to acquire certainty about the reception of $\grave{v}_{root}$. Afterword, and as a confirmation of the agreement, the root node informs all other nodes about the global reception of the final result, and nodes can then accomplish a global task or take a system-wide decision.

Commitment protocols such as 3PC protocol can also achieve consensus in dynamic systems [42, 43, 99]. The agreement process in the 3PC protocol make transition in three phases to achieve the consensus on the output of an aggregation function. The protocol begins in (Broadcast phase) in which a coordinator node broadcasts a 'request' message into the system and collects acknowledgements from nodes. Each node i receives the request message sends $\grave{v}_i$ in the acknowledgement to the sender node which can be the coordinator node, a parent node or a neighbour node based on the adopted topology, where $\grave{v}_i = f(v_i, \grave{v}_1, \ldots, \grave{v}_{m_i})$, and m_i is the number of collected acknowledgements from peers at the node i and $\grave{v}_1, \ldots, \grave{v}_{m_i}$ are the aggregate results at peer nodes. In consequence, the aggregate results of nodes propagate towards the coordinator node. The successive phase is the (Pre-Commit phase) and in the absence of faulty nodes, the coordinator computes the final result $\grave{v}_{root} \approx \mathcal{V}$ and disseminates it back to the system.

Since the final result $\grave{v}_{root}$ is disseminated to all nodes in the Pre-Commit phase, each node become aware of the result and ready to commit. Nodes end the Pre-Commit phase and acknowledge their readiness through 'accept' message. Upon the collection of acceptance messages from all nodes, the coordinator has achieved the certainty on the reception of the final result, and so it broadcasts 'commit' message and starts the (Commit phase). On the reception of 'commit' message, nodes can commit and can make a designated action or decision. Otherwise, the coordinator may send 'abort' message or nodes approach the timeout threshold, and either case, nodes abort the agreement process.

Paxos is a distributed consensus protocol that lacks the centralised coordinator and more complicated than the 3PC protocol. The protocol achieves reliable agreement among participants on some value in the presence of dynamical conditions [86, 111]. In the general formation of Paxos protocol, each protocol instance in a node can have one of the following roles: *proposers* or *acceptors*. Also, the protocol can have multiple consensus tasks running simultaneously for an agreement on the same target. An instance of Paxos protocol proceeds in two phases. During Phase-1, a proposer that wishes to submit a value sends a 'request' message to a quorum of acceptors. Upon receiving a request message, acceptors make a global selection for the latest consensus task using the one with the higher identifier, and they respond to the proposer confirming the currently accepted request. The second phase starts when the proposer receives answers from all acceptors in the quorum.

In Phase-2, the proposer also selects the consensus task with the higher identifier and then sends an 'accept' message to the same quorum of acceptors. The acceptors either acknowledge receiving of the 'accept' message and confirm reach to the consensus, or they respond to the proposer informing a new consensus task. However, further steps are used in the Paxos protocol to avoid scenarios on which proposers compete indefinitely, e.g. a leader node can be elected. For a further explanation on the Paxos protocol, the reader may refer to [86, 111, 112].

One of the interesting findings in the previous explanation is the transition in consecutive phases to attain the required certainty on the global agreement. In the CTP, nodes make awareness of the final result just after disseminating it by the root node; however, they still need additional confirmation step to achieve global awareness. Also, nodes in the 3PC protocol make a local awareness about the final result at the end of the Pre-Commit phase. In the Commit phase and by receiving the 'commit' message, nodes achieve the global awareness on the final result, which implies acquiring of the global agreement and so the consensus. The Paxos protocol exchanges 'request', 'accept', and acknowledgement messages among proposers and acceptors to attain the certainty on the correct consensus task and achieve global awareness on the consensus target.

Moreover, it is notable the need for global information about the system size, the number of non-faulty nodes or the quorum for the agreement formation. Typically, the decision-making requires the collection of the same number of confirmations from participating or quorum nodes to conclude the consensus. For example, the variation in the number of acknowledgements in a coordinator or proposer node between the first phase and the second phase of the agreement process can lead to the abortion of the process. In general, the agreement process in the coordination problem makes the transition into phases to ensure global awareness and achieve the consensus. Typically, the distributed result of the data aggregation is the target of the agreement process. The data aggregation process eventually achieves local and global convergence, which can be utilised to acquire awareness.

In contrast to using convergence in the distributed averaging to reach the consensus and consistency, which is probabilistic, and solely possible in reliable and static systems. Also, in contrary to the fast computation of the final result in the CTP, the 3PC protocol can achieve the consensus in unreliable systems, and with the optimum overhead when used over tree topologies. However, it still susceptible to the single-point of failure problem, for instance, the failure of the coordinator node in the Commit phase. On another hand, static trees are not ideal in the real-world systems due to nodes failure and dynamics [27]. Dynamic trees are introduced to cope with the real conditions but dynamic trees require additional overhead to establish the tree for every change in the system [109]. Generally, the robustness of tree-based schemes relies on the consistency of the underlying network, and dynamic trees require additional effort for the tree construction and maintenance.

Algorithms for the consensus in distributed systems are typically probabilistic, and it is theoretically assured in stable systems; this conclusion is the synopsis of the extensive review of the literature presented in this section and Section 2.3. However, algorithms for the consensus in unreliable distributed systems adopted the transition into phases to attain the global awareness on the agreement. Recently, transition into phases is also used in an epidemic algorithm for the consensus on the failure detection. The algorithm is introduced in

[68], and it makes the transition into phases in the synchronous systems to achieve distributed consensus on the number of the detected failures in the system. The next section describes the Phase Transition Algorithm (PTA) for epidemic systems.

4.2 Phase Transition Algorithm (PTA)

The PTA is an epidemic algorithm to achieve explicitly detected consistent states in epidemic systems. The algorithm extends and generalises the model presented in [68] to asynchronous epidemic systems that are usually stable. PTA consists of several successive phases. Each phase is a separate epidemic data aggregation process that is typically used to achieve global synopsis targets such as average, summation and count [26, 30], and the aggregate results eventually converge to the target value $\mathcal{V}$. The epidemic data aggregation and convergence have been described in Section 3.2.

The PTA has two main processes: a *Task* process that can be a dissemination or an aggregation task; and an *Agreement* process. The task process is the first phase, while the agreement process involves several consecutive phases. The transition among different phases is performed after the detection of convergence in each phase using one of the detection methods described in Section 3.3. The number of consecutive phases determines the desired level of certainty about the targetted system state, and it is an application-specific parameter. Each phase in the agreement process aims to achieve and detect global convergence explicitly. For this purpose, the formation of data aggregation process in each phase is in fact a decentralised collective decision-making process, in which the *count* aggregate function is used. The method aggregates each node that achieves and detects local convergence. The aggregation process of each phase is targeting the initial system size N. The global convergence (i.e., global awareness) is achieved when all system nodes converge and locally detect the correct system size N.

Algorithm 6 illustrates the PTA for the coordination in epidemic systems. In the algorithm, PHASE-1 is dedicated to the main dissemination or aggregation task. In this phase, nodes need to detect local convergence before making the transition to the next phase. The local detection of convergence in PHASE-1 at node i implies its awareness of the target value. To achieve global awareness and agreement, node i proposes itself to the decision-making process in PHASE-2 and makes the transition to the phase. The aggregation process in PHASE-2 increases the aggregate results by one for each node participating in the phase. Eventually and after a sufficient number of cycles, all system nodes will join PHASE-2, the aggregate result at each node will also converge to N. Each node can detect global convergence by verifying the local convergence to N. By the end of PHASE-2, all nodes will detect the global convergence, at which global awareness and agreement on the main task target are achieved. Also, nodes will make the transition to PHASE-3, in which nodes may apply a system-wide action or decision, or may complete PHASE-3 to detect global convergence of the process in PHASE-2. Due to the asynchrony and during PHASE-2, nodes achieve global agreement on

Algorithm 6: Phase Transition Algorithm (PTA)

```
1 At each node $i$, $0 < i \le N$:
    // Task process
      PHASE-1, perform the distributed data dissemination or aggregation step.
        Monitor the local state $e_i$, and upon the detection of convergence, i.e. $e_i \approx \mathcal{V}$,
        propose $i$ to the next phase and make the transition.
    // Agreement process
      PHASE-2, perform the distributed step of the aggregate count. Monitor the local
        state $e_i$ and upon the detection of convergence, i.e. $e_i \approx N$, propose $i$ to the next
        phase and make the transition.
      PHASE-3, perform the distributed step of the aggregate count. Monitor the local
        state $e_i$ and upon the detection of convergence, i.e. $e_i \approx N$, propose $i$ to the next
        phase and make the transition.
      ⁞
    // The desired level of certainty achieved
      PHASE-M, apply a designated action or decision.
```

the target of the process in Phase-1 at different times and have no awareness about the state of nodes in Phase-2. Therefore, allowing all nodes to achieve convergence in Phase-2 is critically required for taking global actions. More explanation about local and global actions is discussed in Section 4.6.

Agreement process in the PTA requires the knowledge of the system size N beforehand to verify the convergence in each phase. For this purpose, the PTA includes a dedicated data aggregation protocol to compute the size of epidemic system. The protocol runs in parallel with the primary task process in Phase-1, and therefore, it can provide an estimate for the size before starting the agreement phases. The following section describes the adopted epidemic protocol for size estimation in the PTA.

4.3 System Size Estimation Protocol (SSEP)

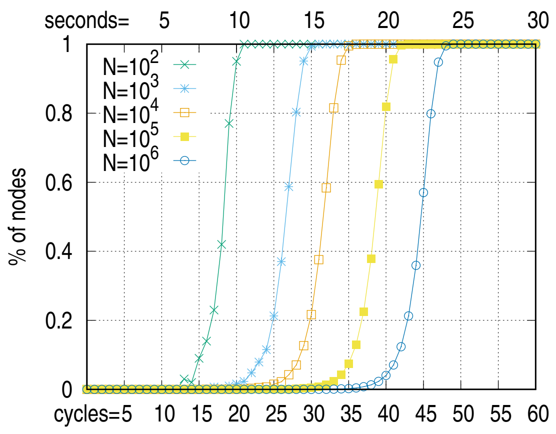
Size estimation is a data aggregation process that targets the number of nodes in the system, and the process can provide a correct aggregate result at each node in stable systems. SSEP implements the SPSP for data aggregation described in Section 2.1.3 and adopts the settings for the global *count* function [26, 30]. SSEP is illustrated in Algorithm 7.

In SSEP, each node i holds a pair of local values v_i, w_i to perform the global aggregation process, where v_i is the value and w_i is the weight. Initially, the protocol sets the pair to $v_i = 1$, $w_i = 0$ at all nodes, except one node $\hat{\imath}$ that has the initial weight set to $w_{\hat{\imath}} = 1$. The initialisation of the weight follows a peak distribution for the global summation function [26, 30]. At each cycle, every node i divides the local pair values into two halves $v_i = \frac{v_i}{2}$, $w_i = \frac{w_i}{2}$ and sends half to a random peer in a Push message. When a node receives a Push message from node j, it divides the local pair values and sends half as a response to node j in a PULL message. Finally, after receiving a PUSH or a PULL message, nodes i and j update their local values $(v_i, w_i) \longleftarrow (v_i + v_j, w_i + w_j)$.

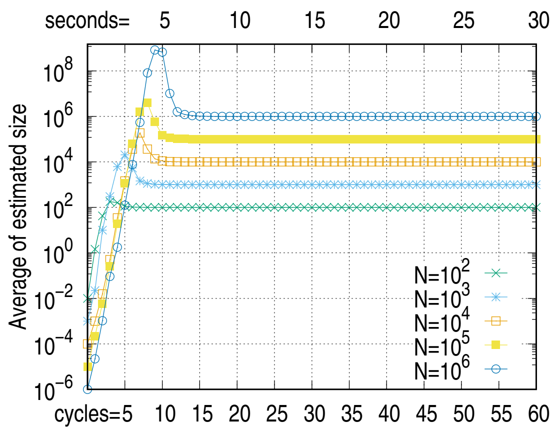
As described in Section 3.2, after a sufficient number of cycles the protocol will converge to the target value $\mathcal{V}$, where $\mathcal{V} = \frac{\mathcal{M}v}{\mathcal{M}w}$ and $\mathcal{M}v$, $\mathcal{M}w$ are the aggregate of initial system mass. Eventually, the local aggregation pair in each node i will converge to $v_i = 1$, $w_i = \frac{1}{N}$, and the size estimation can be computed locally by $\frac{v_i}{w_i}$.

Although interleaving messages are present in the system and as long as the mass invariant holds, the aggregate result $\frac{v_i}{w_i}$ at each node i will quickly converge to N with a very small error [26, 27, 30]. However, since the protocol is continuously executed, it can also adapt to changes in dynamical systems. This project proposes a continuous and adaptive protocol in Section 6.2.

Figure 4.1 shows the experimental results of simulating the protocol SSEP. Results validate the protocol ability to achieve correct size estimations for systems of various sizes. In Figure 4.1.a, results present the percentage of nodes that have achieved local convergence in a system after a sufficient number of cycles. The figure confirms that 100% of nodes will converge to the correct size. It also shows the variation in convergence speed and times concerning the system size. It is apparent that despite the logarithmic increase in sizes, the interval from the first detection of convergence to the achieving of global convergence is approximately the same and ranges between 10 to 12 cycles. However, the first detection of convergence takes longer to achieve as system scale increases. Figure 4.1.b confirms that aggregate results at all nodes eventually converge to the same target. The figure shows a variation among local estimates in the early cycles of the aggregation process. The variation reduces quickly in later cycles to precision at which the variation is hardly noticeable. After convergence, aggregate results and size estimations remain correct as long as the system is stable.



(a)



(b)

Figure 4.1: The convergence in SSEP to true size N. Convergence is detected using the general method in 3.3, $\epsilon = 0.01$, $\Upsilon = 5$. $\ddot{\mathfrak{T}} = 500ms$, $k = 10$.

Algorithm 7: System Size Estimation Protocol (SSEP)

```
Require: a peer-sampling service, e.g. NCP$^+$.
Initialisation: at each node $i$, $v = 1$, $w = 0$, except one node has $w = 1$.

1  At each cycle at node $i$:
2  $j \longleftarrow getRandomPeer()$
3  $v = \frac{v}{2}$,    $w = \frac{w}{2}$                  // divide aggregation pair
4  send $\langle v, w, reply = true \rangle$ to $j$            // a PUSH message to node $j$

5  At event 'receive message $m$ from $j$' at node $i$:
6  if $m.reply$ then                                      // $m$ is a PUSH message
7      $v = \frac{v}{2}$,    $w = \frac{w}{2}$              // divide aggregation pair
8      send $\langle v, w, reply = false \rangle$ to $j$   // a PULL message to node $j$
9  $v = v + m.v$,    $w = w + m.w$                         // update local pair

10 function size()                    // public service to get size estimation
11     $size = \varnothing$
12     if $w_i > 0$ then $size = \frac{v_i}{w_i}$
13     return $size$
```

4.4 Agreement in Epidemic Information Dissemination

Achieving consensus on the information dissemination is essential to the data consistency in epidemic systems. Information can take any form of data that are in the interest of others, for example, news, updates, failure notification or failures count, etc. Typically, a node generates an information item and starts a propagation process into the system for that item [23, 56, 57]. Each node stores each item it receives into a buffer and continuously forwards all information in the buffer to a peer that is randomly selected at each cycle. The convergence in information dissemination is a convergence of content, and the propagation process of a particular information item achieves convergence upon the reception of the item at all system nodes. The efficient diffusion of epidemics guarantees the convergence of the propagation process [23].

The formation of consensus in epidemic information dissemination has to be decentralised. Also, the consensus requires the explicit detection of convergence and agreement, as stated in the PTA. For the sake of the clarity, we adopted an Information Dissemination Application (IDA) to simulate the distributed generation of information items for which propagation and global agreement are required. The protocol PTP is introduced as a part of IDA to manage and achieve consistency on information items. The PTP achieves the consistent state for an item through acquiring global agreement on the dissemination of that particular item. The next section presents the conceptual design and practical scenario of IDA.

4.4.1 Information Dissemination Application (IDA)

In the IDA, each node randomly generates new information items with a given probability. New items are assigned unique identifiers (ID) using the next locally incremental ID with no global or centralised coordination. This way, items with a specific ID can be generated simultaneously at different nodes, and ID duplication must be resolved.

Each information item is associated with one of three possible states: PROPAGATION, AGREEMENT and COMMIT. The state diagram in Figure 4.2 shows the states and the transition criterion as stated by the PTA. The same item can be associated with different states at different nodes. The ultimate goal is for each item to reach the final state (COMMIT) at all nodes, which corresponds to achieving global agreement on that item.

The node at which an information item is generated is called the *originator*. Each new item is represented by a tuple, which includes the item ID, the originator ID and the item state. Also, each node is started with an empty information cache C. The tuple of new item is added to the local cache C with initial state PROPAGATION. Each tuple also contains some numerical variables that are used by the consensus protocol, and they are described later. Each node periodically disseminates the items that are present in its local cache C in the system by sending it to a randomly chosen peer. When a node receives a message with a remote information cache, it updates its local cache by merging the local and remote entries, aggregating identical items and resolving ID duplicates.

Typically, nodes have no prior knowledge of the system size N. Thereby, each node runs the protocol SSEP in parallel with the consensus protocol as stated in the PTA. SSEP runs independently of the consensus protocol, and its communication exchanges are separate and may have different propagation patterns. On the other hand, IDA involves the Phase Transition Protocol (PTP), which is the consensus management protocol that determines and updates states of information items. Also, the application includes the peer-sampling service NCP$^+$ as described in Section 2.5.

Generally, the IDA is a simplified model which may find applications in diverse domains, such as failure detection and consensus, transactions in distributed databases, or consent on replicas, etc. The consensus protocol PTP is explained in the next section.

4.4.2 Phase Transition Protocol (PTP)

PTP is the consensus protocol that manages successive phases of the PTA and determines the states of the information items. The protocol is a cascade of three phases which correspond to the states of information items in IDA, and each phase in PTP is an epidemic data aggregation process that converges to a specific target. Mainly, each phase aggregates the number of nodes in the system which have a particular item in a specific state, respectively, at PROPAGATION and AGREEMENT states. When the aggregate results match the system size N with some error under the tolerance threshold, the state of the local copy of the item is updated to the next state (*phase transition*). The action taken at the transition

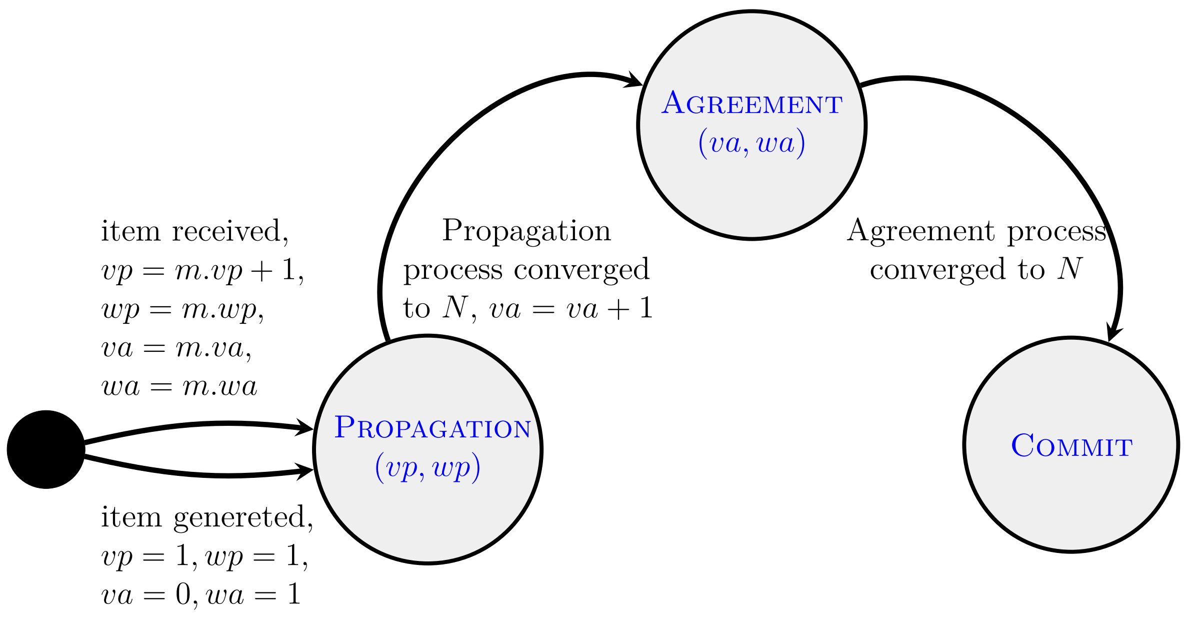


Figure 4.2: IDA state diagram for an information item

to the Commit state is application-specific and it is further discussed in Section 4.6. The illustration of the protocol PTP spans over two parts of Algorithm 8.

PTP maintains a local cache C_i of information items at each node i. The local cache is initially empty and will be used to store items either generated locally or received from other nodes. Each item in C_i is represented by a tuple, as described earlier. The tuple also contains two aggregation pairs. The pair (vp, wp), is used by the process in Propagation phase to estimate the number of nodes which have received the item, i.e. the $(p)ropagation$ count. The second pair (va, wa), is used to estimate the $(a)greement$ count that is the number of nodes holding the item in the Agreement phase.

Algorithm 8 demonstrates the action of PTP at each event. At the event of new information item is generated at node i, the protocol selects the next unique identifier (line 2) and inserts a new tuple into C_i (line 4). Also, at each cycle, each node i divides the aggregation pairs of each tuple in the cache C_i and sends a copy of C_i to a random peer in a Push message. Then, the protocol manages items in C_i by verifying the criterion of the transition from a phase to the next for each item. On the event of receiving a message from a peer, the protocol responds in a Pull message as shown in lines 13-15, and it updates the local items in C_i using the received times in the remote cache.

In the second part of Algorithm 8, two local procedures are defined. The $MangeItmes$ procedure obtains the estimated system size N from SSEP using the service $size()$. Also, it verifies the state of each tuple τ in C_i, the criterion in line 5 decides upon the transition from the Propagation to the Agreement phase. The detection of convergence in the Propagation phase uses the formula 3.3 of the general method described in Section 3.2. The method verifies the error in local estimates to the system size being under the relative tolerance threshold ϵ for a number of consecutive cycles Υ. Cycles threshold Υ ensures a robust transition to the next phase and avoids early false transitions. The transition to the

COMMIT phase is associated with a similar verification on the AGREEMENT phase (line 9).

In IDA, information items of the same ID may generate at different nodes and times. The procedure *UpdateItems* of Algorithm 8 adds new items if not exists in C_i in line 20. Also, it updates the tuples and resolves duplicates by keeping either the oldest tuple or keeping the one with the lowest originator ID if the tuples have the same creation time. Tuples related to the same information item are updated in line 16, and duplicate IDs are resolved in line 18.

4.4.3 Experimental Results for PTP

In simulations, the following settings have been adopted. Each experiment run is associated with different random seeds to validate performance and enforce randomisation. The system default size is $N = 10^4$ nodes, and the maximum experiment length is 100 cycles. NCP$^+$ maintains the overlay topology with $k = 10$. Simulation cycles are of fixed length, and a cycle adopts the structure used in [30, 52]. In general, the cycle length is defined long enough for all nodes to complete the pairwise exchange transaction, which follows the partial synchrony model adopted in this project for the cycle length $\ddot{\mathfrak{T}}$ as described in Section 3.5. In the model,

Algorithm 8: Phase Transition Protocol (PTP)

Require: a size estimation service, e.g. SSEP; a peer-sampling service, e.g. NCP$^+$; tolerance threshold ϵ; cycles threshold Υ;

Initialisation: at each node i: $C \longleftarrow \emptyset$; C is the local cache of items defined as $C = \{\tau = \langle id, o, t, vp, wp, va, wa, state\rangle, \ldots\}$, where 'id' is the item identifier, 'o' is the originator identifier, 't' is the item generation time, (vp, wp) is the propagation pair, (va, wa) is the agreement pair and 'state' is the item state;

```
1  At event 'new item generated' at node $i$:
2  $id \longleftarrow$ next locally unique identifier
3  $t \longleftarrow$ current cycle
4  $C \longleftarrow C \cup \{\langle id, i, t, 1, 1, 0, 1, \text{PROPAGATION}\rangle\}$

5  At each cycle at node $i$:
6  $j \longleftarrow getRandomPeer()$
7  foreach $\tau \in C$ do          // divide aggregation pairs for each local tuple
8      $\tau = \langle \tau.id, \tau.o, \tau.t, \frac{\tau.vp}{2}, \frac{\tau.wp}{2}, \frac{\tau.va}{2}, \frac{\tau.wa}{2}, \tau.state\rangle$
9  send $\langle C, reply = true\rangle$ to $j$                    // a PUSH message to node $j$
10 $MangeItems()$                              // manage the states of local tuples

11 At event 'received $m$ message from $j$' at node $i$:
12 if $m.reply$ then                                       // $m$ is a PUSH message
13     foreach $\tau \in C$ do      // divide aggregation pairs for each local tuple
14         $\tau = \langle \tau.id, \tau.o, \tau.t, \frac{\tau.vp}{2}, \frac{\tau.wp}{2}, \frac{\tau.va}{2}, \frac{\tau.wa}{2}, \tau.state\rangle$
15     send $\langle C, reply = false\rangle$ to $j$               // a PULL message to node $j$
16 $UpdateItems(m.C)$           // resolve duplicate and update local tuples
```

```
17 procedure ManageItems()
18     foreach $\tau \in C$ do                    // manage the states of local tuples
19         switch $\tau.state$ do
20             case Propagation do
21                 if $size() > 0$ and $\left|\frac{size() - \frac{\tau.vp}{\tau.wp}}{size()}\right| \leq \epsilon$ for at least $\Upsilon$ cycles then
22                     $\tau.state$ = Agreement                // make the transition
23                     $\tau.va = \tau.va + 1$
24             case Agreement do
25                 if $size() > 0$ and $\left|\frac{size() - \frac{\tau.va}{\tau.wa}}{size()}\right| \leq \epsilon$ for at least $\Upsilon$ cycles then
26                     $\tau.state$ = Commit                   // make the transition
27             case Commit do
                   /* Take some application-specific action or decision.    */
28 procedure UpdateItems(Cr)
       Input: remote items cache $Cr$;
29     foreach $\tau 0 \in Cr$ do
30         if $C$ contains $\tau 1$ where $\tau 0.id == \tau 1.id$ then
               // remote item exists in the local cache
31             if ($\tau 0.t == \tau 1.t$ and $\tau 0.o == \tau 1.o$) then
                   // remote item is identical, update local tuple
32                 $\tau 1 = \langle \tau 1.id, \tau 1.o, \tau 1.t,$
                    $\tau 1.vp + \tau 0.vp, \tau 1.wp + \tau 0.wp, \tau 1.va + \tau 0.va, \tau 1.wa + \tau 0.wa,$
                    $\tau 1.state \rangle$
33             else if ($\tau 0.t == \tau 1.t$ and $\tau 0.o < \tau 1.o$) or ($\tau 0.t < \tau 1.t$) then
                   // remote item is a duplicate, retain the oldest
34                 $\tau 1 = \langle \tau 0.id, \tau 0.o, \tau 0.t, \tau 0.vp + 1, \tau 0.wp, \tau 0.va, \tau 0.wa, \tau 0.state \rangle$
35         else
               // remote item not exists, add a copy to the local cache
36             $C \longleftarrow C \cup \{\langle \tau 0.id, \tau 0.o, \tau 0.t, \tau 0.vp + 1, \tau 0.wp, \tau 0.va, \tau 0.wa, \tau 0.state \rangle\}$
```

the maximum communication latency corresponds to the round trip time on the diameter of the network, and in principle, some messages may take very long to arrive and deliver in later cycles (*Out-of-Cycle Message*). However, out-of-cycle messages slightly delay the convergence in SSEP and PTP protocols due to potential loss of mass in transportation, which is restored when out-of-cycle messages deliver.

All protocols introduced in this section are simulated using the event-driven engine in PeerSim, where two common events are defined. The Run Event occurs at every cycle, and in this event, generating of information items, sending Push messages and making the transition among phases are performed. The event stops when a maximum number of cycles is reached. The Message Event occurs when a node receives a message from a peer. At this event, the incoming message is processed.

The simulations of PTP included the validation of generating single information item and of generating multiple items. Due to randomization, the generation of new items in the PTP is adjusted to stop after the completion of 50% of the experiment cycles to observe the protocol performance in the residual cycles. The propagation of a single item is shown in Figure 4.3, where the percentage of nodes that have achieved a particular phase for the item is illustrated. Furthermore, experiments on the single item propagation have investigated the impact of changing the thresholds ϵ and Υ values. Figure 4.4 summarises the results and shows the number of cycles needed to complete a phase transition for an item when varying Υ and the value of ϵ.

From the results presented in Figure 4.4, it is apparent the increase in the number of cycles required to achieve convergence when values of Υ and ϵ are set for higher accuracy. The results support the previous analysis for choosing adequate values for the application-specific parameters and thresholds to avoid faster incorrect convergence and to prevent penalising epidemic protocols. Figures 4.4.a, 4.4.b, and 4.4.c summarise the results of running the PTP protocol for 30 times under different values of Υ and ϵ. The illustrated values are the average of exported results and the error bars shows the variance among the results.

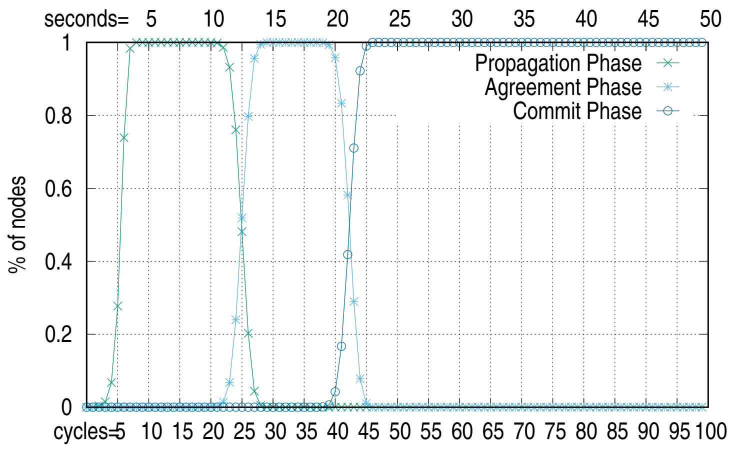


Figure 4.3: Percentage of nodes at each phase for a single information item in PTP, $N = 10^4$, $\epsilon = 0.001$, $\Upsilon = 5$, $k = 10$, $\ddot{\Upsilon} = 500ms$.

The convergence of the Propagation and Agreement phases in PTP is demonstrated in Figure 4.5. For a single information item, Figure 4.5.a shows the variance of local estimates over all nodes; while Figure 4.5.b shows the average of the estimates in the system. The results show a quick reduction in the variance and the average of estimates indicating the correct convergence to the target value. Moreover, PTP is examined for the propagation of 50 distinct items in the presence of item duplication. Figure 4.5.c shows the variance of the estimates over all items and all nodes; while Figure 4.5.d shows the average of the estimates for all items and nodes too. It can be inferred that the protocol correctly manages items duplication and that local estimates in the nodes correctly converge to the system size N in both phases of PTP.

In conclusion, PTP in particular and the PTA in general provide an adequate solution for the decentralised consensus problem, and they can achieve global agreement and coordination

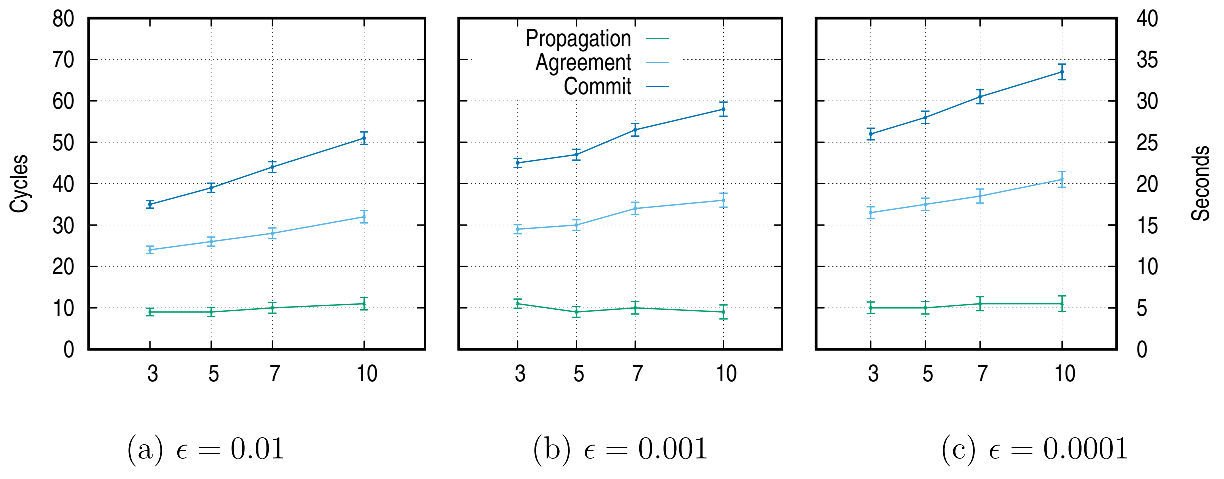


Figure 4.4: Number of cycles to complete a phase transition in PTP for a single item varying the cycles threshold $\Upsilon \in \{3, 5, 7, 10\}$ and the tolerance threshold ϵ, $N = 10^4$, $k = 10$, $\ddot{\mathfrak{T}} = 500ms$.

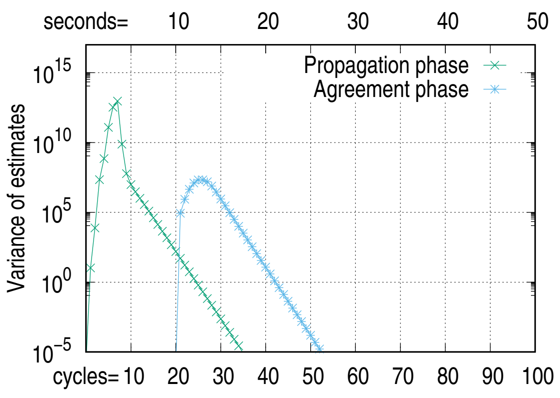
without deterministic communications such as those in centralised approaches. The PTA is flexible and can have more phases to acquire additional certainty on the coordination target. The certainty of the PTA comes with noticeable overall communications overhead. However, the overhead is distributed over the network diameter and execution time, and it is only two exchange messages per node at every cycle. The solution is epidemic-based and inherits their features such as scalability and fault-tolerance, which are critical for services in extreme-scale distributed systems.

4.5 Agreement in Epidemic Data Aggregation

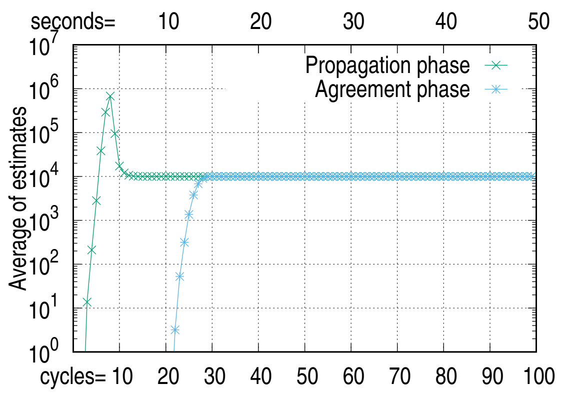
Data aggregation is essential for a wide range of services, especially in distributed systems. The aggregation process is typically independent of the overlay topology and the underlying networks, which gives the aggregation process the flexibility and applicability for various applications. For example, the distributed aggregation process can be used to calculate system size, resource capacity, and average uptime [26, 27]. Also, it is used to achieve distributed consensus, consistency, and coordination [48, 78, 113]. Epidemic models for data aggregation are scalable, which made many distributed services to adopt epidemic data aggregation models. For example, for failure detection [68], for distributed data mining [66], for global attribute computation [113], and for coordination and consistency [35, 39].

As described earlier in Section 4.1, distributed data aggregation protocols compute a global synopsis for an aggregate function over a distributed set of data values [26, 58]. The aggregation protocol exchanges local values among nodes to compute a particular function. Eventually, all nodes in a system converge to the same output $\mathcal{V}$, which is the aggregation target. The process of epidemic data aggregation and its convergence is described in Section 3.2.

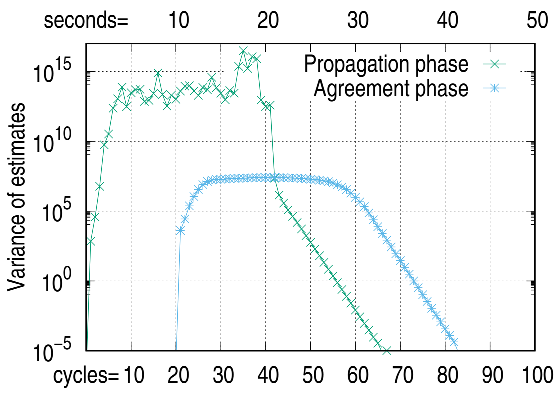
The problem presented in this section is the extension of the distributed consensus problem to distributed data aggregation problem where the agreement process is an epidemic



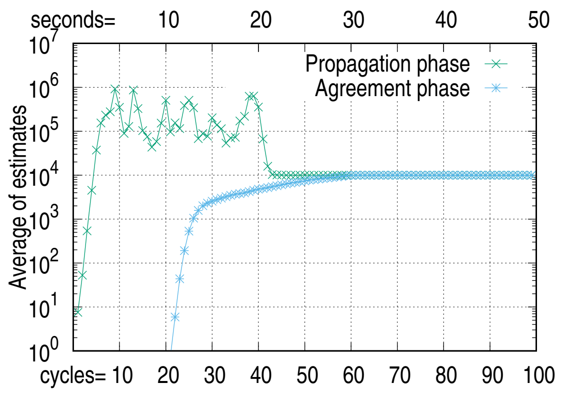
(a) Variance of estimates for a single item



(b) Average of estimates for a single item



(c) Variance of estimates for multiple items (50 distinct items)



(d) Average of estimates for multiple items (50 distinct items)

Figure 4.5: The convergence of item estimates in PTP phases, $N = 10^4$, $\epsilon = 0.001$, $\Upsilon = 5$, $k = 10$, $\ddot{\mathfrak{T}} = 500ms$.

aggregation task. In order to achieve global agreement on a particular target, nodes in the system have to (1) compute a local estimate of the target value using a specific data aggregation function, (2) detect local convergence on the target, (3) acquire explicit awareness on the convergence of the system. A typical example of the proposed agreement process in epidemic systems is the local estimate of a data synopsis function, where a node may need to achieve three different levels of information, the target value $\mathcal{V}$ with some approximation, awareness of local convergence to the target and certainty of a global convergence on the target. The main contribution presented in this section is the proposing of the Epidemic Consensus Protocol (ECP). The protocol ECP is a typical implementation of the PTA for achieving consensus on epidemic data aggregation with a high certainty on the convergence of the system.

4.5.1 Epidemic Consensus Protocol (ECP)

ECP consists of four subsequent phases: Aggregation, Convergence, Agreement and Commit. The Aggregation phase is the primary task process and it is determined by a service or an application. The three successive phases compose the agreement process. ECP maintains a tuple containing three aggregation pairs (vd, wd) for primary aggregation process, (vc, w) and (va, w) for the aggregation processes of Convergence and Agreement phases respectively. The tuple also contains *leader* for the computation of the election. The leader election is explained later in this section. Also, ECP requires two other services to run in parallel with the protocol, size estimation service, e.g. SSEP and peer-sampling service, e.g. NCP$^+$.

In the Aggregation phase, ECP computes the global average for the distributed set of initial data values in the system, although any other basic or complex aggregation can be accomplished. Each node i initialises the pair (vd_i, wd_i) where vd_i is set to local value x_i and the weight is set to $wd_i = 1$. At each cycle, node i divides the values of the pair $(\frac{vd_i}{2}, \frac{wd_i}{2})$ and the half is sent to a random peer alongside other aggregation pairs in a Push message. Upon receiving a message from node j, node i divides the pair values, and the half is sent to node j in a Pull message with other pairs values. Also, node i updates its local pair values $(vd_i, wd_i) = (vd_i + vd_j, wd_i + wd_j)$. In consequence, the initial pair $(x_i, 1)$ at each node i is divided and evenly distributed to the entire system. After a number of cycles, data values eventually converge to $\frac{1}{N}\sum_{i=1}^{N} vd_i$ and weight values converge to $\frac{1}{N}\sum_{i=1}^{N} wd_i = 1$. An approximation of the global average can be obtained at each node i by $\frac{vd_i}{wd_i}$ at any cycle.

The Convergence phase is the first phase in the agreement process. In this phase nodes attain awareness about the convergence of other nodes in the Aggregation phase. Following the PTA, the explicit global awareness is obtained using the decision-making method by aggregate the count of nodes in the Aggregation phase which have achieved convergence to the target. For this purpose, each node i holds the pair (vc_i, w_i), where $vc_i = 1$ at all nodes, $w_{\hat{\imath}} = 1$ at a single node $\hat{\imath}$ and $w_i = 0$ at all other nodes, $0 < i \leq N, i \neq \hat{\imath}$. Eventually, $w_{\hat{\imath}}$ will distribute equally in the system and the weight w in each node will converge to $\frac{1}{N}$. Also, an estimation of the number of nodes in the phase is given by $\frac{vc_i}{w_i}$ at every node i, whereas $\frac{1}{N}\sum_{i=1}^{N} vc_i = 1$ and $\frac{1}{N}\sum_{i=1}^{N} w_i = \frac{1}{N}$.

The next phase is the agreement process is the Agreement phase that follows the same procedure as in the Convergence phase, but this time to acquire the certainty on the global agreement. The pair (va_i, w_i) is used for the decision-making process in the phase. Noticeably, the weight w_i is used in the pairs of the Convergence and the Agreement phases for the optimisation as described later in this section. Finally, the agreement process in ECP ends in the Commit phase, which implies the achieving of the global agreement, the reach to the consensus, and the right moment to take a system-wide decision or action.

In the data aggregation, the target value $\mathcal{V}$ is unknown to the aggregation process unless it has been provided explicitly by the service or application. Also, ECP has no information about the target value for the aggregation process in the Aggregation phase. In realistic scenarios, it is challenging to obtain the correct target value beforehand, especially under dynamic conditions. Therefore, an innovative detection method is required for the ECP to allow each node has the protocol to locally detect the convergence. ECP adopts the heuristic detection method of Poonpakdee et al. [94] with the CV as described in Section 3.3.

Algorithm 9: Epidemic Consensus Protocol (ECP)

```
Require: a size estimation service, e.g. SSEP; a peer-sampling service, e.g. NCP$^+$;
         tolerance thresholds $\epsilon_1$, $\epsilon_2$; cycles threshold $\Upsilon$; queue length $l^{\mathcal{Q}}$;
Initialisation: at each node $i$, $vd = x$, $wd = 1$, $vc = 0$, $va = 0$, $w = 0$;
                $phase$=AGGREGATION; $\mathcal{Q} \longleftarrow \emptyset$ and $|\mathcal{Q}| = l^{\mathcal{Q}}$; $leader = i$;

At each cycle at node $i$:
$j \longleftarrow getRandomPeer()$
$vd = \frac{vd}{2}$, $wd = \frac{wd}{2}$,    $vc = \frac{vc}{2}$, $va = \frac{va}{2}$, $w = \frac{w}{2}$
send $\langle leader, vd, wd, vc, va, w, reply = true\rangle$ to $j$     // PUSH message to node $j$
/* detect convergence and make phase transitions                                  */
switch $phase$ do
    case AGGREGATION do
        if $\frac{\mathcal{Q}.s}{\mathcal{Q}.\bar{e}} \leq \epsilon_1$ for at least $\Upsilon$ cycles then
            /* make transition to the CONVERGENCE phase                         */
            $phase$ = CONVERGENCE
            $vc = vc + 1$
        if leader has not changed for at least $\Upsilon$ cycles then
            if $leader == i$ then $w = 1$
    case CONVERGENCE do                                 // attain global convergence
        if $\left|\frac{size() - \frac{vc}{w}}{size()}\right| \leq \epsilon_2$ for at least $\Upsilon$ cycles then
            /* make transition to the AGREEMENT phase                          */
            $phase$ = AGREEMENT
            $va = va + 1$
    case AGREEMENT do                                    // attain global agreement
        if $\left|\frac{size() - \frac{va}{w}}{size()}\right| \leq \epsilon_2$ for at least $\Upsilon$ cycles then
            $phase$ = COMMIT
    case COMMIT do
        /* take some application-specific decision or action                */

At event 'received $m$ message from $j$' at node $i$:
if $m.reply$ then                                       // $m$ is a PUSH message
    $vd = \frac{vd}{2}$, $wd = \frac{wd}{2}$,    $vc = \frac{vc}{2}$, $va = \frac{va}{2}$, $w = \frac{w}{2}$
    send $\langle leader, vd, wd, vc, va, w, reply = false\rangle$ to $j$  // PULL message to node $j$
$\mathcal{Q} \longleftarrow \mathcal{Q} \cup \{\frac{vd}{wd}, \frac{m.vd}{m.wd}\}$                        // enqueue estimates to $\mathcal{Q}$
$vd = vd + m.vd$, $wd = wd + m.wd$,                       // update local pairs
  $vc = vc + m.vc$, $va = va + m.va$, $w = w + m.w$
$leader = max(leader, m.leader)$                          // select a leader node
```

The convergence in the Aggregation phase is detected as follows. Each node i maintains a fixed length history queue $\mathcal{Q}_i$ and uses $\mathcal{Q}_i$ to store the local estimate $e_{i,t}$ and a remote estimate $e_{j,t}$ after receiving a message from node j at cycle t, $\mathcal{Q}_i \cup \{e_{i,t}, e_{j,t}\}$. At each cycle, each node computes the local estimation error $\varepsilon_{i,t}$ using the formula of the CV, $\varepsilon_{i,t} = \frac{\mathcal{Q}_i.s}{\mathcal{Q}_i.\bar{e}}$ for all elements in $\mathcal{Q}_i$, where $\mathcal{Q}_i.s$ is the standard deviation (formula 3.7) and $\mathcal{Q}_i.\bar{e}$ is the average (formula 3.6). The local convergence is detected upon verifying the criterion $\varepsilon_{i,t} \leq \epsilon_1$ for a number of consecutive cycles Υ. On another hand, local convergence detection in the phases of the agreement process uses the general detection method and the formula 3.3.

There are two error tolerance thresholds in the ECP ϵ_1 and ϵ_2, to allow different accuracy requirement to be applied for the primary aggregation and agreement processes. The error threshold ϵ_1 is used in the convergence detection of the Aggregation phase and ϵ_2 is used in all the subsequent phases. The determination of the thresholds is an application requirement that trades-off convergence speed to the desired accuracy. Also, the consecutive cycles threshold Υ is used to avoid precocious convergence detection in all phases.

In ECP, an epidemic leader-election process is performed to appoint a single node as a leader [30, 32]. The leader node $\hat{\imath}$ is used to set the weight value $w_{\hat{\imath}}$ to $w_{\hat{\imath}} = 1$ as an initialisation requirement for the summation aggregation [26, 30]. Leader-election process has a simple assumption that each node has a unique identifier, e.g. IP address. The leader is the node with the highest identifier. The convergence to a leader node is achieved when the local *leader* estimate holds for Υ cycles. Leader-election process is performed in parallel with the Aggregation phase to allow early propagation of $w_{\hat{\imath}}$ and enabling faster convergence in the subsequent phases.

The ECP is illustrated in Algorithm 9. At each cycle, the protocol divides the aggregation pair of each phase and sends them to a random peer (lines 1-4). In lines 5-19, the protocol detects convergence of each phase and makes the transition upon verifying the associated criterion. Also, the protocol detects the convergence of the leader-election process in line 10, and sets the value of w to 1 at the leader node. At the event of receiving a message, ECP responds in the lines 20-13, and in line 24, the protocol enqueues the local and remote estimates of the Aggregation phase in $\mathcal{Q}$ for the convergence detection in the next cycle. In line 25, the protocol updates the local pairs and selects a new leader in line 26.

4.5.2 Experimental Results for ECP

Simulations model is fully event-based and uses the event-driven engine in PeerSim. Two events are defined in the model: (1) The Run Event is scheduled to occur at every cycle and stops after a predefined number of cycles. At this event, a node contacts a random peer and makes the transition among phases. (2) The Message Event occurs when a node receives a message from a peer. At this event, the incoming message is processed. The local aggregation pairs are updated, and a leader is selected.

The simulation model includes four protocols, ECP, PTP$^+$ and two tree-based 3PC protocols. The protocol PTP$^+$ is a typical implementation of the PTP as described in

Section 4.4. However, PTP$^+$ has an additional data aggregation phase that assumes a single aggregation item exists at all nodes at the start of the system, and the target is to achieve the consensus on the aggregation item. PTP$^+$ is used in the simulations to analyse the performance of PTP$^+$ in comparison to ECP.

The protocol 3PC is simulated over a static binary tree [114]. The simulation of 3PC protocol is made purposely over a binary tree to achieve fair performance comparisons with the ECP, which sends two messages in average per-node at every cycle. Nodes identifiers in 3PC protocol are assumed globally unique and incremental. The node with identifier 0 is the coordinator node and each node i has two child nodes $2i + 1$ and $2i + 2$. The tree is constructed in a binary structure by a dedicated initialiser prior to the simulations. 3PC is implemented to achieve the global agreement on the outcome of the distributed averaging over the same data distribution that is used in ECP and PTP$^+$. Two versions of 3PC protocol are used in the simulations, a classic 3PC and a modified version of 3PC motivated by the CTP [109] namely Three-Phase Commit-Convergecast or (3PC-C). In 3PC-C, the step of broadcasting *compute* message is omitted and the protocol starts in a Convergecast step from the depth of the tree towards the coordinator. The subsequent phases of 3PC-C proceeds as same as in classic 3PC. This optimisation is expected to improve the total time required for 3PC-C to achieve consensus and thus challenge the performance competition with ECP.

The performance and communication overhead of ECP, PTP$^+$, 3PC and 3PC-C are monitored at every cycle using dedicated observation modules. All protocols continue after the global agreement is achieved, and a simulation experiment terminates when the total number of cycles is reached. Different random seeds are used in each experimental run to enforce randomisation and each experiment is repeated for tens of times to validate the settings and ensure results. The protocols are initialised by a Peak data distribution where $vd_{\hat{\imath}} = N$ in a single node $\hat{\imath}$ and $vd_i = 0$ where $0 < i \leq N$ and $i \neq \hat{\imath}$. Application-specific parameters for ECP and the other protocols are carefully chosen based on the experimental results which are described later in this section. The error thresholds ϵ_1 and ϵ_2 are set to $\epsilon_1 = \epsilon_2 = 0.01$. The cycle threshold is set to $\Upsilon = 5$. The length of the history queue $\mathfrak{Q}$ is set to $l^{\mathfrak{Q}} = 10$. NCP$^+$ is configured to maintain a random k-regular overlay with $k = 10$.

The detection of convergence and the phases transition towards the global agreement in the ECP is illustrated in Figure 4.6. Figure 4.6.a shows the percentage of nodes in each phase over time. The figure also illustrates the smooth transition from a phase to the successive. Figure 4.6.b shows the average of estimates in each phase. It is apparent that estimates in each phase converge to the same target at all nodes. In the Aggregation phase, local estimates converge to 1 which is the correct target value for spreading $vd_{\hat{\imath}} = N$ over N nodes. Estimates in the Convergence and Agreement phases converge to N as expected. In Figure 4.6.c, the variance of estimates over all nodes is tending towards a very small value indicating the reduction in estimation error and the reach of convergence among nodes in each phase. Results in Figure 4.6 validates the ability of ECP to locally detect convergence, makes the transition in phases and attain the certainty of the global agreement on the outcome of

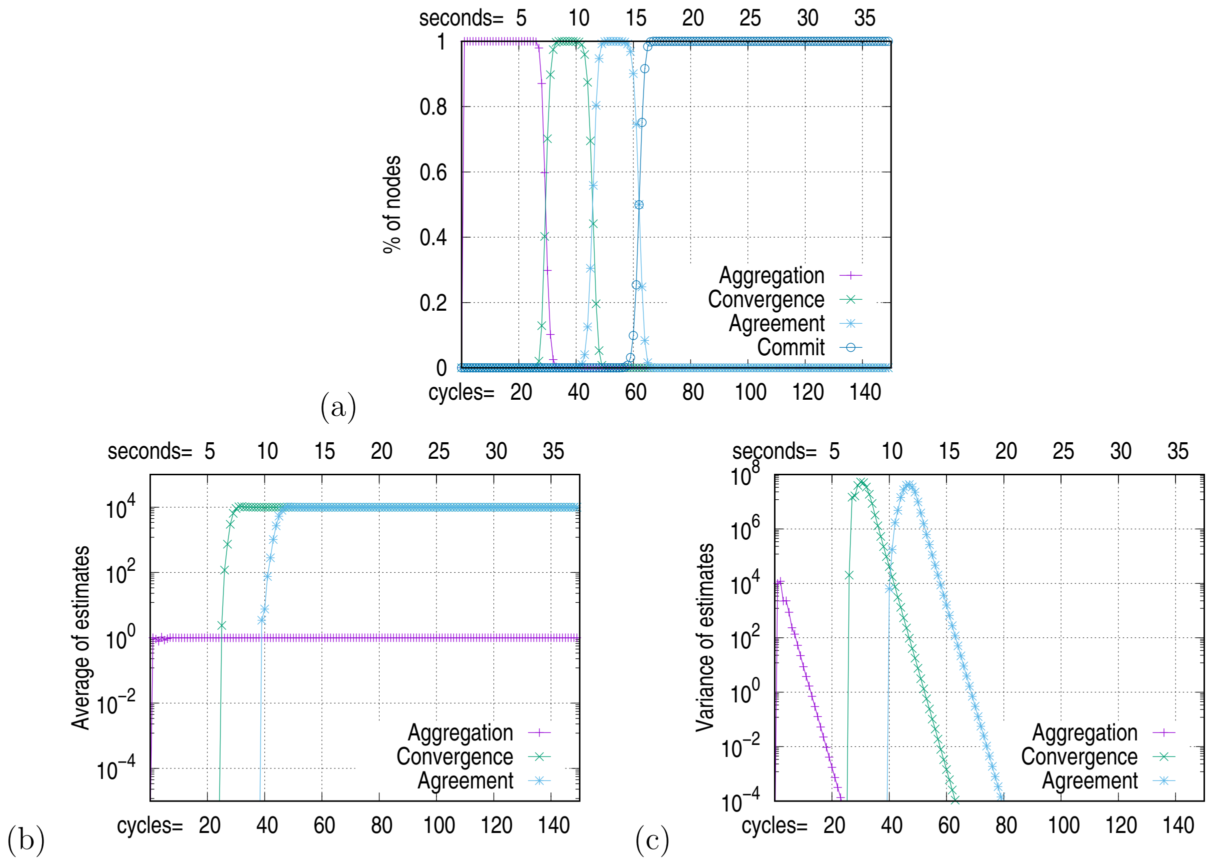
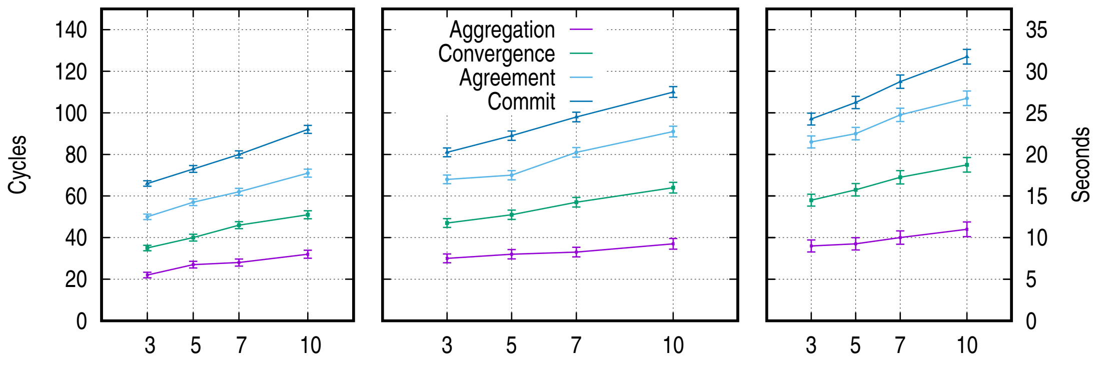


Figure 4.6: Convergence detection and phases transition in ECP, $N = 10^4$, $\epsilon_1 = \epsilon_2 = 0.01$, $\Upsilon = 5$, $l^\Omega = 10$, $k = 10$, $\dot{\mathfrak{T}} = 250ms$.

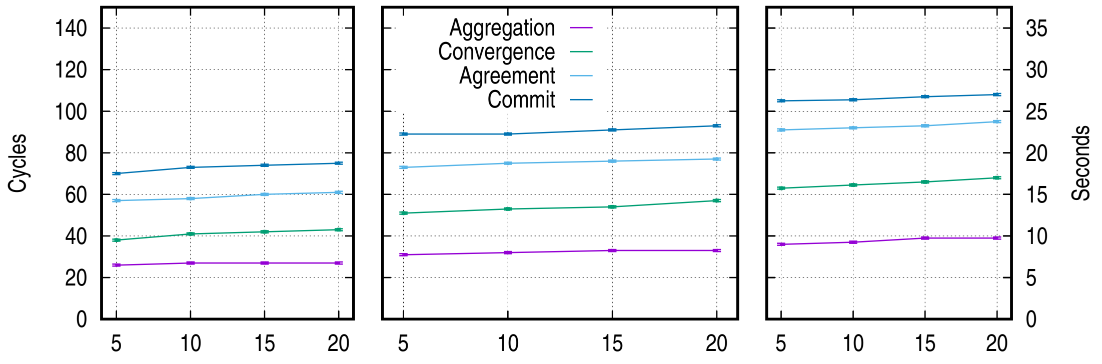
the global averaging.

The performance of the ECP is also examined under different settings of the application parameters. The error thresholds ϵ_1 and ϵ_2 are used in different phases, the effect of each one can be recognised by monitoring the convergence of the corresponding phase, and hence, both parameters are set to the same value. Figure 4.7 shows linear rising in the completion times of each phase when error thresholds are set for a higher accuracy. The values of ϵ_1 and ϵ_2 can be tuned to trade-off between accuracy and convergence speed. For instance, a repetitively small error threshold can be used in Aggregation phase, while some accuracy can be tolerated in the CONVERGENCE and AGREEMENT phases to speed up the convergence. On another hand, the use of higher values for the parameter Υ significantly slows the detection of convergence in each phase. Thereby, Υ is set to 5 which allows feasible convergence speed for large network sizes up to one million nodes. In contrary, a small delay in the completion time in the AGGREGATION phase is noticed when the length l^Ω of the history queue increases as shown in Figure 4.7.b. and thus the use of minimum reasonable length is preferable for faster convergence and less computation load. The results presented in the Figures 4.7.a and 4.7.b are summery of 30 runs of the protocol ECP. The illustrated values are the average of exported results and the error bars shows the variance among the results..

Figure 4.8 shows the phase transition and the completion times of the protocols ECP, PTP$^+$, 3PC and 3PC-C. The summary in Figure 4.8.a shows the linear increase in the



$\epsilon = 0.01$ $\epsilon = 0.001$ $\epsilon = 0.0001$
(a) Varing values of the cycles thershold $\Upsilon \in \{3, 5, 7, 10\}$
for three values of the error thershold ϵ,where in simulations $\epsilon = \epsilon_1 = \epsilon_2$.



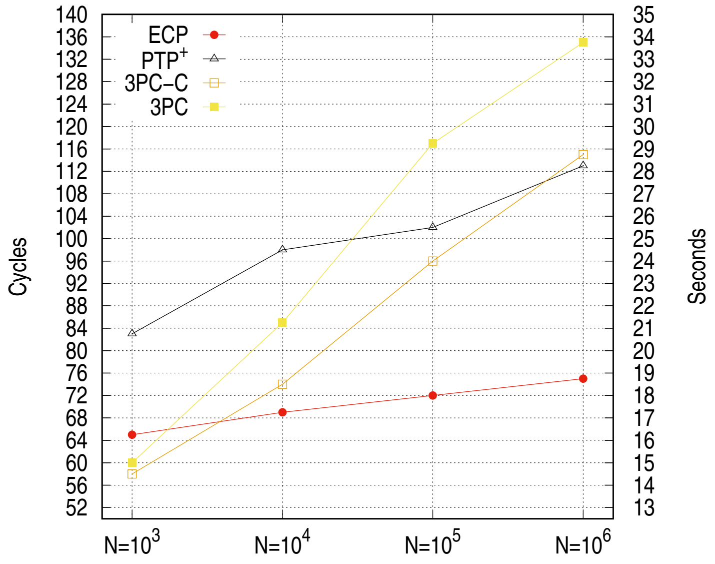
$\epsilon = 0.01$ $\epsilon = 0.001$ $\epsilon = 0.0001$
(b) Varing the legth $l^{\Omega} \in \{5, 10, 15, 20\}$
for three values of ϵ, where in simulations $\epsilon = \epsilon_1 = \epsilon_2$.

Figure 4.7: Phases completion times in ECP when varying simulations parameters, $N = 10^4$, $k = 10$, $\dot{\mho} = 250ms$.

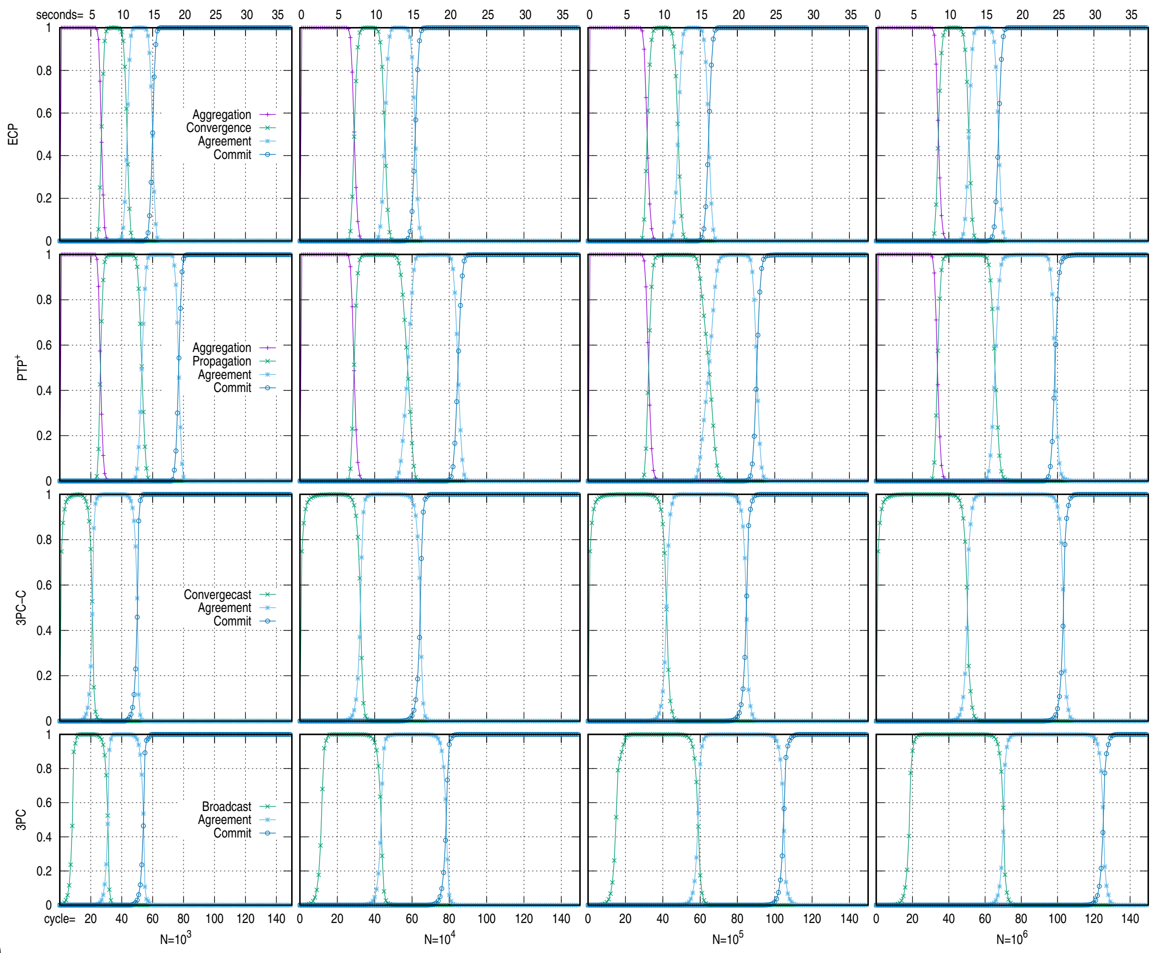
completion times in ECP concerning the system size. However, it shows faster completion times in comparison to PTP$^+$, which presents the impact of the early leader election mechanism in ECP. The completion times of 3PC and 3PC-C protocols significantly rise with respect to the system size. The spanning of the tree depth increases the convergence time in 3PC and 3PC-C protocols. The ECP shows similar performance to other protocols in small sizes but performs much better in large sizes. Also, ECP outperforms tree-based 3PC and 3PC-C protocols without the need to establish any network structures. A sample of results for the protocols performance over various system sizes is presented in Figure 4.8.b.

The communication overhead in ECP, 3PC and 3PC-C protocols are illustrated in Figure 4.9. Figure 4.9.a shows the average number of messages at each cycle in ECP. Each node sends two messages in average at every cycle, one to contact a random peer and one to reply to an incoming message. Figure 4.9.a illustrates the distribution of communication load among all nodes in ECP. Figure 4.9.b illustrates the total number of sent messages in each phase. ECP produces higher overhead in each phase due to the continuous communication nature of epidemic protocols.

The communication overhead in tree-based 3PC and 3PC-C protocols are shown in Figures



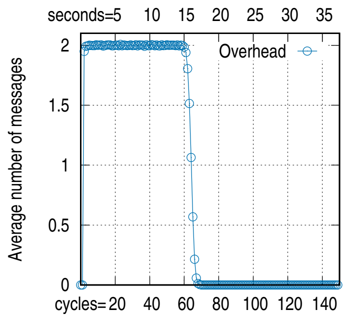
(a)



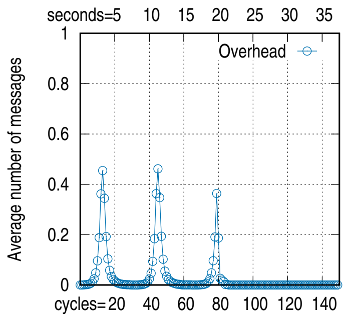
(b)

Figure 4.8: Phases transition and completion times in ECP, PTP$^+$, 3PC and 3PC-C protocols for various system sizes, $\epsilon_1 = \epsilon_2 = 0.01$, $\Upsilon = 5$, $l^\Omega = 10$, $k = 10$, $\dot{\mho} = 250ms$.

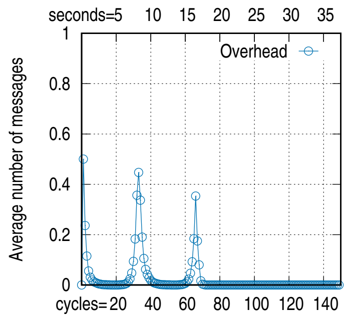
Average communication overhead per-cycle



(a) ECP [1]

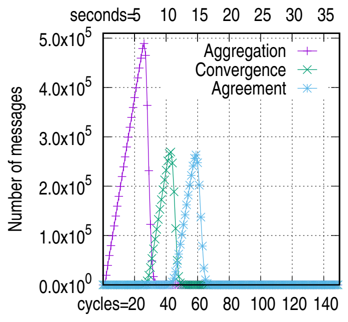


(c) 3PC

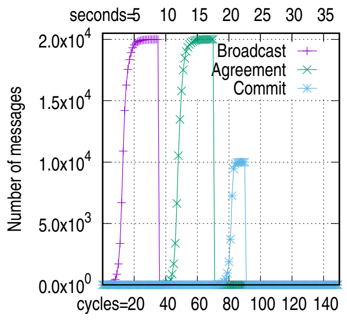


(e) 3PC-C

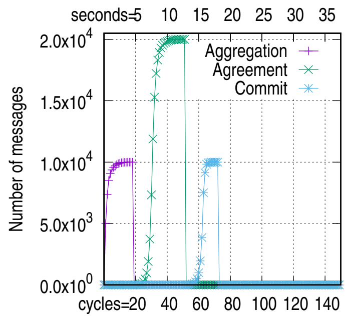
Total communication overhead in each phase



(b) ECP



(d) 3PC



(f) 3PC-C

Figure 4.9: Communication overhead in ECP, 3PC and 3PC-C, $N = 10^4$, $\epsilon_1 = \epsilon_2 = 0.01$, $\Upsilon = 5$, $l^{\Omega} = 10$, $k = 10$, $\dot{\mathfrak{T}} = 250ms$.

4.9.c-4.9.f. In general, nodes in 3PC and 3PC-C protocols send two *compute* messages to child nodes in Broadcast step and two acknowledgement messages are sent back to the parent node in Convergecast step in each phase. Broadcast messages and Convergecast messages occur at different cycles and encounter different latency, hence two messages are sent in average per-node at each cycle. As the structure of the tree expands towards the depth, the number of messages increases in the Broadcast step and decreases in the Convergecast step until nodes reach the global agreement. At the most depth of the tree, the number of sent messages is $\frac{N}{2} + 1$ and thus in the corresponding cycle, around $\frac{1}{2}$ message is sent in average per-node. Figures 4.9.c and Figure 4.9.e illustrate this effect in 3PC and 3PC-C protocols. Figures 4.9.d and Figure 4.9.f, shows the total number of messages in each phase which is noticeably less than ECP protocol. In summary, ECP achieves consensus faster than tree-based protocols in large system sizes. Tree-based protocols have optimal communication overhead whilst ECP has higher overall overhead. The overhead in ECP is distributed over system nodes and execution time and hence, the per-node and per-cycle communication overhead is perfect.

The conclusion is the presentation of an innovative protocol for the consensus, consistency

[1]In this experiment, nodes of ECP are enforced to stop in Δ cycles after the detection of convergence, where $\Delta > \Upsilon$.

and coordination in epidemic systems. Services and applications looking for a practical, decentralised, scalable and fault-tolerance coordination solution should consider the epidemic consensus protocol. The ECP is introduced for the determination of the consensus on the convergence of distributed data aggregation, and it can be altered for any distributed coordination task. Also, the protocol adopts a novel method for the local detection of convergence and makes phase transitions to achieve the explicit detection of convergence and global agreement. Experimental results have shown that ECP can achieve global agreement in lesser time in comparison to structured and centralised protocols that are not fault-tolerance and require structure maintaining. However, ECP produces higher communication overhead, which is typically distributed over the network diameter in extreme-scale systems. Finally, ECP is a typical implementation of the PTA and has proved the applicability of the algorithm. Further research may amend PTA to attain consistency and coordination in dynamic systems.

4.6 Discussions

The previous sections have introduced the phase transition algorithm PTA for achieving distributed consistent states in large and extreme-scale epidemic systems. Also, they have presented two novel epidemic protocols PTP and ECP, which are practical implementations of the PTA. Experimental results and the analysis of the performance of PTP and ECP have proven their ability to achieve global consistency in various epidemic tasks and with the explicit detection of the global agreement. The results have validated the applicability of the PTA and its performance in practice. The PTA is flexible, and the agreement phases can be cascaded to achieve additional certainty and global awareness on the target. The algorithm inherits the intrinsic properties of epidemic algorithms and can achieve the global agreement without deterministic communication or network structures. It is a typical decentralised, scalable and fault-tolerant solution to modern services in extreme-scale distributed systems. This section discusses some findings and matters of the PTA and its implementations, i.e., PTP and ECP.

- A note on the time complexity of the protocols PTP and ECP. Each protocol makes the transition into phases to achieve the globally consistent state, i.e., global agreement or COMMIT phase. Each phase is an epidemic process that eventually converges to the target value. Following the analysis of the convergence in epidemic protocols described in Section 3.2 and the general time complexity of PUSH-PULL protocols [26], the time needed for each protocol to achieve the consistent state is $O(M \log(N))$, where M is the number of phases in the protocol and N is system size. And the time t_{action} in which system nodes achieve the consistent state is defined as follows:

  Definition 4.6.1. Let tc be the convergence time of a phase defined as $tc = \log(N) + \log(\frac{1}{\epsilon}) + \Upsilon$. There is time $t_{action} = M \times tc$ such that for each $t \geq t_{action}$ each node i has achieved COMMIT phase and can apply a designated action or decision.

Epidemic protocols usually exhibit high communications overhead due to their intrinsic nature. Protocols of PTA have a typical overhead with the general complexity $O(2N)$. As shown in the experimental results, the protocols PTP and ECP generate only 2 messages per node per cycle. System nodes are widely distributed over the system network. Hence, communications overhead is spread over the whole network as well, which minimises the probability of traffic congestions and bottlenecks. However, the amount of messages required per each node to achieve a global consistent state on a target is $O(2t_{action})$, and the overall overhead is $O(2Nt_{action})$.

- In an epidemic system, services usually need to perform an action or take a decision after achieving global agreement using one of the PTA protocols. Nodes of an epidemic system do not converge at the same time due to the asynchrony, and nodes have no explicit knowledge of the convergence of other nodes. During PHASE-2, nodes achieve global agreement on the target of the process in PHASE-1 at different times and have no awareness about the state of nodes in PHASE-2. Therefore, it is critical to all nodes to converge in PHASE-2 to correctly achieve global agreement. Nodes that promptly implement a local action such as *stop* or *leave* after they detect global convergence can cause a detrimental effect on the aggregation process in PHASE-2 and prevent other nodes in the phase from achieving convergence. This effect is an expected result of violating mass-conservation invariant caused by the nodes that have stopped or left the phase. To avoid such effect, nodes must continue participating in the aggregation process for a sufficient time (e.g., tc) to allow other nodes in the phase to converge. At this stage, N is known to all nodes, and reasonable value of tc can be computed.

  In the same context, making a system-wide action or decision after achieving global agreement is also a matter, e.g. *restart* or *terminate*. The first node that detects global agreement in PHASE-2 can start the global action and begin the propagation of the action information. However, other nodes may receive information on the action before they achieve convergence, and in this case, they are enforced to apply the action. Any service that does not wish to apply the global action in the current phase can merely add another subsequent phase, and nodes in the new phase have to implement the action upon the convergence.

- In PTA, each phase (e.g., PHASE-2) in the agreement process is a separate decision-making process using the aggregate function *count*. The distinct property of the decision-making in PHASE-2 is the decentralised fashion of the decision. Each node decides to join the aggregation process upon the detection of local convergence in a prior phase, i.e. PHASE-1. In PHASE-2, each node decision propagates to other nodes until the aggregate results converge to the system size, i.e., all nodes have decided. Also, each node can detect the convergence of the phase locally, which is very convenient for large systems. Moreover, the method achieves the explicit global decision made by each participating node on the agreement target.

- In realistic scenarios, some nodes may wish to abort the agreement process due to a private event or reason. In this case, a node may withhold itself from joining the decision-making process, causing the process to converge to the incorrect value. Also, in dynamic conditions, some nodes may fail or leave before they decide, which prevent the decision-making process from achieving convergence. In a more complicated case, a node may wish to withdraw its decision after it has been made. Two cases need to be distinguished here, nodes that decide to abort the process voluntarily, and nodes that fail or depart the system non-reportedly.

  The decision-making method in PHASE-2 can detect abort and leaving requests of nodes by adding another aggregation process for the aggregate *count* to the phase. In essence, PHASE-2 would have two parallel aggregation processes, one for counting converged nodes and the other for counting nodes that want to abort or leave. Eventually, the aggregate results of the two processes can be used to detect the convergence of the phase. Moreover, the convergence of the second aggregation process to non-zero results implies a request to abort the agreement process, in which a predefined action may need to be globally executed. However, the two proposed decision-making methods are not sufficient and will not converge under dynamic conditions [34].

  Typically, nodes may fail or depart in adversary way at any time during the agreement task. In consequence, all involved data aggregation processes may not converge, for instance, the SSEP and phases of the PTP and the ECP protocols, which degrades the applicability of PTA in real-world systems. Data aggregation is the core of the agreement protocol, and hence, the robustness of the aggregation process requires practical solutions. Two techniques are proposed in this project to deal with the robustness of epidemic data aggregation in dynamic systems. An algorithm for instant support of the epidemic aggregation process during the convergence time is described in the next chapter. Another mechanism to detect churn impact on the convergence and adapts the epidemic task to new system conditions is described in Chapter 6.

- Phases of the agreement process in the PTA require global parameters for the convergence detection. In particular, the tolerance thresholds ϵ_1, ϵ_2 and the consecutive cycles threshold Υ are adjusted according to the application preference and need to be available beforehand. In real practice, settings of the application-specific parameters require global information about the system, which hard to obtain and may limit protocols applicability, especially in dynamic systems. The protocol ECP has used the heuristic detection method with the Coefficient of Variance as a formula for computing the estimation error, which does not need any global information and taking advantage of the evolved method, which is described in Section 3.2.

- In other respect, the setting of aggregate function in data aggregation requires some deterministic initialisation, especially, adjusting the data aggregation process for the summation function (i.e., *sum* and *count*). In the PTA, the protocol SSEP and the aggregation process in the agreement phases uses the global aggregate function *count*, in which the initial weight w_i of the aggregation pair (v_i, w_i) needs to be set to 1 at a single node $\hat{\imath}$ and to 0 at all other nodes, $0 < i \leq N, i \neq \hat{\imath}$. The determination of node $\hat{\imath}$ is problematic. For example, the protocol ECP uses leader-election mechanism to initialise the phases in the agreement process. Also, node $\hat{\imath}$ can be selected randomly with some defined probability. However, the problem arises when the node $\hat{\imath}$ fails before starting the diffusion in the aggregation process. To overcome this initialisation issue, we introduce a seed selection method in Chapter 6.

Insight to future, the PTA and the PTP, in particular, can achieve explicit consensus on each information item, and hence, the PTP can be adapted to maintain the consistency of transactions information, especially when the total ordering of transactions is required. Consider, for example, the Blockchain technology, in which certainty on data immutability in the distributed ledger is required [38]. Each transaction information in the Blockchain platform needs to be applied at most once at each node of the distributed ledger and any two updates have to be applied in the same order at all ledger nodes. The PTP and via continuous communication among ledger nodes can exchange new transactions, and the agreement process can acquire the certainty on each transaction. The decentralised decision-making method can either verify the transaction or abort the agreement process due to incorrectness of the transaction or due to violation of transactions order.

In summary, this chapter has introduced the PTA for achieving globally consistent states in epidemic systems. Also, two epidemic protocols for the practical implementation of the PTA are proposed in this chapter. Experimental results have proven the efficiency and applicability of the protocols in PTA for epidemic systems in particular and decentralised services in general. The protocols are capable of achieving the consensus, maintain the consistency, and attaining any other coordination service. The PTA is flexible and can be adapted to acquire explicit certainty on the target. The data aggregation process is the main foundation of the PTA, and convergence detection is the key feature of the aggregation processes. However, the data aggregation process can damage in the presence of dynamical conditions, and the agreement task as well. Therefore, further research is required to maintain the robustness of the data aggregation in unreliable systems. The next chapter presents the second contribution of the research work, which achieves robust epidemic data aggregation under churn.

Chapter 5

Robust Epidemic Aggregation under Churn

Distributed data aggregation has been addressed in many parts in the thesis due to the substantial role it has in the formation of distributed services in general and the proposed protocols in this project in particular. The data aggregation process is independent of the overlay topologies and the underlying networks, which makes the process flexible and efficient for a wide range of distributed applications [27, 48]. It usually computes a synopsis function over a set of distributed data values to provide global information about a system property or state. Nevertheless, epidemic data aggregation protocols are scalable, resilient and considered the practical alternatives to centralised models [20]. In the literature, there are many distributed services that have adopted epidemic data aggregation [52, 66, 68, 92].

Computations and communications of the epidemic data aggregation provide stochastic guarantees on the convergence of results to the target value in logarithmic time concerning the system size [26, 27]. However, the convergence to the target is expected under stable conditions. In dynamic systems where churn is usually present, the accuracy of the aggregation results cannot be guaranteed and the results may significantly differ from the correct target value [27, 115]. This detrimental impact of churn is a direct result of the violation of the mass-conservation invariant [30]. The failure or unreported departure of nodes determines the loss of the mass stored in these nodes, hence violating the conservation invariant and leads to an estimation error of the target value that depends on the local state at nodes departing the system.

There are number of contributions presented in this chapter. The next, Section 5.1 explains and discusses an inspection survey that has been accomplished in the research project aiming to collect information about ideal churn models and rates in the P2P networks. The Section provides the model of churn in epidemic system that is adopted for the project. In Section 5.2, our work has investigated the epidemic data aggregation process, in which three distinct phases of the process have been identified. The study has shown that node churn at each phase has a different impact on the estimation error. In particular, one of the three phases is critical for the robustness of the aggregation process, and it is further investigated, resulting in a novel mechanism to address the violation of mass conservation invariant. Consequently, we proposed two protocols for robust epidemic data aggregation in Section 5.3 and Section 5.4, respectively. The protocols implement distributed failure detection and mass restoration mechanisms that takes advantage of the findings of the analysis in the Section 5.2. The protocols have been verified and found able to detect convergence even under severe churn rates. Finally, the chapter ends discussing the outcomes of the work presented. The research in this chapter has been published in a full conference paper [34].

5.1 Node Churn in P2P networks

The analysis of churn impact on networks structure and systems behaviour is a popular research area that accrues significant attention. The general theme of the research was to correctly identify a churn model that is typical and realistic to common types of P2P networks. Therefore, researchers have extensively studied real-world network traces and revealed several empirical and statistical models of churn with good accuracy. Measurements on the node churn are required to evaluate churn impact on the epidemic systems, particularly, the expected churn rates in short periods (e.g. within one minute). The reason for analysing the studies to find a churn model of short time intervals that is compatible with the time needed by an epidemic task to accomplish the intended mission. The previous chapter has shown different epidemic tasks that can achieve global convergence in one minute. However, there are common impediments in the research studies, which made adopting a typical churn model a challenging task. For example, the churn rates notably vary from one study to another [15]. Also, and due to large system sizes and data acquisition methods, e.g. crawlers or trackers, the minimum time interval considered in the provided statistical distributions usually expand for several minutes. Moreover, the studies usually introduce their findings in synopsis measurements such as average, median, or CDF, which conceal vital information about churn rates in short intervals.

The survey work presented in this section aims to identify the typical churn rates of nodes in P2P networks in general and epidemic systems in particular. The hypothesis is that encountering a high or severe churn rate in a distributed system that is widely-deployed, uses decentralised services, and composed of many nodes is rarely to happen [15, 116]. In such systems, although nodes may fail or join a system, the failure and join events are either highly distributed or only affecting a specific part of the system. The P2P systems exhibit adequate resilience and fault-tolerance properties towards churn, and they are capable of providing services under various conditions [117]. Also, the survey targeted churn rates in short periods, particularly expected rates within intervals of one minute. However, acquiring precise information about global churn rates in short periods is not possible due to the way information is collected. Therefore, the results presented in this section serve as general indicators, and the adopted churn rates are expectations of the normal and worse scenarios.

In the studies, characteristics of a churn model are usually recognised by the duration of sessions in which nodes were active, and by the interval between the start of two sessions (a.k.a *inter-arrival* times). A node join a system to provide a service to other nodes or to consume other nodes services. After a certain time, the node departs the system either graciously or adversary. This join-participate-depart period is known as a *session* [15, 118]. Nodes sessions in P2P networks are known to have heterogeneous durations because some nodes may participate in a system for long periods while other nodes stay a little. Also, the number of participants in a system may vary during day times or among week days. In addition to sessions durations, some studies [15, 119, 120] have considered nodes *availability* as a key factor to characterise churn models in P2P networks. The availability of a node is defined as a ratio of total online sessions to the overall lifetime of the node which includes offline periods [116]. Although the study of nodes availability is crucial to some distributed applications (e.g. resource sharing [120]), we only address models of active and online sessions in this section. Furthermore, sessions are considered of various nodes and not necessarily of the same node, and each session is of a different node.

The description of a churn model usually comprises two separate rates: the *joining rate* and the *departure rate*. Principally, the joining rate corresponds to and can be computed from sessions inter-arrival times. The departure rate corresponds to intervals between the departure of different nodes or their failure, which is usually a result of combining inter-arrival times and sessions durations. However, the surveyed studies use session duration as a standard metric. The distribution of sessions durations can be used to estimate the departure rate under the following general assumptions: that a node departs without informing other nodes and each node departure is a fail-stop event. In this respect, the term *churn rate* refers to all node departure types, and the terms joining rate and churn rate will be used in the remaining of the thesis.

Let us consider a P2P system with N nodes. Assume that at a random time t, all nodes are online and each node would remain online for a session duration that is derived from a survival distribution function $R(t)$ [15, 119, 121]. After some time Δ, a number of nodes f have turned offline and departed the system. The failure rate λ is the rate at which nodes have departed the system during the interval $[t, t+\Delta]$ and so, $\lambda = \frac{f}{\Delta}$ is the probability of a node to fail per unit time. From this perspective, the characteristics of a typical failure distribution $F(t)$ can be identified from the sessions durations distribution $R(t)$. The probability of a node to remain in a system longer than Δ denoted as $P(T > \Delta)$ is given by $R(t)$ and the probability of a failure to occur before or at Δ is denoted as $P(T \leq \Delta)$, and it is given by $F(t) = 1 - R(t)$. The probability of a failure $P(T \leq \Delta)$ is typically the proportion (i.e. *percentage*) of nodes in a system that have failed before time $t + \Delta$.

Results illustrated in the studies have presented several probability distributions for the duration of sessions and inter-arrival times, in which a duration ranges from a few minutes to several hours. Also, cumulative distributions of the observed durations are provided, and they are usually tailed exponential distributions. The table 5.1 presents various churn rates estimation extracted from the studies for several P2P networks. The table also includes

Network	**Session Duration** $P(T \leq \Delta)$, $\Delta = 1$ **minute**	**Average Inter-Arrival Times**
Gnutella [15, 122]	15%	2 seconds
Bit-torrent [15, 122]	20%	10 seconds
Skype [14]	5%	30 minutes
OSN [118]	25%	less
Facebook [123]	30%	than
WWW [122, 124]	90%	one second

Table 5.1: Churn rates in the P2P and Social networks.

estimations for Online Social Network (OSN)[1]and the World-Wide Web (WWW), although these networks are not P2P, there are many P2P applications and services which are built to work for these networks, e.g. mobile apps.

The rates in Table 5.1 indicate that 30% of system nodes depart the system in one minute, which are the expected dynamical changes in normal working conditions of those P2P systems. The table also shows the churn rates in online services, which are notably higher. Also, join rate is greater than online services than conventional P2P systems. In general, churn rates in the table imply the need for robust epidemic systems, and that strong robustness techniques are required for online epidemic services. Also, having a typical strategy for dealing with nodes join within particular interval is essential.

From another perspective, in the surveyed studies, one minute is a relatively short interval of observation, which makes the estimated join rate and churn rate approximately constant in the interval. In consequence, it is practical to distribute the proportion of affected nodes uniformly over the interval, e.g. assuming a constant failure rate. Also, at any moment, several nodes depart the system while other nodes join it. Hence, N is changing over time, and the overall join rate equals the churn rate, which prevents the system from expanding forever and from vanishing [15]. A direct implication of this constant behaviour is that in addition to the robustness of the epidemic systems, they need to be continuous to adapt to change overtime.

The model of the epidemic system in this project is dynamic where nodes may join or depart the system at any time and with moderate rates. Nodes departure are not specified gracefully or ungracefully, and both cases are treated as generic node failure. A node failure is a fail-stop event, and nodes that may rejoin the system are considered new nodes. In the system, failing nodes may violate the initial system mass invariant and usually leave invalid cache entries in the peer-sampling service (a.k.a *broken-links*) [33], and churn may also affect the overlay structure. At this stage of the research work, new nodes may join the system at any time, but they do not participate in the current epidemic task. In next chapter, the

[1]The study in [118] includes *Linkedin*, *MySpace*, *Hi5*, and *Orkut*

continuous epidemic systems are proposed. A continues epidemic task consists of consecutive epochs, and new nodes restrain themselves during the current epoch and start contributing in the following epoch [27]. The fractions of joined and departed nodes in a specific time interval define the churn rates. The distribution of churn rates over time intervals is not assumed constant and there must be a time at which the system is sufficiently stable to allow convergence.

The remaining of this chapter describes two innovative epidemic protocols for robust data aggregation in the presence of node churn. The two protocols propose decentralised replication and failure detection mechanisms for the instantaneous recovery of initial system mass, which makes the aggregation process converges to good approximation results of the target value. In the next chapter, i.e., Chapter6, a continuous epidemic protocol is proposed with a novel adaptive restart mechanism to detect and adapt the system according to the churn impact.

5.2 Phases of Epidemic Data Aggregation Process

The epidemic data aggregation is a process to compute a synopsis for the set of distributed initial data values and provides global information about the system [26, 30]. The computation of aggregate functions such as *average*, *sum*, *min*, *max*, etc. is distributed over system nodes, and through randomised communications, nodes exchange their local values. Computations and communications follow a designated protocol to achieve a specific global aggregation task. Results of the distributed aggregation are locally available at each node, and due to the iterative reduction in the variance among local estimates, they eventually converge to the target value with an error as small as desired. Section 2.1 describes popular data aggregation protocols.

In this section, the process of epidemic data aggregation is investigated, and three distinct phases of the process are identified. Also, the phase that is critical for producing accurate results in the presence of node churn is further studied, resulting in a novel mechanism to address the violation of mass conservation invariant. Without loss of generality, the problem of estimating system size is chosen for the investigations, in which the distributed computation corresponds to the global summation function '*count*' [26, 30]. The process for the aggregate *count* is initialised for the peak data distribution resulting from the diffusion of the single weight value over the system. In consequence, the duration needed to achieve convergence to the size is relatively longer than other data aggregation functions (e.g. *average*). The longer the aggregation process requires achieving convergence, the more errors it gets due to the system dynamics. Also, the protocol SSEP is used in the analysis and verification experiments. SSEP is described in Section 4.3.

The fundamental characteristic of the epidemic data aggregation is the ultimate convergence to the desired target. As explained in Section 3.2, the target value $\mathcal{V}$ is a numeric value that usually corresponds to the initial set of distributed data values in the system and the required synopsis function. The formation of convergence is typically spanned from the start of the diffusion process of initial system values to the moment of which each node estimates the target value. At this moment, each node can detect the local convergence using one of the detection methods described in Section 3.3.

The summation function '*count*' requires a particular initialisation setting: the value $v_{i,t0}$ is set to the count unit, i.e. $v_{i,t0} = 1$, and the weight is set to $w_{\hat{\imath},t0} = 1$ at a single node $\hat{\imath}$ whilst $w_{i,t0} = 0$, $0 < i \leq N, i \neq \hat{\imath}$, where $t0$ is the system starting time and the node $\hat{\imath}$ is an arbitrary node that can be selected randomly or determined by a leader election method [92]. We called the node $\hat{\imath}$ the (*seed node*) due to its purpose of holding the initial seed value for the peak distribution. During the aggregation process, the pair $(v_{i,t}, w_{i,t})$ at each node i is exchanged at each cycle, and so propagated and aggregated. Eventually, the aggregate of all pair values in the system is evenly distributed over all nodes. After some time tc where $tc > t0$, the local estimate $e_{i,tc}$ at each node i converges with a high probability to $\mathcal{V}$, which is the count of system nodes N with some local error $\varepsilon_{i,tc}$:

$$e_{i,tc} = \frac{v_{i,tc}}{w_{i,tc}} = N \pm \varepsilon_{i,tc} \tag{5.1}$$

On the other hand, the aggregates of the initial system mass $\mathcal{M}v, \mathcal{M}w$ are time-invariant. Considering the formulas in Equation 3.1, which are the typical aggregate of initial system states. Thus, the totals of the initial pair values in the aggregation of the *count* function always gives the following results:

$$\mathcal{M}v = \sum_{i=1}^{N} v_{i,t0} = N, \qquad \mathcal{M}w = \sum_{i=1}^{N} w_{i,t0} = 1$$

As a result of the diffusion and the aggregation of the initial values during the base process, the mass is distributed and equalised in all system nodes, see the Equation 3.2 on page 38. In the ideal system conditions and absence of churn, the variance among local estimates of nodes approximates the minimal tolerated error, and the system achieves the ultimate global state, which is the convergence. Also, the local estimate $e_{i,t}$ at each node i computes the target value N at the convergence time tc as follows:

$$e_{i,t} = \frac{\frac{\mathcal{M}v}{N}}{\frac{\mathcal{M}w}{N}} = \frac{\sum_{i=1}^{N} v_{i,t}}{\sum_{i=1}^{N} w_{i,t}} = \frac{N}{1} \pm \varepsilon_{i,t} \quad \forall t \geq tc \tag{5.2}$$

Evidently, when the mass invariant holds due to the ideal system conditions, the local estimate $e_{i,t}$ converges to the correct value N and the local estimation error $\varepsilon_{i,t}$ tends to a minimum value that can be tolerated [26]. The tolerance value ϵ is a verification threshold for the estimation error, and it is an application parameter. Also, it has been discussed earlier alongside the consecutive cycles threshold Υ in Section 3.3. In the section, detection methods typically detect the convergence by periodically computing the estimation error $\varepsilon_{i,t}$ in the local estimate $e_{i,t}$. The parameter$\varepsilon_{i,t}$ keep decreasing as a result of the iterative reduction in the variance among local estimate until it approaches the minimal value that can be tolerated, $\varepsilon_{i,t} \leq \epsilon$. However, the variance reduction operation continues as long as the pairwise communication lasts and hence, the detection criterion is constantly true after convergence.

The case differs for the convergence under dynamical system conditions and when node churn is present. Two issues need to be addressed here. First, the target value (i.e. system size) is a time variant function. Although the protocol could be restarted periodically to follow the changes in the system size [27], a single global target value must be defined in each aggregation process to detect the convergence. Second, the mass-conservation invariant does not hold and node churn affect the accuracy of the estimates and may even prevent the convergence. In the following, we show that partitioning of the aggregation process into phases helps to address these issues.

Considering the aggregation process for the global *count* function, the target is achieved with the process starting at $t0$ and initiated at the seed node $\hat{\imath}$, which holds the weight $w_{\hat{\imath},t0} = 1$. The seed node initiates the exponential diffusion process of the non-null weight. The computation at a generic node with the weight $w_{i,t0} = 0$ are ineffective to the aggregation process until the node joins the diffusion process of the non-null weight, $w_{i,t} > 0$, either by a PUSH or a PULL message. The convergence is achieved at time tc when each node i holds a similar fraction of the initial weight $w_{i,tc} = \frac{w_{\hat{\imath},t0}}{N} = \frac{1}{N}$. At the time $t \geq tc$, although the pairwise exchanging continues, the local estimate $e_{i,t}$ at every node i holds to approximately the same aggregation result due to the decreasing error and the equal distribution of the system mass among nodes.

In the context of the previous analysis, each node i can be in one of the following three phases during the aggregation process:

1. INITIAL phase, when $w_{i,t} = 0$ and $t \geq t0$,
2. PROPAGATION phase, when $w_{i,t} > 0$ and local convergence is not yet achieved, $t0 \leq t < tc$, and
3. CONVERGENCE, when local convergence is detected at tc, $t \geq tc$.

To analyse the impact of node churn on the accuracy of local estimates during each phase, let us consider node i in the INITIAL phase. At each time $t \geq t0$, the local estimate $e_{i,t}$ is not available due to the null weight value. Message exchanges with other nodes in the same phase do not contribute nor modify estimate values, either local or remote. A message exchange with a node in the PROPAGATION phase determines a local phase transition. Although the failure of a node i in the INITIAL phase does alter the system mass $\mathcal{M}v$, it has no impact over the convergence or the accuracy of the aggregation process because the more critical system mass $\mathcal{M}w$ is preserved. As a consequence, the unexpected departure of a node i in the INITIAL phase only affects the target value from N to $N-1$, as if the node i was never part of the system. There is no need to address churn in the INITIAL phase, apart from redefining the target value Np that refers to the number of nodes which have entered the PROPAGATION phase. The new target value Np can be defined as $Np = N - f_{init,t}$ $(t \leq t_p)$

where $f_{init,t}$ is the total number of nodes which departed the system whilst in the INITIAL phase and t_p is a moment of time when all existing nodes have entered the PROPAGATION phase.

Definition 5.2.1. Let f_{init} be the number of nodes departed before joining an aggregation process, N be the initial system size, $Np = N - f_{init}$ is the number of nodes in the aggregation process.

Likewise, node churn in the CONVERGENCE phase has no impact on the accuracy of the estimations concerning the redefined target value Np. This is due to the equal distribution of the initial mass $\mathcal{M}v, \mathcal{M}w$ among existing nodes. The pairwise exchanging and the computation at each node in the CONVERGENCE phase always produce the result that the system has converged to with a very small error. This can be proven as bellows, where for simplicity we assume ideal system conditions and that all nodes have entered the PROPAGATION phase, $Np = N$ and $f_{init,t} = 0$. Considering the formula 5.2 at start time $t0$, the target value would be:

$$\frac{\sum_{i=1}^{N} v_{i,t0}}{\sum_{i=1}^{N} w_{i,t0}} = N,$$

and hence,

$$\sum_{i=1}^{N} v_{i,t0} = N \sum_{i=1}^{N} w_{i,t0}, \tag{5.3}$$

and the local estimate of each node i in the CONVERGENCE phase is a good approximation of the target value with a tolerated error:

$$e_{i,t} = N \implies N = \frac{v_{i,t}}{w_{i,t}},$$

and so the following relation also holds:

$$v_{i,t} = N w_{i,t} \quad \forall t \geq tc \tag{5.4}$$

Assume a node j in the CONVERGENCE phase has failed at time $t \geq tc$. In this case, the system mass has deprived of the pair $(v_{j,t}, w_{j,t})$. Also, the target can be estimated as

$$e_{i,t} = \frac{\sum_{i=1}^{N} v_{i,t0} - v_{j,t}}{\sum_{i=1}^{N} w_{i,t0} - w_{j,t}},$$

and by using the formulas (5.3) and (5.4), it is possible to compute the following aggregation result:

$$e_{i,t} = \frac{\sum_{i=1}^{N} v_{i,t0} - v_{j,t}}{\sum_{i=1}^{N} w_{i,t0} - w_{j,t}} = \frac{N \sum_{i=1}^{N} w_{i,t0} - N w_{j,t}}{\sum_{i=1}^{N} w_{i,t0} - w_{j,t}} \quad \forall t \geq tc \tag{5.5}$$

The previous analysis proves that the departure of a node in the system after it has converged does not change the result of the aggregation process. Any node still in the system maintains an asymptotic convergence to the same target value, which corresponds to the number of nodes which have entered the PROPAGATION phase, Np. Thus, there is no need to address churn in the CONVERGENCE phase.

It is now apparent that the PROPAGATION phase is the critical phase concerning the convergence and the accuracy of the data aggregation process. Unexpected departure of nodes in the PROPAGATION phase need to be addressed explicitly. Any node i in the PROPAGATION phase has the weight $w_{i,t} > 0$ that is a portion of the total weight mass $\mathcal{M}w$. Also, $w_{i,t} \neq \frac{\mathcal{M}w}{Np}$ because node i did not achieve convergence yet. Nodes in this phase hold critical information needed for the correct convergence to the target value. At start time $t0$, the seed node $\hat{\imath}$ is the first node in the PROPAGATION phase and the most critical node. At a time $t > t0$, other nodes make the transition from the INITIAL phase to the PROPAGATION phase during the diffusion process. Additionally, the probability of losing a local weight value is very small at $t0$. However, it exponentially increases with the diffusion process. At the same time the portion of the weight mass held at each node in the PROPAGATION phase decreases over time. Thus, the impact of losing some weight values also decreases.

To demonstrate the impact of node churn on the performance of the data aggregation process, the protocol SSEP is simulated under ideal and dynamic conditions. Figure 5.1 illustrates the results of the simulations. In the left column, the protocol is simulated in the ideal conditions, and the right column shows the protocol performance under gradual node churn. Figure 5.1.a and 5.1.d shows the average of local estimates converging to the target value. Also, figures 5.1.b and 5.1.e are zoom-in of the figures 5.1.a and 5.1.d respectively. The estimation error is presented in the figures 5.1.b and 5.1.f.

Simulation results show the perfect and accurate convergence of all nodes under the ideal conditions. The system size N and the number of nodes that made a transition to the PROPAGATION phase Np, alongside the average of local estimates, converge to the correct target N. Also, local estimation error tends to a very small value at all nodes and continue decreasing despite the achieving of convergence. Under the node churn, system size N changes every cycle and the number of nodes in the PROPAGATION phase is less than the initial system size, $Np < N$, which implies some nodes have departed the system during the INITIAL phase. Moreover, the average of local estimates shows the convergence of nodes, but to an incorrect value that is neither N nor Np. The local estimation error also shows a higher value indicating the existence of errors in all nodes. In general, the data aggregation process can converge under moderate dynamic conditions; but, it converges to wrong results at all nodes. Additionally, although some detection methods can detect local convergence, they cannot specify whether results have achieved the correct target or not.

The loss of mass due to node churn causes an estimation error during the aggregation process. To maintain a robust and correct convergence in the presence of churn, the local state $(v_{i,t}, w_{i,t})$ in each node i needs to be protected while the node is in the PROPAGATION phase, $t0 \leq t < tc$. The next section describes the PUSH-RELEASE model and its implementation, the Robust Epidemic Aggrgation Protocol (REAP) that adopts an embedded failure detection and immediate mass restoration mechanisms to achieve robustness and accuracy.

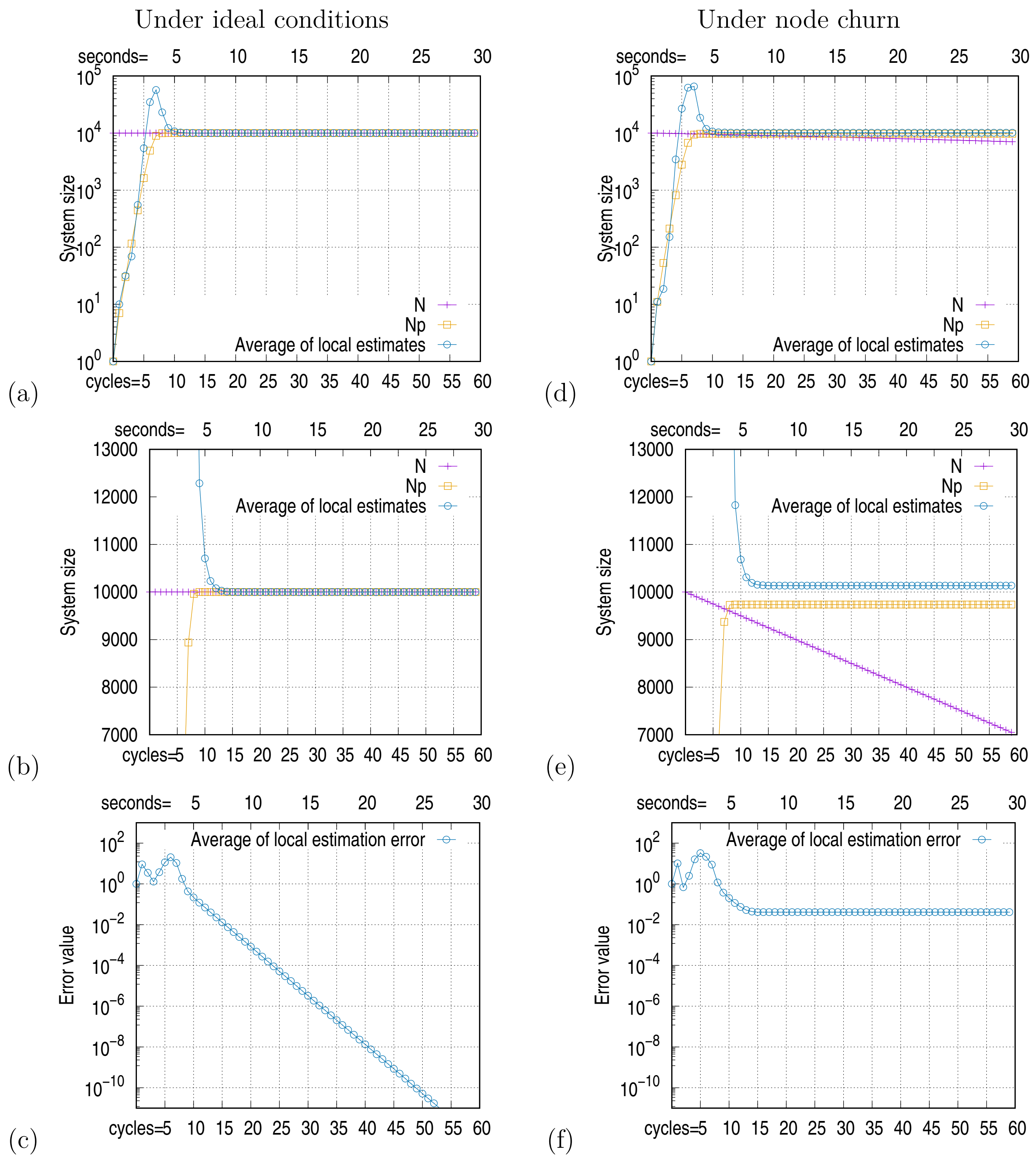


Figure 5.1: Performance of SSEP under ideal and dynamic system conditions, $N = 10^4$, $\ddot{\mathfrak{T}} = 500ms$; On the right, the protocol encounters gradual churn rate: a node fails at every second.

5.3 Push-Release Model

Dynamic system conditions such as node churn and network failure cause a direct violation of the system mass invariant, which needs to be conserved during the epidemic task to achieve the intended target. The data aggregation process is a typical example of major epidemic tasks and the common building unit for many epidemic services and applications. In particular, the aggregate *count* is the most susceptible process to the dynamical conditions, since the seed node holding the initial peak value. In the previous section, three phases of the aggregation process are identified, the INITIAL, PROPAGATION, and the CONVERGENCE phase. It is also verified in the section that the PROPAGATION phase is the most critical phase for the accuracy of the aggregation process.

Practically, existing nodes in a system join or participate in a particular aggregation task promptly after receiving a request or according to a pre-defined instructions. Either case, not all nodes start at the same time, but they have been presumed part of the system. Moreover, although they are included in the system, and they are aggregated into the initial mass that needs to be preserved, they are not effective to the aggregation process until they join the PROPAGATION phase. Nodes in the PROPAGATION phase hold critical information needed for the correct convergence to the target value, and this information needs to be protected. Also, after achieving convergence, node churn has no impact on the aggregation results, and hence, protecting nodes states is not necessary.

Replication is the fundamental technique to achieve fault-tolerance and data availability in distributed systems. Through sharing information and redundant resources, a distributed service ensures data consistency, reliability, and accessibility. However, replication protocols in distributed systems also have restraints, such as the central points of failure and the limited scalability. However, many epidemic solutions are used as alternative approaches [23, 39]. In general, an effective distributed replication service requires a reliable failure detector and coordination protocol, which are complicated tasks for a process that intended to be a simple component such as the data aggregation process.

As an example of using failure detection to achieve robust aggregation, the work in [75] presented a robust aggregation protocol that implements a failure detector to recover from the damage to the system mass. However, the failure detector is based on symmetric values exchanging and correlation among neighbour nodes. Also, the correct detection of nodes failure takes a certain amount of time in addition to a noticeable delay that is needed for local estimates to re-converge to a correct value. An other example is the work in [68], which presented a failure detector that uses epidemic data aggregation to achieve the consensus on the failure information. Although the failure detection protocol has reduced the time to the consensus, it takes longer time than the needed for a data aggregation to achieve convergence. From another perspective, several studies proposed supporting the accuracy of the data aggregation through maintaining robust topology manger protocols [13, 33, 76]. Although those attempts have build membership and peer-sampling services that are robust in one extent or another, the accuracy of the aggregate results is not ideal, in addition to time needed for the topology maintenance. Furthermore, modern services require lightweight and resilient solutions for data aggregation and topology management, e.g. OSN and IoT applications.

The following introduces the *Push-Release* model, which achieves robust data aggregation in the presence of node churn and network failures. The model implements a decentralised replication and failure detection mechanism that is compatible with the epidemic data aggregation process. It also provides instantaneous recovery and mass restoration. The model mainly addresses nodes in the PROPAGATION phase and apply the new mechanism to protect the data, which is critical for the aggregation process to achieve correct results. The model applies the symmetric-push sum exchange model and sends a PUSH message to random peer at every cycle. Also, it detects the local state, which is critical to the aggregation process. In consequence, each node j receives a remote state with a critical data flag from node i, creates a replica of the remote state and stores it in a local cache. In the followed cycle, node i sends a RELEASE message to node j to delete the local state of node i received in the previous cycle. However, when node j does not receive the RELEASE message for a predefined timeout, it restores the state into the system using the local aggregation pair. Through the RELEASE message, nodes can detect the failure of other nodes, and they can restore the missing mass using the replicas in the local cache. Additionally, replicas in the local cache of each node are either deleted in the next cycle or remain only for a few cycles until they are restored, which prevent overflowing the cache and keeps memory usage minimum. The sequence of messages in the *Push-Release* model is described in Figure 5.2.

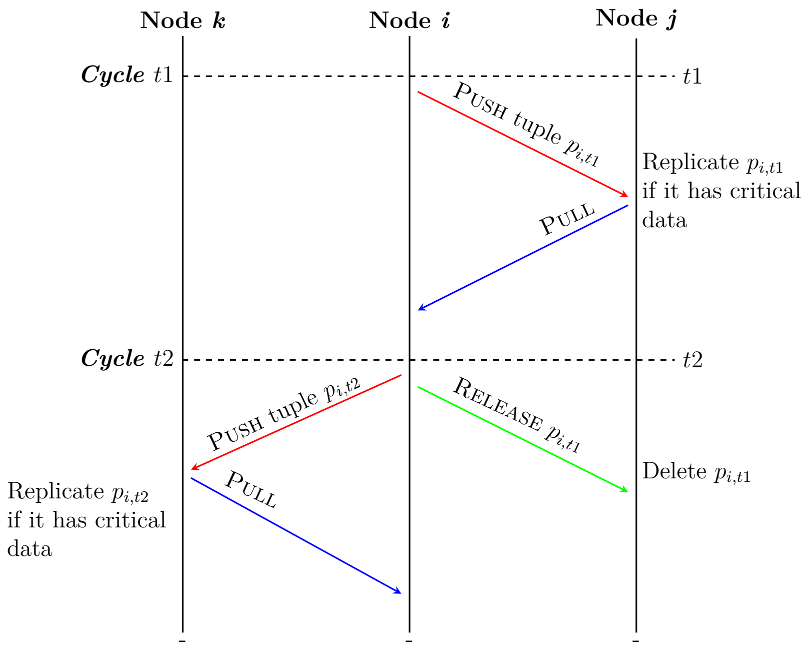


Figure 5.2: Messages exchange in *Push-Release* Model

In addition to inheriting all the intrinsic features of the epidemic aggregation process, the model supports the process by the detection of nodes failure and the prompt recovery from the damage due to the loss in system mass. Failure detection and mass restoration mechanism is performed along the line with the regular decentralised computations, and hence, it does not require any complicated external services and does not produce significant overhead, except for the RELEASE message which is generated by nodes in the PROPAGATION phase. The model is called '*Push-Release*', because it generates two symmetric messages in two consecutive cycles, i.e. the PUSH and the RELEASE messages. The next describes the Robust Epidemic Aggregation Protocol REAP, which implements the model.

5.3.1 Robust Epidemic Aggregation Protocol (REAP)

REAP is a novel epidemic protocol to achieved robust data aggregation in the presence of node churn and network failures. The protocol REAP detects nodes in the PROPAGATION phase to recognise the critical aggregation data and to flag the pairwise data exchange as follows. Each node i joins the PROPAGATION phase when it receives a non-null weight $w_{j,t} > 0$ from node j and exits the phase when node i detects the convergence locally. The local detection of convergence is accomplished using the heuristic detection method of Poonpakdee et al. [94], which is described in Section 3.3. Each node i maintains the queue $\mathcal{Q}_i$ and computes the local estimation error using the Coefficient of Variation formula, $\varepsilon_{i,t} = \frac{\mathcal{Q}_i.s}{\mathcal{Q}_i\bar{e}}$, where $\mathcal{Q}_i.s$ is the standard deviation and $\mathcal{Q}_i.\bar{e}$ is the arithmetic mean. The convergence is detected at each node i when $\varepsilon_{i,t}$ falls below the error threshold ϵ for a number of consecutive cycles Υ. The error threshold ϵ controls the desired level of aggregation accuracy and Υ is used to prevent precocious convergence detection during the aggregation process.

Also, each node i in the PROPAGATION phase sends two messages to the random node j, a PUSH message and a RELEASE message at two consecutive cycles. Assume the node i has sent a PUSH message to node j at cycle $t1$, so it sends the RELEASE message in cycle $t2$, Figure 5.2 illustrates messages exchange in REAP. The PUSH and RELEASE messages correspond to a cycle t should carry identical information, which helps REAP to handle the asynchronous exchange of messages where a RELEASE message may deliver before the prior PUSH message. REAP distinguishes the type of messages by the reception order. Typically, the RELEASE message specifies the sender node situation, node i determines j as failed when the RELEASE message from node j does not arrive before $\hat{\mathcal{T}}$, and $\hat{\mathcal{T}}$ is the maximum timeout value.

The protocol also maintains the cache $\mathcal{P}$ for the PUSH messages to manage the *Push-Release* operation. Each cycle the protocol inserts an entry $p_{i,t}$ of the current PUSH information in $\mathcal{P}$. Each entry $p_{i,t}$ contains the receiver node id, the current time t, and the local aggregation pair, $p_{i,t} = \langle \vec{\alpha}, t, v_{i,t}, w_{i,t} \rangle$. In the following cycle, the protocol sends the RELEASE message using the information in the entry $p_{i,t}$. After sending the RELEASE message, the entry $p_{i,t}$ is removed from $\mathcal{P}$, and so the cache size is kept minimum.

Nodes in the PROPAGATION phase assign a flag to each PUSH message indicating that the message has a critical information. On the reception of a PUSH message with a critical flag, the receiver node i updates its local aggregation pair, replicates the local pair, and stores the replica in the local cache $\mathcal{R}$ using the entry $r_{\alpha,t_\alpha} = \langle \alpha, t_\alpha, \hat{t}, v, w \rangle$, where α is sender node id,

t_α is the PUSH time and it is used to break ties when receiving a PUSH message from the same node i at different cycles, $\hat{t}$ is the maximum time to wait for the RELEASE message and (v, w) is the aggregation pair. In the subsequent cycle, node i receives a RELEASE message and deletes the corresponding entry in $\mathcal{R}$. In case the RELEASE message does not arrive for $\hat{\mathfrak{T}}$ cycles, the protocol assumes that sender node, say node k is the sender, has failed and the corresponding recovery entry is used in the mass restoration procedure. The replicated pair values in r_{k,t_k} are applied to the local pair $(v_{i,t}, w_{i,t}) = (v_{i,t} + r_{k,t_k}.v, w_{i,t} + r_{k,t_k}.w)$. The applied amount is then propagated and distributed among nodes through the regular message exchanging. However, the restored mass portion will encounter a small diffusion delay before it corrects local estimates in the existing nodes.

At the node j and after receiving a flagged PUSH message from node i, node j creates the recovery entry $r_{i,t_i} = \langle i, t_i, \hat{\mathfrak{T}}, v_{j,t}, w_{j,t} \rangle$. The tuple r_{i,t_i} includes a replica for the local pair values of the node j, which is replicated after sending the PULL message to node i and after updating the pair. Therefore, the replica has the latest state and the one which node i will achieve after receiving the PULL message from node j. Also, in case the node i has failed before it receives the PULL message, the sum of the lost values includes the aggregation pair at node i after the PUSH message and the pair at node j before the update, which equals the replica pair values after the update at node j.

As shown in Algorithm 10, at each cycle, the protocol detects the PROPAGATION phase and sets the PUSH flag in line 2. In lines 3-5, the protocol performs the PUSH operation. The protocol sends the RELEASE message in lines 6-8 and inserts a new entry in $\mathcal{P}$ (lines 9-10). The detection of node failure and the mass restoration procedure are performed in lines 11-15. The protocol specifies the PUSH and RELEASE messages in line 17 and line 18. The PULL message is generated in lines 22 and 23. The protocol inserts the local and the received estimates in the queue $\mathcal{Q}$ in the line 24. The critical state is replicated, and the entry r is stored in $\mathcal{R}$ (line 27). The local convergence is detected in line 28.

5.3.2 Experimental Results for REAP

The protocol REAP is simulated using the event-driven engine in PEERSIM. Two events are used in the simulation: (*i*) The RUN EVENT is scheduled at every cycle, and the event stops after a predefined number of cycles. At this event, a node sends PUSH and RELEASE messages, detects node failure and performs the mass restoration. (*ii*) The MESSAGE EVENT occurs when a node receives a message. At this event, the incoming message is processed, the local aggregation pair is updated, and the detection of convergence is performed.

Two protocols are examined in the simulations: SSEP and REAP. The performance and the accuracy of the protocols are monitored and recorded at each cycle using a dedicated external observer. The communication latency is adjusted for all messages to be delivered in the same global cycle. This adjustment is necessary to avoid the transition of some system mass $\mathcal{M}v, \mathcal{M}w$ across cycles during the aggregation process. The adjustment is applied only for the sake of the simulations and can be relaxed in the realistic contexts. In each simulation

Algorithm 10: Robust Epidemic Aggregation Protocol (REAP)

Require: a peer-sampling service, e.g. NCP$^+$; tolerance thresholds ϵ; cycles threshold Υ; queue length $l^{\mathcal{Q}}$; and maximum timeout value $\hat{\mathfrak{T}}$.

Initialisation: each node i has: $v = 1$, $w = 0$, except one seed node has $w = 1$; a flag for critical data $\rho = false$; a flag for local convergence $\hat{\rho} = false$; a cache for Push entries $\mathcal{P} = \{p = \langle \vec{\alpha}, t, v, w \rangle, ...\}$, where $\vec{\alpha}$ is peer's id, t is current time, and (v, w) are aggregation pair; a recovery cache $\mathcal{R} = \{r = \langle \alpha, t_\alpha, \hat{t}, v, w \rangle, ...\}$, where α is sender's id and t_α is the Push time, $\hat{t}$ is Release receiving timeout, and (v, w) are replica pair; $\mathcal{Q} = \emptyset$.

```
1  At each cycle $t$ at node $i$:
2  $\rho \longleftarrow (w > 0 \wedge \neg\hat{\rho})$                         // detect PROPAGATION phase
3  $j \longleftarrow getRandomPeer()$
4  $v = \frac{v}{2}$,    $w = \frac{w}{2}$
5  send $\langle v, w, t, \rho, reply = true \rangle$ to $j$              // a PUSH message to node $j$
6  foreach $p \in \mathcal{P}$ do                  // send RELEASE messages for entires in $\mathcal{P}$
7      send $\langle p.v, p.w, p.t, \rho = true, reply = true \rangle$ to $p.\vec{\alpha}$
8      $\mathcal{P} \longleftarrow \mathcal{P} - \{p\}$
9  if $\rho$ then                          // this PUSH is critical, create an entry in $\mathcal{P}$
10     $\mathcal{P} \longleftarrow \mathcal{P} \cup \{p = \langle j, t, v, w \rangle\}$
11 foreach $r \in \mathcal{R}$ do                  // failure detection and mass restoration
12     $r.\hat{t} \longleftarrow r.\hat{t} - 1$
13     if $r.\hat{t} == 0$ then                                  // timeout, restore mass
14         $v = v + r.v$,    $w = w + r.w$
15         $\mathcal{R} \longleftarrow \mathcal{R} - \{r\}$

16 At event 'receive message $m$ from $j$' at node $i$:
17 if $m.reply$ then                                           // $m$ is a PUSH message
18     if $\exists r \in \mathcal{R}$ where $r.\alpha == j$ and $r.t_\alpha == m.t$ then      // $m$ is a RELEASE message
19         $\mathcal{R} \longleftarrow \mathcal{R} - \{r\}$                        // delete the recovery entry $r$
20         return                              // end the procedure and do not update
21     else
22         $v = \frac{v}{2}$,    $w = \frac{w}{2}$                            // a PULL message to node $j$
23         send $\langle v, w, \rho = false, reply = false \rangle$ to $j$
24 $\mathcal{Q} \longleftarrow \mathcal{Q} \cup \{\frac{v}{w}, \frac{m.v}{m.w}\}$
25 $v = v + m.v$,    $w = w + m.w$                       // update local aggregation pair
26 if $m.\rho$ then                                        // $m$ has a critical data
27     $\mathcal{R} \longleftarrow \mathcal{R} \cup \{r = \langle j, m.t, \hat{\mathfrak{T}}, v, w \rangle\}$
28 $\hat{\rho} \longleftarrow (\frac{\mathcal{Q}.s}{\mathcal{Q}.\bar{e}} \leq \epsilon$ for $\Upsilon$ cycles)                  // detect local convergence
```

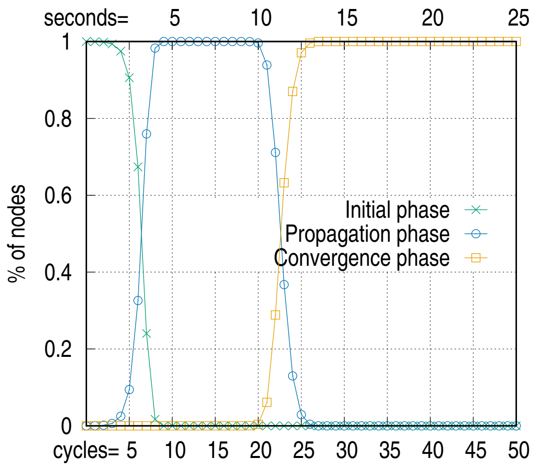
run, a different random seed is applied to enforce randomisation, and each experiment is repeated tens of times to validate the setting and the results. The protocols are initialised by a peak data distribution whereas $v_{i,t0} = 1$, $0 < i \leq N$; $w_{0,t0} = 1$ at seed node 0 and $w_{i,t0} = 0$, $1 < i \leq N$. Global parameters are adjusted to certain values as described in [52, 92]: the timeout value is set to $\hat{\mathfrak{T}} = 3$ cycles, the tolerance threshold is set to $\epsilon = 1\%$, the minimum number of consecutive cycles is set to $\Upsilon = 5$ cycles, the length of the $\mathcal{Q}$ is set to 10 elements. On another hand, the protocol NCP$^+$ is the peer-sampling service, and it is configured to maintain a random k-regular overlay with $k = 30$ and link expiry value $\check{\mathfrak{T}} = 10$.

The performance of REAP is examined in static network conditions. Figure 5.3.a shows the smooth transition of nodes among phases and the correct computation of the entry and exit points of the Propagation phase. Also, Figure 5.3.b presents the percentage of nodes that have locally detected the convergence. Nodes detect convergence after they make a transition to the Convergence phase. In Figure 5.3.c the local estimate value of a randomly chosen node is illustrated for SSEP and REAP showing the performance similarity when churn is absent. Figure 5.3.d shows the communication overhead of both protocols, and REAP produces only one new message for a small period during the Propagation phase.

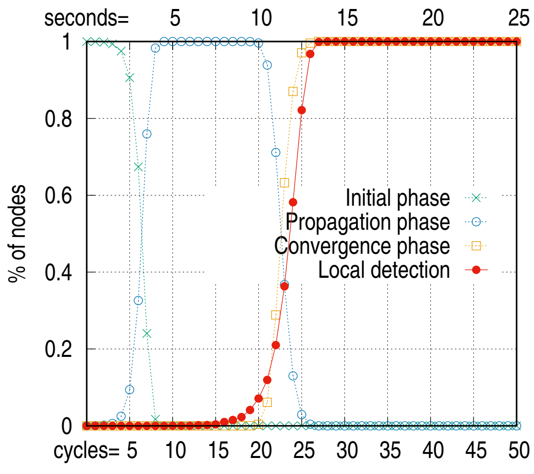
To evaluate the protocol REAP under the dynamical conditions, the protocol is simulated against SSEP in the presence of node churn with the following rates: {0%, 1%, 5%, 10%, 20%} of the system size N. Nodes are selected randomly at each cycle from cycles $[0, 30]$ and removed from the system by an external actor. The number of removed nodes at each cycle is a distribution of the total number of failures over the 30 cycles. This ensures the occurrence of node churn in different phases during the aggregation process. Also, the external observer detects the changes in the system size N and counts the number of failed nodes $f_{init,t}$ in the Initial phase. It then computes the correct size of nodes participating in the aggregation process Np, where $Np = N - f_{init,t}\ \forall t \leq t_p$, and t_p is the time when all existing nodes have joined the Propagation phase. Afterwards, the local estimation error is computed at each cycle t using $\varepsilon_{i,t} = \frac{|e_{i,t} - Np|}{Np}$. The results are illustrated in Figure 5.4 and Figure 5.5.

Figure 5.4 shows the recorded totals of the system mass $\mathcal{M}v, \mathcal{M}w$ during the aggregation process. The figure shows the correction in the mass amount in comparison to SSEP as a result of the mass restoration mechanism in REAP. The improvement of accuracy in REAP is shown in Figure 5.5. The figure summarises the results of ten experiments on each of the given churn rates. In the first case, no churn is present, and estimation error tends to a minimal value indicating the correct convergence to the target value. In the rates less than 10%, the estimation error rises to 1%, while the error exceeds 1% for the higher rates. Generally, estimation error in REAP is lower than the SSEP, which validates REAP ability to improve the accuracy of the epidemic data aggregation and to provide better results. However, some error still present in the aggregation results of REAP.

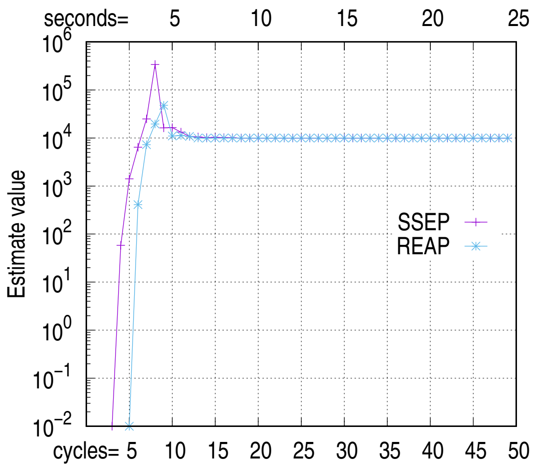
In summary, the partitioning of the data aggregation process into phases has helped understand the process and specified the most critical phase for the accuracy of the process results. Also, the protocol REAP is introduced to address the vital phase of the data



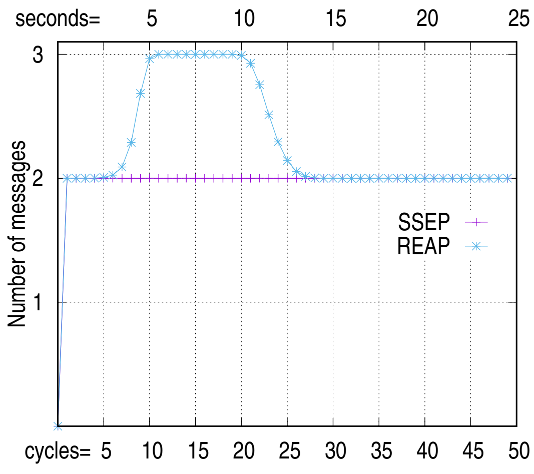
(a) Percentage of nodes in each phase.



(b) Local convergence detection.

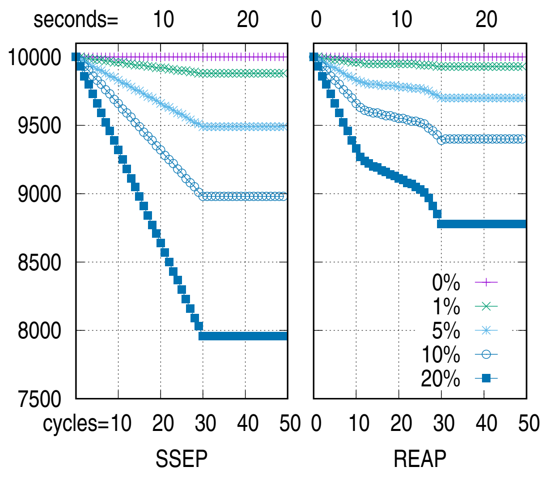


(c) Local estimate in a single node.

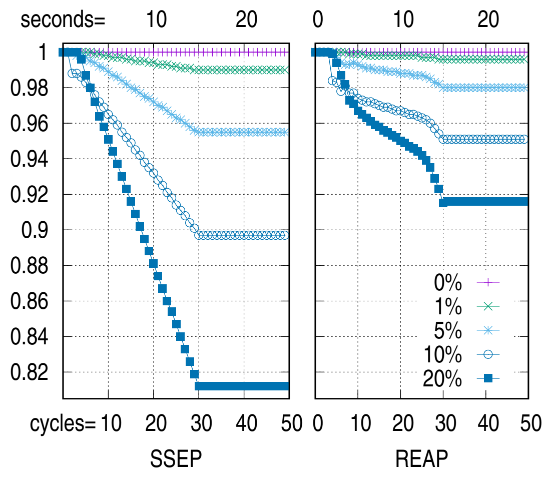


(d) Average number of messages per node.

Figure 5.3: Size estimation in REAP, $N = 10^4$, $\epsilon = 1\%$, $\Upsilon = 5$, $l^{\Omega} = 10$, $\ddot{\mathfrak{T}} = 500ms$.



(a) Total system mass $\mathcal{M}v$.



(b) Total system mass $\mathcal{M}w$.

Figure 5.4: System mass in REAP under various churn rates, $N = 10^4$, $\epsilon = 1\%$, $\Upsilon = 5$, $l^{\Omega} = 10$, $\ddot{\mathfrak{T}} = 500ms$.

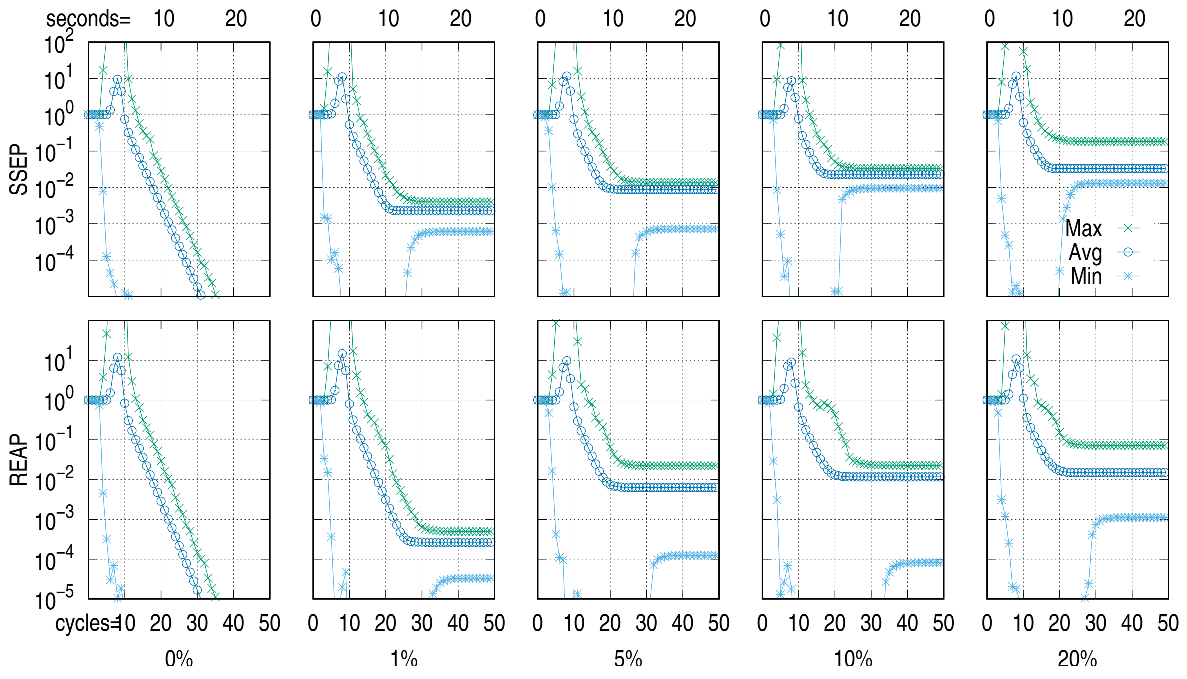


Figure 5.5: Estimation error in REAP under various churn rates, $N = 10^4$, $\epsilon = 1\%$, $\Upsilon = 5$, $l^{\Omega} = 10$, $\ddot{\mathfrak{T}} = 500ms$.

aggregation process and achieve robust results in the presence of node churn and network failure. REAP is a robust epidemic data aggregation protocol that performs basic aggregate functions similar to other protocols such as SSEP and SPSP. However, it has been verified that REAP can achieve better accuracy under moderate churn.

Although REAP is robust, the protocol has shown degraded efficiency when examined under high churn rates, which in some cases results accuracy was similar or even worse than the existing aggregation protocols, e.g SSEP. Thus, further investigation on the performance of the protocol REAP may improve the efficiency of the protocol and the data aggregation in general. The next section describes additional work that has been carried out on REAP and introduces the protocol REAP$^+$, the enhanced version of REAP.

5.4 Pull-Release Model

The previous section introduced the protocol REAP, which is a robust aggregation protocol that recognises the phases of the aggregation process which have been identified in Section 5.2. REAP is able to achieve good accuracy under moderate node churn; however, the produced accuracy under higher rates of churn was close to other aggregation protocols. This section investigates the efficiency and performance of REAP, targeting further improvement to the epidemic data aggregation process.

The protocol REAP has been simulated alongside the protocol SSEP to assess the performance and capture behaviour pitfalls and trends. The aggregation error (i.e. estimation error) produced by REAP was lower than SSEP when low and moderate node churn rates are used in the simulations. Despite that, REAP was not able to eliminate the error in the local estimates and achieve the ideal robustness. Technically, the protocol has achieved convergence to a result that has approximately 10% error of the target, and the target was the size of a dynamic system. Some services may tolerate 10% error in the aggregation results, for example, a coordination service that uses the protocol ECP may adjust the decision-making process to tolerate losing some participating nodes, and achieve the consensus of the majority rather than the full agreement. However, there are applications which demand higher accuracy. Furthermore, the moderate node churn (about 30%) is expected in extreme-scale systems, and it would be a great advantage when REAP achieve robust aggregation under the unexpected levels of churn. Table 5.1 shows expected churn rates in several real-world networks.

Typically, there are circumstances when churn and failures become very high and severe, e.g. a large system where 60% to 90% of nodes fail in less than one minute. More catastrophic scenarios are also possible, on which the overlay topology is disconnected, and the system network is partitioned [16]. Those cases required different solutions, e.g. [33], and they are not addressed in the present study. Under very high churn, the protocol REAP and SSEP exhibits approximately similar behaviour, and in some occasions REAP has achieved results with aggregation error higher than SSEP. This undesirable performance has motivated the investigation to identify sources of error which prevented REAP from achieving better accuracy, although it implements a failure detection and mass restoration mechanisms.

The investigation involved the development of additional simulation modules to capture more information about the aggregation process in REAP and SSEP protocols. Information such as nodes which have failed and their local states, recovery entries and the enclosed replicas, detected and undetected failures and more have been recorded and studied. The simulations also included examining the protocols under different churn rates and types, such as sudden and gradual churn with rates ranging from $30\% - 90\%$. Simulation cycles are further divided into sub-rounds, and nodes failure are distributed over the rounds to experience different scenarios, such as a node that fails before sending the Push message or before receiving the Pull message. Intensive experiments have been carried out, and many results have also been logged and analysed. The following list describes the investigation outcome, which determines the sources of aggregation error and proposes some solutions.

I) *Data Initialisation*:

In an epidemic system, each node i holds a data value x_i, $0 < i \leq N$, which presents a property, a sensor data or an internal parameter. Initially and before the aggregation process starts, the node i assigns a pair of aggregation values (v_i, w_i) for the process, v_i is set to the local data value, $v_i = x_i$, and the weight w_i is determined according to the required aggregation function [26, 30]. After the start of the aggregation process, nodes exchange their pair values and compute local estimates. However, due to the asynchronous message exchanging, interleaving messages are present at all times [33, 125], which resulting in asymmetric values at each node. In the protocol REAP, nodes in the Initial phase have a null-weight, but some nodes exchange their local data value v_i with nodes in the Propagation phase, and they leave the system before they enter the PROPAGATION phase. Nodes in the INITIAL phase should not contribute to the total mass of the aggregation because they are not part of the process.

Consider the size estimation process corresponding to the *count* aggregation function, in which $v_i = 1$ at all nodes and Np is the number of nodes in the PROPAGATION phase. Let $\mathcal{M}v_p$ be the sum of v_i values for all nodes in the PROPAGATION phase, $\mathcal{M}v_p = \sum_{i=1}^{Np} v_i$. Obviously, the statement $\mathcal{M}v_p = Np$ should be true. However, investigation results have shown that $\mathcal{M}v_p \neq Np$ due to sharing a portion of the mass with nodes in the INITIAL phase which have failed. Eventually, aggregation results will converge to an incorrect estimation of Np.

Data initialisation in the protocol REAP is a potential source of errors in the aggregation results. To limit errors from nodes in the INITIAL phase, all nodes should be initially assigned a null-values for the aggregation pair until they join the PROPAGATION phase, except for the seed node where the pair has to get the correct values. At the beginning, the local pair should be nulls, $(v_i, w_i) = (0, 0)$ at each node i, $0 < i \leq N$, except of $\hat{\imath}$ which has $(v_{\hat{\imath}}, w_{\hat{\imath}}) = (x_{\hat{\imath}}, w)$, where w is determined by the required aggregation function, e.g. $w = 1$ for the summation function. When a node i receives a non-null pair (v_j, w_j) from a remote node j and $w_j > 0$, node i enters the PROPAGATION phase and sets the local pair (v_i, w_i) as follows $(v_i, w_i) = (x_i + v_j, w_i + w_j)$. This initialisation mechanism eliminates errors from the exchange of messages with nodes in the INITIAL phase.

Another initialisation issue is the determination of the seed node. For instance, a seed node can be elected or randomly selected [32, 92]. However, this requires additional beforehand computation, and it is ultimately susceptible to a single-point failure. The requirement of a single-point for initiating the peak data distribution in epidemic tasks has been a challenge for a long time. In Section 6.1, we propose the decentralised seed selection mechanism to overcome this challenge.

II) *Detection method and error tolerance threshold:*

The detection method of local convergence has a vital role in the protocol REAP. In each cycle the protocol associates a boolean flag with the PUSH message to indicate that the message is holding a critical data from a node in the PROPAGATION phase. Also, the receiver node makes an entry for the critical data in the recovery cache whenever the PUSH message is flagged. The boolean flag is set by each node in the PROPAGATION phase and PUSH messages are not flagged when nodes make the transition to the CONVERGENCE phase. Nodes are required to detect the local convergence to make the transition, and the precise moment of convergence is determined by the detection method.

The convergence detection method in REAP is the heuristic method of Poonpakdee et al. [94], which is described in Section 3.3. The method computes the aggregation error ε using the coefficient of variance formula, which produces relative error values. Thus, the tolerance threshold ϵ is also specified in relative values, e.g. 10% or 0.01.

The verification of the threshold conceals some error which is neglected due to the threshold value, and it is usually a percentage of the target value. For example, 10% makes REAP to tolerate 10^4 nodes in a system of 10^6 nodes, which is a large amount of error for the size estimation task. Figure 5.3.b shows some nodes that had detected convergence before they truly converge to the target Np.

Mainly, nodes in the PROPAGATION phase may detect the local convergence earlier than required, which cause nodes to set the critical data flag off. Also, receiver nodes will stop replicating data and detecting failures before nodes achieve the correct convergence and thus, any node failure at this point increases the error in the aggregation results.

The detection method can achieve higher accuracy by either adjusting the relative threshold to higher precision or using the absolute value for the threshold setting as described in Section 3.3. For instance, the Standard Error (SE) formula can be utilised with a tolerance threshold set to the minimum desired error. Figure 3.1.d illustrates the convergence detection in SSEP when the heuristic method is used with SE formula and $\epsilon = 1$ node. Although the use of SE may delay the detection of convergence, it provides better accuracy and requests minimum information about the global system initialisation.

III) *After-push failures:*

In principle, all epidemic PUSH-PULL protocols including REAP select a random peer in each cycle and send a Push message to the peer. Typically, the sender node has no information on the state of the peer node and whether it is active or has failed. In this scenario, a node i randomly select a node j and sends a PUSH message to node j. If node j has failed before receiving the PUSH message, the protocol REAP has no way to recover the mass which is sent to node j. Fortunately, REAP is a PUSH-PULL protocol as well, and thus, node i can detect the failure of node j by monitoring the PULL message of node j. This mechanism can easily be implemented using the PUSH information and the recovery cache $\mathcal{R}$. After sending the PUSH message, node i replicates the local state in a recovery tuple $r_{i,t} = \langle j, t, \hat{t}, v_{i,t}, w_{i,t} \rangle$ and stores it in the cache $\mathcal{R}$, where j is the receiver node and t is current time at node i, i.e. the PUSH time. Later, the node i deletes the entry $r_{i,t}$ upon the receiving of the PULL message from node j. The mechanism allows the node i to recover the values sent in the PUSH message in case the PULL message is not received within the given timeout.

The absence of RELEASE messages is the initiator for the mass restoration procedure. In the real-world networks, RELEASE messages can have random delays and may deliver after the timeout, in which case node i has restored the values to the system. The setting of the timeout period is another source for the error, although it has a small probability. In another respect, the use of PULL messages to detect the failure of receiver nodes suggests that maybe sending the RELEASE message in the next cycle after receiving the PULL message makes more sense. However, this raises some issues too. For example, the Pull and Release messages will experience random delays, which might be in total higher than the timeout period, thus, causing a mass restoration at the receiver node. In general, the timeout value can be defined to minimise errors, and any Pull or Release messages that deliver after the timeout period should be ignored.

Additional source of the aggregation error in REAP is the non-flagged Push messages from a node in the Convergence or Initial phase. Assume a receiver node j is in the Propagation phase while the sender node i is not. The critical data flag controls the insertion of the recovery entries, which only considers the node i's data. The Pull message from the node j also carries critical data to the node i, which is not protected against node i failure. REAP was at the ignorance of this case because no entry is added to the recovery cache for this transaction and no Release message is expected.

IV) *Before-push failures:*

Investigations have shown that nodes failing before sending their next Push message is the major source of aggregation error in the protocol REAP. In the scenario, each node i sends a Push message at every cycle and when the Push is flagged, the receiver node j replicates the data and adds an entry in the recovery cache. In the same cycle, node i receives one or more Push messages from other nodes, updates its local state, and fails before sending a new Push message in the following cycle. Due to the failure of node i, the system has lost the updates in the local state of the node i. Also, the mass restoration procedure at node j will restore an out-of-date replica for the node i.

We called the error caused by the Before-push failure, the '*Compensation Error*' and it is denoted by $(\tilde{\varepsilon})$. The presence of the compensation error is unavoidable situation due to the asynchronous message exchanging and the interleaving messages. In essence and after the timeout period, node j restores the pair values in the replica $r_{i,t} = \langle i, t_i, 0, v, w \rangle$ to the system. However, the values which are lost due to the failure of node i are not equivalent. Also, the difference between the lost mass and its replica is usually higher when node i fails before it receives the Pull message from node j. The difference between the lost values and the restored replica values are calculated as follows:

$$\Delta v_i = v_{i,t} - r_{j,t}.v, \qquad \Delta w_i = w_{i,t} - r_{j,t}.w, \tag{5.6}$$

where i is the sender node that has failed and j is a receiver and mass restoration node, and Δv_i, Δw_i are the difference values. The total compensation error caused by all failed nodes during the aggregation process can be computed as follows:

$$\tilde{\varepsilon} = \frac{\left| \dfrac{\sum_{i=1}^{Np} \Delta v_i}{\sum_{i=1}^{Np} \Delta w_i} \right|}{Np}. \tag{5.7}$$

The leading cause of the compensation error is the updates, which the replica has missed. The replica needs to be updated whenever the local state of the original node is modified to overcome the formation of the compensation error, which is not trivial due to the required overhead for synchronising replicas. A mechanism for matching the replicas to their original values are needed. For instance, creating a new replica for each new update and deletes the previous one. Also, only one replica for each node is required to remain in the system. The matter is how to synchronise replicas in a decentralised manner for the highly distributed systems. The protocol REAP$^+$ provides a solution technique, and it is explained later in this section.

V) *Cascading failures:*

In very high dynamic conditions where nodes churn is severe, the failure of the sender node and the node which holds the replica is very likely to occur. In consequence, the system mass has deprived the values in the aggregation pairs in the failed nodes. Moreover, the mass restoration mechanism has failed as well. In essence, node i sends a PUSH message to node j and node i fails at time $t1$. Node j fails too at time $t2$, where $t2 \leq \hat{\mathfrak{T}}$ and $\hat{\mathfrak{T}}$ is the maximum timeout value. Also, $t1$ and $t2$ can be in the same cycle or $t2$ can be in a later cycle, $t2 \geq t1$. Thus, the aggregation error is caused by the presence of the *cascading failures*.

Cascading failures have the most detrimental effect on the performance of REAP due to the damage it causes to the mass restoration mechanism. During this project, there were attempts to solve this issue by creating more replicas. However, every replica requires additional messaging to release other replicas. Otherwise, the mass restoration will be inconsistent. Other techniques such as pinging and heart-beating are also investigated, and the techniques which have been suggested made the robust protocol very complicated and produces higher overhead. Also, the outcome was not feasible to achieve better performance and higher accuracy. It is an objective of this project to make robust protocols lightweight, rapid and resilient, hence, the restraint on using the pairwise mechanism for the failure detection and mass restoration in REAP and later in REAP$^+$.

The maximum timeout parameter of $\hat{\mathfrak{T}}$ can be used to reduce the effect of cascading failures. A shorter timeout value can limit the time in the face of churn, however, timeout value has to handle network latency and messaging delays as well. Generally, the protocol REAP has no mechanism to deal with the cascading failures when severe churn is present.

The previous list has described potential sources of aggregation error in the protocol REAP. Some sources can be overcome by making the correct adjustment, such as data initialisation and the threshold parameters of the convergence detection method. The Before-push and After-push failures can be handled by adopting new mechanisms and altering the protocol. However, the cascading failures are unavoidable in severe dynamic conditions, and the protocol is incapable of dealing with the resulting impact. The following explains the model *Pull-Release* that makes up-to-date replicas and reduces the effect of the compensation error.

The model *Pull-Release* adopts a different strategy on the creation and deletion of replicas, and in the same time, it preserves the intrinsic asynchronous exchanging and rapidity features of the *Push-Release* model. In principle, the creation of a replica follows the same procedure, a receiver node makes a replica after receiving a PUSH message and before sending the PULL message. However, in the model of *Pull-Release*, a sender node changes the remote replica and the host node of the replica whenever an update is received from another node, typically by a PUSH message. The new replica host is the node which has sent the latest PUSH message. Also, the new replica is for the aggregation values after the update and it is sent to the host node using the PULL message.

In the data aggregation, each update transaction includes two nodes: *sender* and *receiver*, and two messages: PUSH and PULL. The update transaction in the *Pull-Release* model makes three replicas as follows:

Prep is a local replica in the sender node for the values which are sent in the PUSH message to a random peer node. This replica will be deleted when the corresponding PULL message arrives. Otherwise, the sender node assumes that the peer node has failed and *Rrep* values are restored to the system after the given timeout.

Srep is a remote replica for the sender node in the receiver node. Upon the reception of the PUSH message, the receiver node updates the local aggregation values and makes a replica for them. This replica belongs to the sender node. The receiver node deletes this replica either by a RELEASE message or by the mass restoration procedure.

Rrep is a remote replica for the receiver node. Upon receiving the PULL message in the response to the previous PUSH message, the sender node deletes the local replica *Prep*, updates the local values, and insert new replica: *Rrep* for the node which has sent the PULL message, i.e. receiver node. *Rrep* is deleted by a RELEASE message or by the mass restoration procedure.

The replica *Prep* protect the mass against the failure of the receiver node and hence, handle the relevant issues of the After-push failures. Also, replicas *Srep* and *Rrep* are copies of the latest aggregation values after performing the local update, which minimises the compensation error that is caused by the Before-push failures. Moreover, the model changes the host of the replica *Srep* each time it receives a PUSH message, which means the replica is always up-to-date.

In the model, a RELEASE message is sent each time the replica *Srep* has a new host node, and the host is the node which sent the RELEASE message, and it sends the message only after receiving the PULL message. Figure 5.6 illustrates the message exchange in the model. A node that receiving a PULL message also receives information about the previous host node of the replica *Srep*. The new host uses this information to send the RELEASE message to the previous node. The RELEASE message is only sent after the PULL message to protect *Srep* from the failure of the new host node and the failure of the *Srep* owner node.

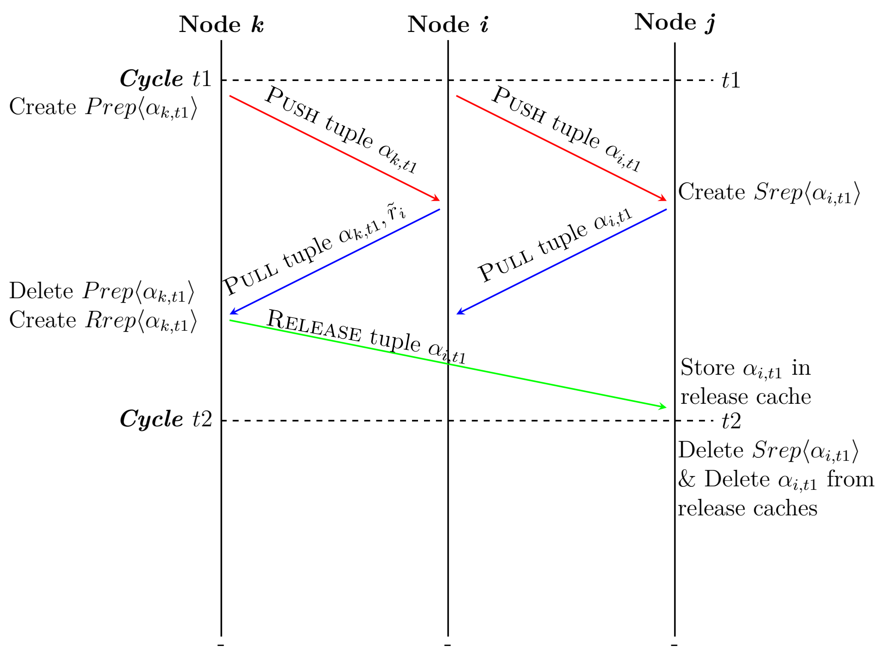

Node k
Node i
Node j
Cycle $t1$
$t1$
Create $Prep\langle\alpha_{k,t1}\rangle$
PUSH tuple $\alpha_{k,t1}$
PUSH tuple $\alpha_{i,t1}$
Create $Srep\langle\alpha_{i,t1}\rangle$
PULL tuple $\alpha_{k,t1}, \tilde{r}_i$
PULL tuple $\alpha_{i,t1}$
Delete $Prep\langle\alpha_{k,t1}\rangle$
Create $Rrep\langle\alpha_{k,t1}\rangle$
RELEASE tuple $\alpha_{i,t1}$
Store $\alpha_{i,t1}$ in release cache
Cycle $t2$
$t2$
Delete $Srep\langle\alpha_{i,t1}\rangle$ & Delete $\alpha_{i,t1}$ from release caches

Figure 5.6: Messages exchange in *Pull-Release* Model.

For the management of the replicas and to control the execution flow of the model, two new parameters are used. The replica identifier α and the replica tuple $\tilde{r}$. The parameter α is a globally unique identifier that defines each replica in the system at any time. A replica identifier can be a hash number, a random number, or any other form of universally unique identifiers. The function $F_\alpha(i, t)$ generates the unique identifier $\alpha_{i,t}$ that can be used globally from the inputs: node identifier and current time t. The replica identifier is used to process and recognise the replicas in the recovery cache. Any insert, update or remove operation requires the replica identifier to be accomplished.

The parameter $\tilde{r}$ is a local tuple at each node and holds information on the current replica and the node that host the replica. Each time the host node of the replica is changed, the tuple $\tilde{r}$ is updated to follow the new replica host. Also, the tuple is sent with the PULL message to the new host, which uses the information in the tuple to identify the old host and to send the RELEASE message. At start time, some PULL messages may have null replica tuples because receiver nodes are in the INITIAL phase and have no replicas in the system.

Each replica in the *Pull-Release* model is subject to certain operations during its lifetime in the system. Figure 5.6 shows the operations on the replicas. Starting by the replica *Prep* at node i, in which node i creates it for its current aggregation values with a unique identifier $\alpha_{i,t1} = F_\alpha(i, t1)$, and $t1$ is the current time. The node i stores the replica in the local recovery cache and sends the values to a random peer j in a PUSH message with the replica identifier $\alpha_{i,t1}$. Also, the local tuple $\tilde{r}_i$ is set to $\tilde{r}_i = \langle\alpha_{i,t1}, j\rangle$ to reference the replica *Srep* hosting node, i.e. node j. Upon the reception of the PUSH message, node j updates the local values and creates the replica *Srep* and use the parameter $\alpha_{i,t1}$ to identify the replica. In the PULL message, node j sends the aggregation values, the parameter $\alpha_{i,t1}$, and the tuple $\tilde{r}_j$. The node i recognises the parameter $\alpha_{i,t1}$ in the PULL message and deletes replica *Prep* with the identifier $\alpha_{i,t1}$. Then after, node i creates the replica *Rrep* for the local aggregation pair after the update and use $\alpha_{i,t1}$ to identify it. Also, node i sends a RELEASE message using the information in the tuple $\tilde{r}_j$. The replica identifier α and the tuple $\tilde{r}$ have a vital role in the management of replicas. However, the model controls the replicas and operations without using any extra messages or creating non-atomic operations. All replicas are created and updated asynchronously, and at each cycle, there is a single replica, i.e. replica *Srep*, for each node at a different host node. In case of the failure of any node, the mass restoration procedure restores the replica.

Due to asynchronous communications, sometimes node i receives a RELEASE message from another node, say h, before receiving the PULL message from node j, which implies that node j has changed the host of the replica *Srep* from i to h, however, node i does not have a replica *Srep* for node j. To preserve the consistency of the replicas, *Pull-Release* model stores the RELEASE message in a cache with a timeout period. Upon the receiving of the PULL message, the model deletes the replica *Prep* corresponding to the prior PUSH message, deletes the replica *Srep* with the identifier in the PULL message and deletes the RELEASE message in the cache.

In the simulations, we adopted simple implementation for function $F_\alpha(i,t)$ which generates unique integer values from the node identifier i and the current time t using the mathematical bitwise operations. The integer values are used as replica identifiers, where $\alpha_{i,t}$ defines each replica by the node which created it and sent it in a PUSH message, and by the time of creation. The time of creation is used to break the ties in case two replicas of the same owner exists at any node, which makes the replica identifier valid over the system at all times. Implementation of the function $F_\alpha(i,t)$ is as follows:

$$F_\alpha(i,t) = (t \ll 32 \vee i), \qquad F_\alpha \in \mathbb{Z}^+ \tag{5.8}$$

Although identifiers generated by the function F_α are limited to 64-bit integers, the generated identifiers preserve the natural order of the numbers, and it is globally unique for the lifetime of all replicas. The protocol REAP$^+$ is introduced next, and the protocol implements the *Pull-Release* model.

5.4.1 Robust Epidemic Aggregation Protocol-Plus (REAP$^+$)

The protocol REAP$^+$ is illustrated in Algorithm 11. The protocol implements the *Pull-Release* model to achieve robust data aggregation. REAP$^+$ adopts a new method for the data initialisation and the convergence detection. It also uses START EVENT to start the first cycle. The protocol requires access to a peer-sampling service, i.e. NCP, and requires information about the desired threshold values in advance.

Initially, the data values at each node i are held in the pair (x_{init}, w_{init}), where x_{init} is the data value that represent a local property, sensor data, or implicit decision, and the w_{init} is the aggregation determinant and it is set globally according to the required aggregation function [26, 30]. Also, the aggregation pair $(v_{i,t}, w_{i,t})$ are used in each node i and are initially set to nulls.

The protocol defines two caches and a tuple at each node: the recovery cache $\mathcal{R} = \{r = \langle \alpha, \hat{t}, v, w \rangle, ...\}$, which stores the replicas of type *Prep*, *Srep* and *Rrep*, and it is used by the failure detection and mass restoration procedure. The cache $\hat{\mathcal{R}} = \{\hat{r} = \langle \alpha, \hat{t} \rangle, ...\}$ is used for the temporary storage of RELEASE messages until receiving the corresponding PULL message or timeout. Also, the replica tuple $\tilde{r} = \langle \alpha, \vec{\alpha} \rangle$ is used in the protocol to refer to the current replica and the current host node. The parameter α is the replica identifier, and it is assigned a new universal value at each cycle, as shown in the procedure *PushAndReplcate*. Also, $\hat{t}$ is the timeout counter, and it is set to the maximum timeout value $\hat{\mathcal{T}}$ whenever an entry is inserted in the caches.

The protocol REAP$^+$ adopts the improved heuristic method for the detection of local convergence and defines the queue $\mathcal{Q}$ at each node. The method is described in Section 3.3. In REAP$^+$, the estimation error ε is computed using the Standard Error (SE), formula 3.8. By using the SE, the protocol avoids the potential residual errors that is tolerated due to the setting of thresholds in the detection method. The protocol detects the convergence at the start of each cycle (line 4, Algorithm 11), where all updates of previous cycles are applied and before sending the PUSH message which, in this case, holds the latest convergence state.

In REAP$^+$ the aggregation pair $(v_{i,t}, w_{i,t})$ is initially set to nulls. The protocol updates the pair values at each node when receiving a message from a node in the PROPAGATION phase. After update, the receiver node makes the transition to the PROPAGATION phase too. To ensure the correct flow of the exchange and update transaction, the protocol detects the first massage with a critical data flag and updates the pair $(v_{i,t}, w_{i,t})$ using the initial pair (x_{init}, w_{init}) as shown in lines 10-11 in Algorithm 11. The update at this point indicates that the node has joined the PROPAGATION phase, and it can share the local pair values with other nodes with a flag for the critical data.

The protocol calls the procedures *PushAndReplicate* at each cycle. In the procedure, a universal replica identifier $\alpha_{i,t}$ is generated and PUSH message is sent. Also, the procedure stores the replica *Prep* in the recovery cache $\mathcal{R}$ using the identifier $\alpha_{i,t}$ when the node is in the PROPAGATION phase. The procedure updates the tuple $\tilde{r}$ with the new host node and the identifier $\alpha_{i,t}$. Procedure *ReleaseReplica* is performed before the procedures *PushAndReplicate*. It releases the remote replica before setting the new one in *PushAndReplicate* and updates the recovery cache should be any RELEASE messages in the cache $\hat{\mathcal{R}}$. The procedure *MassRestoration* detects nodes failure and restores their mass. The mass is applied to the local pair values of the node. On the other hand, when a message is received (line 9 Algorithm 11), the protocol recognises the type of the message and performs a certain action for each type following the algorithm. Also, the protocol updates the local pair values, inserts the local and remote estimates in the queue $\mathcal{Q}$, and makes a recovery entry for *Srep* in the recovery cache $\mathcal{R}$.

Algorithm 11: Robust Epidemic Aggregation Protocol-Plus (REAP$^+$)

Require: a peer-sampling service, e.g. NCP$^+$; tolerance thresholds ϵ; cycles threshold Υ; queue length $l^{\mathcal{Q}}$; and maximum timeout value $\hat{\mathcal{T}}$.

Initialisation: each node i has: initial data pair (x_{init}, w_{init}); aggregation pair $(v = 0, w = 0)$; a flag for critical data $\rho = false$; a flag for local convergence $\hat{\rho} = false$; a recovery cache $\mathcal{R} = \{r = \langle \alpha, \hat{t}, v, w \rangle, ...\}$, where α is tuple id; $\hat{t}$ is timeout counter, and (v, w) is the replica pair; a cache for RELEASE messages $\hat{\mathcal{R}} = \{\hat{r} = \langle \alpha, \hat{t} \rangle, ...\}$; a tuple for the remote replica data $\tilde{r} = \langle \alpha, \vec{\alpha} \rangle$, where $\vec{\alpha}$ is the remote node, $\tilde{r} = \emptyset$; $\mathcal{Q} = \emptyset$.

```
1  At start time $t0$ at node $i$:
2  $PushAndReplicate()$                          // send and replicate PUSH message

3  At each cycle $t$ at node $i$:
4  $\hat{\rho} \longleftarrow (\frac{\mathcal{Q}.s}{\sqrt{l^{\mathcal{Q}}}} \leq \epsilon$ for $\Upsilon$ cycles)      // Detect local convergence
5  $\rho \longleftarrow (w > 0 \wedge \neg\hat{\rho})$                 // Detect the PROPAGATION phase
6  $ReleaseReplica()$                           // send RELEASE to the replica host
7  $PushAndReplicate()$                         // send and replicate PUSH message
8  $MassRestoration()$        // detect failed nodes and restore thier replica

9  At event 'receive message $m$ from $j$' at node $i$:
10 if $\neg\rho \wedge m.\rho$ then               // currently in INITIAL phase, assess $m$
11     $v = x_{init}$,    $w = w_{init}$          // make transition to PROPAGATION phase
12 if $m$ is PUSH then                            // $m$ is a PUSH message
13     $v = \frac{v}{2}$,    $w = \frac{w}{2}$                          // a PULL message to node $j$
14     send PULL= $\langle m.\alpha_{i,t}, v, w, \rho = (\rho \vee m.\rho), \tilde{r} \rangle$ to $j$
15     $\tilde{r} \longleftarrow \langle m.\alpha_{i,t}, j \rangle$        // update local tuple of the remote replica
16 if $m$ is PULL then                            // $m$ is a PULL message
17     $\mathcal{R} \longleftarrow \mathcal{R} - \{r\}$ where $r.\alpha_{i,t} == m.\alpha_{i,t}$      // delete the PUSH entry
18     if $m.\tilde{r} \neq \emptyset$ then                    // a RELEASE to node $m.\tilde{r}.\vec{\alpha}$
19         send RELEASE= $\langle m.\tilde{r}.\alpha_{i,t} \rangle$ to $m.\tilde{r}.\vec{\alpha}$
20 if $m$ is RELEASE then                         // $m$ is a RELEASE message
21     $\hat{\mathcal{R}} \longleftarrow \hat{\mathcal{R}} \cup \{\hat{r} = \langle m.\alpha_{i,t}, \hat{\mathcal{T}} \rangle\}$
22     return                                 // end the procedure and do not update
23 $\mathcal{Q} \longleftarrow \mathcal{Q} \cup \{\frac{v}{w}, \frac{m.v}{m.w}\}$
24 $v = v + m.v$,    $w = w + m.w$                  // update local aggregation pair
25 if $m.\rho$ then                               // $m$ has a critical data
26     $\mathcal{R} \longleftarrow \mathcal{R} \cup \{r = \langle m.\alpha_{i,t}, \hat{\mathcal{T}}, v, w \rangle\}$
```

```
procedure PushAndReplicate()
    $j \longleftarrow getRandomPeer()$
    $v = \frac{v}{2}, \quad w = \frac{w}{2}$
    $\alpha_{i,t} \longleftarrow F_\alpha(i,t)$                          // generate global tuple id
    send PUSH= $\langle \alpha_{i,t}, v, w, \rho \rangle$ to $j$              // a PUSH message to node $j$
    if $\rho$ then                                         // the PUSH has critical data
        $\tilde{r} = \langle \alpha_{i,t}, j \rangle$                        // record the remote replica
        $\mathcal{R} \longleftarrow \mathcal{R} \cup \{r = \langle \alpha_{i,t}, \hat{\mathcal{T}}, v, w \rangle\}$          // create entry in $\mathcal{R}$

procedure ReleaseReplica()
    if $\tilde{r} \neq \emptyset$ then                           // a RELEASE message to replica host
        send RELEASE= $\langle \tilde{r}.\alpha_{i,t} \rangle$ to $\tilde{r}.\vec{\alpha}$
        $\tilde{r} = \emptyset$
    foreach $\hat{r} \in \hat{\mathcal{R}}$ do                          // process RELEASE messages in $\hat{\mathcal{R}}$
        $\hat{r}.\hat{t} \longleftarrow \hat{r}.\hat{t} - 1$
        if $\exists r \in \mathcal{R}$ where $r.\alpha_{i,t} == \hat{r}.\alpha_{i,t}$ then
            $\mathcal{R} \longleftarrow \mathcal{R} - \{r\}$        // the PULL $\alpha_{i,t}$ is recieved, delete entries
            $\hat{\mathcal{R}} \longleftarrow \hat{\mathcal{R}} - \{\hat{r}\}$
        else
            if $r0.\hat{t} == 0$ then
                $\hat{\mathcal{R}} \longleftarrow \hat{\mathcal{R}} - \{\hat{r}\}$                    // timeout, delete entry

procedure MassRestoration()
    foreach $r \in \mathcal{R}$ do
        $r.\hat{t} \longleftarrow r.\hat{t} - 1$
        if $r.\hat{t} == 0$ then                           // timeout, restore mass
            $v = v + r.v, \quad w = w + r.w$
            $\mathcal{R} \longleftarrow \mathcal{R} - \{r\}$                               // delete entry
```

5.4.2 Experimental Results for REAP$^+$

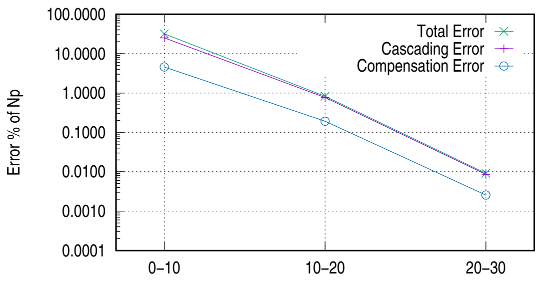
The protocol REAP$^+$ is simulated using the event-driven engine in PEERSIM. Three events are used in the simulation: (*i*) The START EVENT occurs only once at the start time of each node. At this event, nodes create the first replica identifier and send the first PUSH message. (*ii*) The RUN EVENT is scheduled at every cycle, and the event stops after a predefined number of cycles. At this event, a node detects the current phase of the aggregation process and sends PUSH and RELEASE messages, detects node failure and performs the mass restoration. (*iii*) The MESSAGE EVENT occurs when a node receives a message. At this event, the incoming message is processed, the local aggregation pair is updated, and the recovery replica insertion is performed.

Three protocols are examined in the simulations: SSEP, REAP, and REAP$^+$. The communication latency is adjusted for all messages to be delivered in the same global cycle. This adjustment is necessary to avoid the transition of some system mass $\mathcal{M}v, \mathcal{M}w$ across cycles during the aggregation process. In each simulation run, a different random seed is used, and each experiment is repeated 30 times to collect extensive results. The protocols are initialised by a peak data distribution whereas $x_{init} = 1$ at each node i; $w_{init} = 1$ at seed node 0 and $w_{init} = 0$ at all other nodes. Global parameters are adjusted as follows: the timeout value is set to $\hat{\mathfrak{T}} = 3$ cycles, the tolerance threshold is set to $\epsilon = 1\%$ and the minimum number of consecutive cycles is set to $\Upsilon = 5$ cycles for the protocols SSEP and REAP. The thresholds for the protocol REAP$^+$ are adjusted to $\epsilon = 1$ and $\Upsilon = 3$. All protocols have the length of the $\mathfrak{Q}$ set to 10 elements. On another hand, the protocol NCP$^+$ is the peer-sampling service, and it is configured to maintain a random k-regular overlay with $k = 30$ and link expiry value $\check{\mathfrak{T}} = 10$.

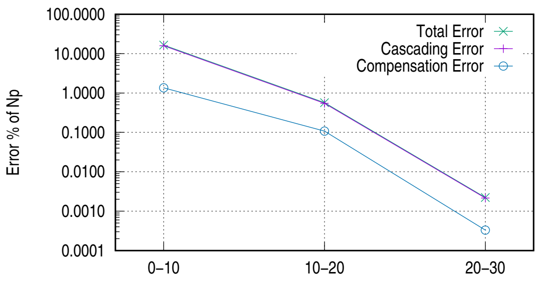
In the simulations, the protocols are examined under two different scenarios. The first scenario aims to investigate the impact of churn at different times during the aggregation process. As described in section 5.2, the probability of losing mass at the start of the aggregation process is small, but the impact can be higher. Due to the diffusion process and the distribution of system mass, the impact decreases as the aggregation process approaches convergence. In this scenario, sudden node churn is applied, where nodes suddenly fail at a particular period, and then the system continues steadily. The simulation time is divided into four intervals as follows $\{[0-10[, [10-20[, [20-30[, [30-60]\}$, the impact on each interval is examined separately. Experiments consist of enforcing 30% of system nodes to fail within the given interval.

Figure 5.7 illustrates the results of experiments in the first scenario. The results present the average aggregation error calculated from the outcome of 30 experiments for each interval. The results of the fourth interval give no useful information and have been omitted. In general, results have shown the expected behaviour and nodes churn has a higher impact on the early intervals. However, the impact is reduced in the later intervals. Also, Figure 5.7 presents three error indicators, the total error, the error caused by the cascading failures, and the compensation error. In REAP the error indicators are higher than the protocol REAP$^+$ in all intervals. The figure also shows that the cascading failure is leading the errors at all times. The compensation error has been seen in the results of the protocol REAP$^+$, although it is less than REAP. This implies that there are other sources of the compensation error rather than the *Before-push* failures, which has been dealt with in REAP$^+$. Mainly, a node may fail after receiving a PUSH message and before sending the PULL message. Although the node hosting the replica *Rrep* for the failed node will perform the mass restoration, the restored values may cause some compensation error.

In the second scenario, nodes churn was continuous during the simulations, from the start to the end of the simulation time. This is an important case because some aggregation protocols fail to converge under constant dynamic conditions. In experiments, three levels of churn are used, $\{30\%, 60\%, 90\%\}$, which range the churn from the moderate to severe. The results are statistical synopsis of 30 experiments under each level of churn. Figure 5.8 illustrates the recorded aggregation error in each level of churn. In general and as presented in the figure, the protocol REAP$^+$ outperforms the protocols SSEP and REAP under all levels of churn. However, REAP$^+$ takes more time to detect convergence due to the precise detection method and due to mass restoration. The figure also presented the numerical value of the recorded errors to give clear limits on the achieved accuracy.



(a) Errors of REAP



(b) Errors of REAP$^+$

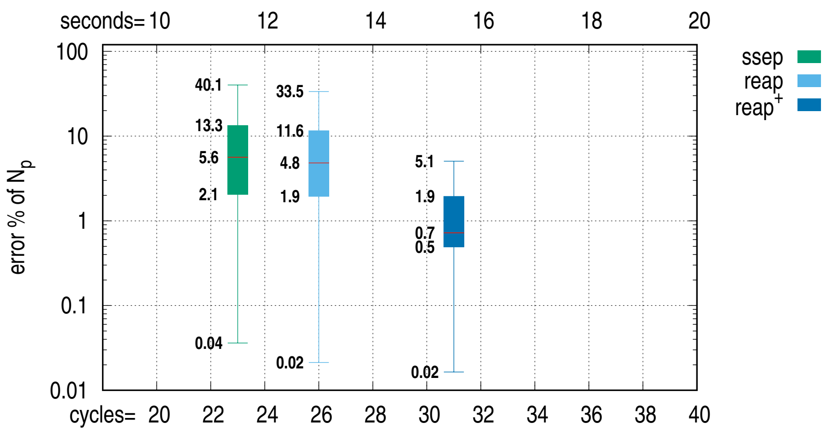
Figure 5.7: Aggregation error of REAP and REAP$^+$ under sudden node churn; $N = 10^4$, $l^\Omega = 10$, $k = 30$, $\ddot{\mho} = 500ms$, $\hat{\mho} = 3$; for REAP$^+$: $\Upsilon = 3$, $\epsilon = 1$; for REAP: $\Upsilon = 5$, $\epsilon = 1\%$.

In summary, this section has introduced the robust protocol REAP$^+$ which implements the model of *Pull-Release*. The protocol involved novel mechanisms to handle issues and problems in the previous version. The experimental results have shown that REAP$^+$ produces higher accuracy among other aggregation protocols, under the sudden and continuous dynamic conditions. However, higher accuracy of the protocol REAP$^+$ comes with a price of longer convergence time. The next section discusses the findings and outcomes.

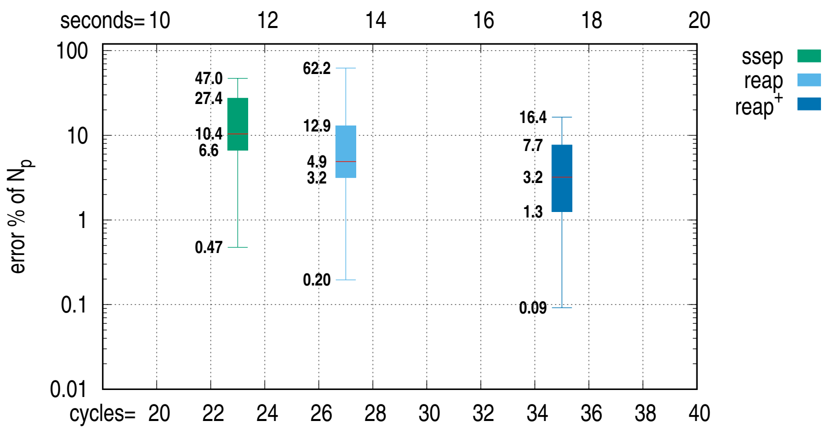
5.5 Discussions

In this chapter, the problem of distributed data aggregation under nodes churn and network failures have been studied. Typically, the data aggregation process is susceptible to damage from the churn of nodes and may produce incorrect results or may not produce results at all. The detrimental impact on the data aggregation process in the dynamic conditions is due to the violation of the mass-conservation invariant, which needs to be constant for the process to converge to the target value [26, 27]. The analysis of the data aggregation process presented in section 5.2 has led to the identification of three implicit phases in the process. Investigations have shown that the nodes churn have a different impact on each phase. Also, the phase which is critical for the aggregation process is identified and has received further study, leading to introducing new robust data aggregation protocols that recognise the implicit phases, and achieve higher accuracy in the presence of nodes churn.

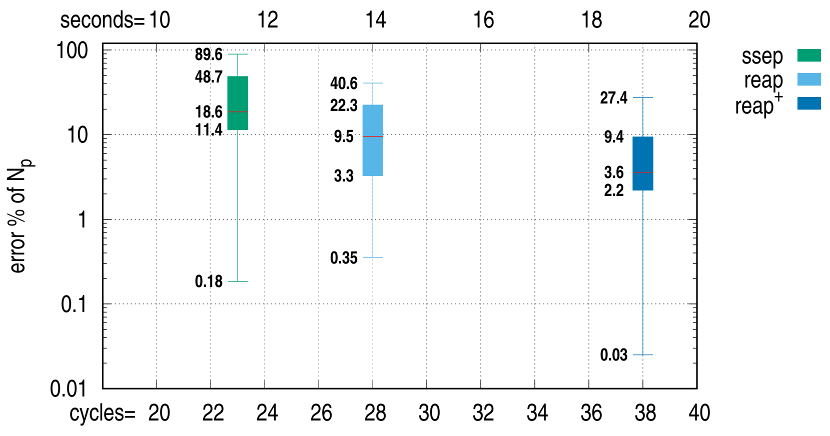
Two protocols are introduced in this chapter, REAP that implements the *Push-Release* model and REAP$^+$ which implements the *Pull-release* model. The protocols are robust epidemic data aggregation protocols that converge to a good approximation of the target value when churn is present. The protocols are scalable, resilient and can be used in combination with any peer-sampling service or membership protocol. They can be used to



(a) Churn rate 30%



(b) Churn rate 60%



(c) Churn rate 90%

Figure 5.8: Aggregation error of SSEP, REAP, REAP$^+$ under continuous churn; $N = 10^4$, $l^{\Omega} = 10$, $k = 30$, $\ddot{\Im} = 500ms$, $\hat{\Im} = 3$; for REAP$^+$: $\Upsilon = 3$, $\epsilon = 1$; for REAP and SSEP: $\Upsilon = 5$, $\epsilon = 1\%$.

build scalable decentralised services independent of the underlying topologies. Moreover, the protocols implement an innovative, distributed pairwise replication mechanism. In addition to the distributed failure detection and instantaneous mass restoration mechanisms. The following discusses the work presented in this chapter:

- The protocols REAP and REAP$^+$ have preserved the intrinsic time complexity of Push-Pull protocols, which is $O(\log(N))$ [26], and they can achieve convergence in the time $tc = (\log(N) + \log(\frac{1}{\epsilon}) + \Upsilon)$ under stable conditions. In dynamic conditions, some nodes may leave the system before and during the aggregation process. Let Np be the number of nodes that have joined the aggregation process and f is the number of nodes of Np that have departed during the aggregation proposes, the protocols can achieve convergence in time $tc = (\log(Np - f) + \log(\frac{1}{\epsilon}) + \Upsilon)$. The communication complexity in protocols REAP and REAP$^+$ increases by one message at the Propagation phase, i.e., 3 messages per node per cycle. After convergence, messages number retains equivalent to the Push-Pull protocols, i.e., 2 messages per node. At $t < tc$ the overall overhead is $O(3Np)$ and $O(2Np)$ otherwise.

- In the context of comparing the performance of the protocols REAP and REAP$^+$ to others related work, the research presented in this chapter has adopted the aggregation error (i.e. estimation error) as a standard metric to measure the accuracy of the protocols. Also, the work has introduced several precise statistics for the expected error under different levels of churn. In the literature, research studies presented the achieved accuracy in other metrics which sometimes do not provide the needed precision for comparative analysis. Mainly metrics such as convergence factor, relative error and coefficient of variance are used to describe the convergence accuracy. The work in [13] used 'Newscast', a dynamic topology manager to support epidemic data averaging under dynamic conditions. The results are shown to converge under 50% of sudden churn, however, no precise estimation error is provided. In [72], The convergence under different types of churn in several aggregation protocols is presented. Results showed that some protocols have achieved nearly convergent, and others have not converged, and the error in the convergent is not specified. About 8% error is monitored in the results of [36] for 50% churn in a structured *Chord* network. In [75], more than 1% error is observed under continuous churn of 25%. In general, the proposed protocols REAP and REAP$^+$ achieve competitive accuracy in comparison to other aggregation protocols.

- Although the achieved accuracy in REAP and REAP$^+$ is adequate for many epidemic services under churn, there is a residual error in the produced results, especially in high churn rates. Investigations have discovered unavoidable sources for the aggregation error, i.e., due to compensation mechanism and due to cascading failures, which impacted the restoration procedure accuracy. However, these sources formed small portion of the total error, and in particular, the protocol REAP$^+$ was able to handle other sources of error. It is a challenge to achieve ultimate accuracy in realistic conditions of epidemic systems due to the asynchrony and dynamism. This complies with the impossibilities of distributed systems that are described in Section 2.4.

- One vital achievement of the robust protocols REAP and REAP$^+$ is: they can detect convergence under all levels of churn. The robust protocols adopt asynchronous message exchanging and processing cycles, and because they can detect convergence, they can eventually and effectively terminate. This contribution satisfies the requirements for asynchronous distributed systems to achieve agreement and acquire consensus [44, 86, 89]. Although, epidemic protocols eventually converge in high probability in stable networks [27, 90], the work in this chapter has empirically proved that the robust epidemic protocols converge and detect convergence under dynamic conditions, and even under the worst churn scenarios.

  The protocols have achieved convergence in dynamical conditions through the adopted heuristic method for the detection of local convergence. The method recognises the state of convergence in aggregation results when the target value is not available. In stable systems, data aggregation protocols that used the heuristic method have detected the convergence to the target value with a desired error that is tolerated [92, 94]. In dynamic conditions, the method enabled the robust data aggregation protocols to detect the convergence on a good estimation of the target value, and the estimation accuracy depends on the level of the node churn.

  A typical example of the advantage of the robust data aggregation protocols is the epidemic protocol for consensus ECP described in section 4.5. ECP can use the robust data aggregation protocol in the agreement process and achieve consensus or coordination in the presence of node churn. However, the decision-making process in the agreement phases may need to tolerate the expected portion of the departing nodes, which implies that the achieved consensus is on the majority rather than complete.

- The accomplished survey presented in Section 5.1 aimed to adopt the typical churn model for studying the epidemic data aggregation process. During the survey, it was difficult to extract accurate churn rates for short time intervals due to the acquisition methods that are used in the research studies. However, and with general assumptions on node departure, estimation for expected departure rates were obtainable from the statistical distributions of the session duration provided in the studies. The survey has shown that in normal system conditions, 30% of nodes churn in the average should be expected. This result implies that churn rates are usually moderate, and data aggregation protocols are required to provide acceptable accuracy in this level of churn. Higher churn levels are also considered; however, the higher levels have a low probability of occurring. Additionally, it has limited global impact on decentralised systems which are deployed over a large-scale network.

  The churn model in this chapter has focused on the departure of nodes as the primary source of churn. Nodes joining the system also cause a detrimental impact on the data aggregation protocols. A typical mechanism to deal with joining nodes is proposed in [27]. In the mechanism, the aggregation task operates in consecutive epochs, and new nodes can join a system at any time. However, new nodes do not participate in the current epoch of the ongoing aggregation process, and they participate in the followed epoch. The restraint of fresh nodes is applied to allow the aggregation process in the current epoch to achieve convergence.

- In this project, joining nodes are studied and simulated using the robust protocols proposed in this chapter. Figure 5.9 shows a sample result of the simulations. The model description and results explanation are omitted due to the limited space and time. In this part of the investigations, we noticed that new nodes have to recognise the start of a new epoch in order to begin participating in the aggregation process. This requires the new nodes to perform regular message exchange with the existing nodes. Additionally, the adopted peer-sampling service does not specify nodes, either new or old. In consequence, the service may select a new node as a random peer, causing an existing node to share some mass with a new node. Also, some of new nodes may fail before they participate in the aggregation task. In summary, the robust protocol requires a mechanism to avoid sharing any mass among existing and new nodes, and new nodes have to promptly return the values that they may receive to the system.

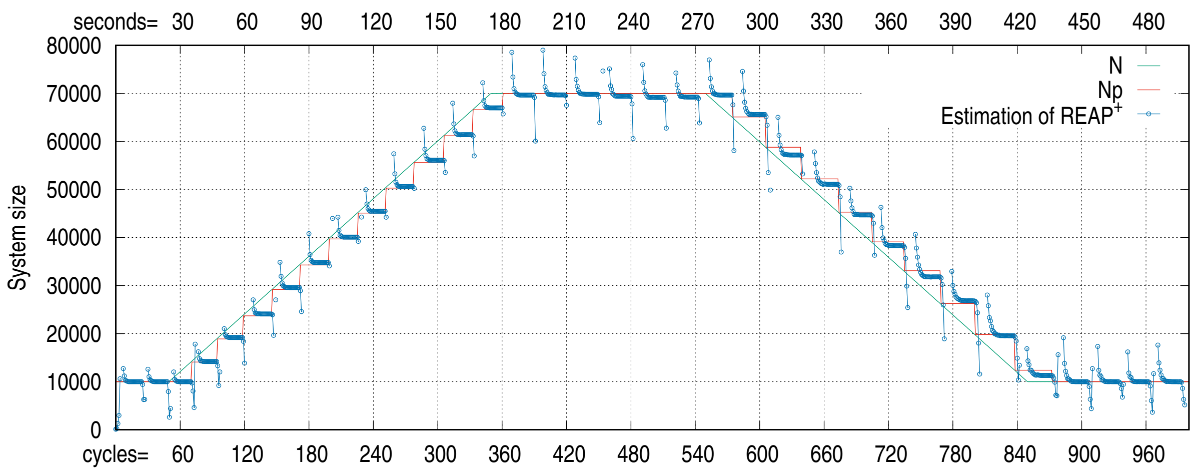


Figure 5.9: Performance of REAP$^+$ under churn of departing and joining nodes; initial $N = 10^4$, churn and join rates are 50%, $l^\Omega = 10$, $\ddot{\Upsilon} = 500ms$, $\hat{\Upsilon} = 3$, $\Upsilon = 3$, $\epsilon = 1$.

Two remaining problems obstruct the work of this project from achieving the perfect decentralised, fault-tolerant and consistent epidemic system:

1) The determination of the seed node problem, in which a single node (seed node) is needed for the initialisation of the aggregation process, i.e. *count* aggregation. All data aggregation protocols need the single-point initialisation in this project, e.g. SSEP, ECP and the REAP$^+$. Also, the problem is a widely known challenge for distributed and epidemic data aggregation [26, 30]. A data aggregation protocol can determine a

seed node using a leader election method or random selection with a given probability. However, the seed node is subject to single-point failure problem.

2) Although robust epidemic data aggregation such as REAP$^+$ converges under the presence of churn; the protocol cannot specify which convergence is achieved to the target value or another value. The tolerance threshold controls the aggregation error that can be tolerated in the local estimates of the protocol, but it does not determine the results and how close they are to the true target.

The next chapter, i.e., Chapter 6 introduces an alternative decentralised mechanism for the single-point initialisation, the chapter also proposes a restarting mechanism that detects the correct convergence to the target, and enable achieving continuous consistency and adaptability in the epidemic systems.

Chapter 6

Consistent Epidemic Systems

Services in large and extreme-scale distributed systems require continuous solutions for achieving the consistency and operational monitoring. The previous research presented in the thesis has proven that epidemic data aggregation is adequate tool for achieving global consistent state and for state determination, monitoring and consensus. However, in dynamic systems where churn is usually present, the accuracy of the aggregation results cannot be guaranteed and the results may significantly differ from the correct target value [27, 115]. The work in the previous chapter has identified the critical phase that has a direct impact on the robustness of the aggregation process. Also, two protocols for robust epidemic data aggregation are proposed with a distributed failure detection and mass restoration mechanisms.

The robust data aggregation protocols have detected the convergence of the aggregation process under severe level of churn, although the protocols adopt asynchronous message exchanging and processing cycles. This vital feature of the protocols has been proved through extensive practical experimentations. The adopted heuristic method for the local detection of convergence in the protocols is accurate and does not require any prior information about the target value. The method enabled the to detect the convergence on a good estimation of the target and the estimation accuracy depends on the level of the node churn. Despite that, robust aggregation protocols, e.g., REAP$^+$ cannot specify which convergence has been achieved; whether it to the target value or to another value? The thresholds control the aggregation error that can be tolerated in the local estimates of the protocol; however, the detection method in the protocols has no way to determine how close the results are to the desired true target.

The consistency in an epidemic system can be achieved by merging the robust epidemic protocols and the protocols for epidemic consensus. This way, the epidemic systems and services can make the most advantage of both approaches. However, large-scale distributed services requires continuous solutions for monitoring consistency and adaptability. The introduced protocols are usually presented as an epidemic task of one mission that may terminate after achieving the desired state. Real characteristics of the consistent systems are the constant operations, frequent updates, and adapting the system to changes in users or nodes [4, 37]. In consequence, a consistent epidemic system or service require exhibiting similar properties, which imposes the need for continuous data dissemination and aggregation protocols that frequently achieve global up-to-date state.

Existing continuous epidemic protocols are either restart periodically at fixed epochs or apply changes in the system state instantly producing less accurate approximations. In the literature, a simple restarting mechanism for epidemic protocols was introduced in [27], where global restarting was achieved using fixed-length epochs with incremental epoch identifiers. Nodes that receive a higher epoch identifier is enforced to join the new epoch with fresh initial data and discard the current state. Authors in [72] proposed a technique that restarts two overlapping aggregation processes in epochs of fixed hops. The protocol improves the aggregation results in dynamical conditions. The work in [36] introduces a continuous epoch-less data aggregation protocol. The protocol is atomic PUSH-PULL with a timeout, and it can produce correct results when prior information about system and churn rates are available. In the work of [75], FU aggregation protocol is introduced. The protocol operates under dynamic conditions without requiring periodic restarting. The protocol is proposed for averaging problems and requires symmetric exchange and correlation among nodes.

The work presented in this chapter introduces continuous epidemic protocol with a novel restart mechanism that enables systems adaptability. The proposed mechanism is adaptive, and restarts the epidemic task upon the detection of convergence or divergence in autonomous and variant-length epochs. Also, the mechanism ensures correct convergence to the target for all nodes through aggregating nodes decisions and acquiring consensus on the restart action. Moreover, the mechanism produces small communication overhead, which can be piggybacked on existing protocol messages. The proposed continuous protocol consists of several aggregation processes, sequential and parallel. Most processes are initiated for the global summation function, in which the initial weights follow a peak distribution and require a single node (*seed* node) to set the initial aggregate of weights [26, 30]. The determination of the seed node is a known challenge that requires a leader election process beforehand. However, the seed node is subject to single-point failure problem. The following section describes the seed selection method that is proposed in this project to overcome the single-point initialisation in epidemic data aggregation protocols. Section 6.2 describes the continuous protocol and the adaptive restart mechanism. The chapter ends with a discussion on the results and findings.

6.1 Data Aggregation and Seed Selection Method

In the global summation function, each node i initially starts with a data value x_i, $0 < i \leq N$ that represents a local measurement, attribute, or an implicit decision. The node also maintains a tuple of aggregate elements $\tau_i = \langle \varsigma_i, v_i, w_i \rangle$, where v_i is the data element and initialised with x_i, $v_{i,t0} = x_i$, and $t0$ is start time; w_i is the weight element of the tuple and σ_i is an identifier further described below. The initialisation of w_i determines the aggregation function. For the global summation and at start time $t0$, it is required to set $w_{\hat{i},t0} = 1$ at a single node $\hat{i}$ (*seed node*), and $w_{i,t0} = 0$ at all other nodes. In real-world decentralised systems, the determination of seed node $\hat{i}$ is challenging, and a seed node $\hat{i}$ is vulnerable to fail causing serious damage to the aggregation process.

To overcome this initialisation problem, we introduce the seed selection method as follows. The tuple identifier ς_i is the seed element and it is used as a '*seed*' selector. It is a synopsis of the minimum function and a target for the selection process. The seed element is a Unique Universal Identifier (UUID) generated by a global function $\mathcal{F}()$. There are two implementations of the function $\mathcal{F}$ in this work. $\mathcal{F}_\alpha(i,t)$ which computes a UUID given a node identifier i and the current time t. The output of $\mathcal{F}_\alpha(i,t)$ preserves the nature order, such that for any two UUIDs: $U_i = \mathcal{F}_\alpha(i,t_i)$ and $U_j = \mathcal{F}_\alpha(j,t_j)$, there is:

$$U_i < U_j, \iff t_i < t_j \vee (t_i = t_j \wedge i < j) \tag{6.1}$$

A practical sequence of the function $\mathcal{F}_\alpha(i,t)$ is given in 5.8. The output of the function $\mathcal{F}_\alpha(i,t)$ is a UUID that preserves the natural order, such that it is comparable to other UUIDs generated in the system. Moreover, the function parameters, which are node identifier i and current time t are used to ensure that for any two nodes, the generated UUIDs are different even if they used the function at the same time. On the other hand, the function $\mathcal{F}_\beta(i)$ generates a random UUID using node identifier as a random seed.

Initially, all nodes are seed nodes and the tuple τ_i is initialised to $\tau_i = \langle \mathcal{F}(), x_i, 1 \rangle$, where $\varsigma_{i,t0}$ identifies the tuple and the wight $w_{i,t0}$ in the system. The initial diffusion process in the aggregation process selects only one seed in the system for this epoch. During the diffusion process, seeds propagate in the system following a random-walk fashion. In line with seeds propagation, the selection process is carried out, targeting the oldest seed or the minimum seed identifier [52]. Each node retains the tuple with the lowest seed identifier and discards other seeds. In this way, nodes failure does not affect the aggregation process. In case a node with the minimum seed identifier fails before it propagates, the system will select the seed with the prior in order identifier.

The aggregation process works as follows, at each cycle, node i divides the data elements $v_{i,t}, w_{i,t}$ in two halves, and sends the tuple $\langle \varsigma_{i,t_i}, \frac{v_{i,t}}{2}, \frac{w_{i,t}}{2} \rangle$ to a random peer j in a PUSH message. Upon the reception of a PUSH message from j, node i selects the minimum seed element, $\varsigma_{\kappa,t_\kappa} = min(\varsigma_{i,t_i}, \varsigma_{j,t_j})$. In case a new seed tuple is selected, node i resets its local tuple to $\langle \varsigma_{\kappa,t_\kappa}, x_i, 0 \rangle$. Next, node i divides the elements $v_{i,t}, w_{i,t}$ in two halves and response to node j in a PULL message, which in turn performs the seed selection too, then both nodes update the tuple τ. The update in node i is accomplished, only and only if $\varsigma_{i,t_i} = \varsigma_{j,t_j}$ and so $\tau_i = \langle \varsigma_{i,t_i}, v_{i,t} + v_{j,t}, w_{i,t} + w_{j,t} \rangle$. Also, at each cycle, a fresh local estimate $e_{i,t}$ can be computed by $e_{i,t} = \frac{v_{i,t}}{w_{i,t}}$, and a node i achieves convergence when $e_{i,t}$ get arbitrary close to $\mathcal{V}$.

The convergence of the data aggregation process is described in section 3.2. In summary, the initial mass is the aggregate of all initial values in the system, $\mathcal{M}v$,$\mathcal{M}w$. During the aggregation process, the initial mass is propagated and aggregated. Eventually at time tc, the system converges and the mass distributes evenly over all nodes, The local estimate converges to the target value $e_{i,tc} \approx \mathcal{V}$, and the value of the target is $\mathcal{V} = \frac{\frac{\mathcal{M}v}{N}}{\frac{\mathcal{M}w}{N}} = \frac{\mathcal{M}v}{\mathcal{M}w}$. Thus, the target of the summation is $\mathcal{V} = \frac{\sum_{i=0}^{N} x_i}{1}$, and in counting is $\mathcal{V} = \frac{\sum_{i=0}^{N} 1}{1} = N$, while the target of averaging is $\mathcal{V} = \frac{\sum_{i=0}^{N} x_i}{N}$.

The aggregate *count* is a special case of the summation, where $v_{i,t0} = x_i = 1$, $0 < i \leq N$, and $\tau_i = \langle \varsigma_{i,t_0}, 1, 1 \rangle$. Aggregation of other functions such as *average*, *min*, *max* is simple because initially all nodes have the same weight element. The initial tuple at each node i is $\tau_i = \langle \varsigma_i, x_i, 1 \rangle$, and the seed element is a constant e.g. $\varsigma = 0$ at all nodes. In consequence, there is only one seed propagating in the system and no selection process is required.

The seed selection method removes the difficulty in determining a single node before the start of the aggregation process. It also protects the aggregation process from the single-point failure problem. The seed selection method is a decentralised method and generates no additional overhead. The method can be generalised and used for any selection and determination task. However, the method takes some time to converge to the correct seed due to the propagation delay in the diffusion process. The next section describes the adaptive restart mechanism and presents a practical example of the use of the seed selection method.

6.2 The Continuous Epidemic Protocol with Adaptive Restart Mechanism

The continuous protocol and the adaptive restart mechanism are illustrated in Algorithm 12. The protocol is a set of epidemic processes that run over sequential epochs, where each epoch has an incremental global identifier (ι). The epoch is the inter-restart interval, and two subsequent epochs identifiers may exist in the system for some time after restart. Nodes are enforced to join the epoch with the higher identifier. Epoch length is variant and depends on the detection of convergence or divergence. Some epidemic processes in the protocol are sequential and others are parallel. The process $\mathcal{A}$ corresponds to the intended mission of the epidemic task, which can be used for data dissemination and data aggregation too. The process $\mathcal{C}$ is a subsequent phase for achieving consensus. Nodes are joining the CONSENSUS phase after they achieve local convergence in the process $\mathcal{A}$, i.e., content convergence or aggregation convergence. Also, the protocol encompasses a tuple $\mathcal{P}$ of several aggregation processes such that each $p \in \mathcal{P}$ runs in parallel with the process $\mathcal{A}$. Processes in the tuple $\mathcal{P}$ are used for convergence detection, and their results define the convergence state, i.e., convergence or divergence.

The intended epidemic task defines the initialisation of the process $\mathcal{A}$. The process $\mathcal{C}$ and processes in $\mathcal{P}$ are all initialised for the aggregate *count*. The process $\mathcal{C}$ counts nodes which have achieved local convergence in the process $\mathcal{A}$, and each process p estimates the total number of nodes joined the process $\mathcal{A}$. Each process $p \in \mathcal{P}$ initially start with a different random seed identifier at each node in the system. During the aggregation process, seeds of all processes in $\mathcal{P}$ are piggybacked and propagated with the messages from the process $\mathcal{A}$. The seed selection method makes a random selection for each process due to the random seed initialisation. Moreover, a node failure will affect a random seed of each process and causes each process to achieve different convergence. Convergence state can be verified using local estimates $e_{p,t}$, $\forall p \in \mathcal{P}$. A correct convergence is confirmed when all local estimates in $\mathcal{P}$ converges to the same target, $\forall e_{p,t} \approx \mathcal{V}, p \in \mathcal{P}$. Otherwise, $\exists e_{p,t} \not\approx \mathcal{V}$ indicates a divergence, which implies experiencing dynamical conditions during the main epidemic process, i.e., the process $\mathcal{A}$.

The convergence detection method is presented in Procedure *DetectConvergence*. The method calculates the average of estimates in $\mathcal{P}$ every cycle and enqueues the average in $\mathcal{Q}$. Eventually, estimates average will converge to an approximation result, and the error among elements of $\mathcal{Q}$ becomes very small. The method verifies the detection of convergence using the SE of $\mathcal{Q}$ and monitors the error approaching the tolerance threshold ϵ_1 for several consecutive cycles Υ. Next, the method verifies the state of the convergence using the SE of estimates in $\mathcal{P}$. The criterion validates that errors among estimates in $\mathcal{P}$ is above the tolerance threshold ϵ_2. A true criterion implies the existence of errors among the estimates of the parallel processes in $\mathcal{P}$, which means a divergence has been detected. The thresholds ϵ_1, ϵ_2 and Υ are global application parameters as previously described in Section 3.3.

Upon the detection of divergence in a node i, the node initiates a global restarting process using a new epoch identified $\iota_i + 1$. The restart steps are described in Procedure *Restart*. Also, upon the detection of a correct convergence, node i makes a transition to the CONSENSUS phase by starting the process $\mathcal{C}$. Other nodes may join the phase at the same time or later when they converge. The seed selection process unifies the seed elements, and each node participates in the phase by adding 1 to the total data mass in the process $\mathcal{C}$. In the CONSENSUS phase, the detection method records the estimate of the process $\mathcal{C}$ in $\mathcal{Q}$ at every cycle. Each node uses the SE of $\mathcal{Q}$ and the thresholds ϵ_1 and Υ to locally detect the convergence of the CONSENSUS phase.

Achieving convergence in the CONSENSUS phase indicates the agreement among nodes to restart the epidemic task as they all have converged to the correct target. However, some nodes in the CONSENSUS phase are enforced to join the next epoch, although they did not yet detect convergence, which optimises the inter-times between epochs. Also, it adapts the epidemic task should it experience any dynamical conditions during the CONSENSUS phase.

Procedure *ResolveEpoch* has two duties: (1) discovering and joining new epochs, and (2) applying the seed selection method to unify seed elements in different processes. Each node receives a new epoch identifier starts a new epoch and reinitialise local process tuples, as shown in Procedure *Restart*. Also, the procedure updates the local tuples upon the detection of a new seed with the smaller identifier. The protocol in lines 7-9 continues processing the received message and responses to the sender node by a PULL message with the adopted epoch identifier and seed elements. In lines 14-16, the protocol updates the tuple for each process.

6.2.1 Experimental Results for The Continuous Epidemic Protocol

The protocol is examined and validated via simulations. Three events are used in the simulation: (1) *Start Event* occurs only once at the start time of each node. At this event, nodes initialise their seed and data elements. (2)*Run Event* is scheduled at every cycle, and the event stops after a predefined number of cycles. At every cycle, a node detects convergence and sends Push messages. (3) *Message Receive Event* is a notification event, in which a receiver node identifies new epochs, applies seed selection method and updates local tuples.

The task of process $\mathcal{A}$ is initialised for the aggregate *count* targeting system size estimation; and it is initiated so for the validation purposes only. It is worth to clarify that size estimation is a peak epidemic process and it is the most vulnerable process for the dynamical conditions. In consequence, seed elements of process $\mathcal{A}$ at all nodes are set using the function $\mathcal{F}_\alpha(i, ts)$, where $ts = [0, t_{off}[$, and t_{off} is a start time synchronisation offset as defined in Section 3.5. The settings of threshold parameters follow previous work recommendations in Section 4.5 and 5.4, and they are set to $\epsilon_1 = 0.5$, $\epsilon_1 = 1$, $\Upsilon = 3$, and $l^{\mathcal{Q}} = 10$. The protocol NCP$^+$ is used with k-regular overlay initialisation, and $k = 30$ and $\check{\Upsilon} = 10$.

Algorithm 12: The Continuous Epidemic Protocol with Adaptive Restart Mechanism

```
Require: tolerance thresholds $\epsilon_1$ and $\epsilon_2$; cycles threshold $\Upsilon$; estimates queue length
         $l^{\mathcal{Q}}$, processes tuple length $l^{\mathcal{P}}$.
Initialisation: at each node $i$: $\iota = 0$; $\mathcal{Q} = \emptyset$; $\widetilde{\mathcal{P}} = \{\mathcal{A}, \mathcal{C}\} \cup \mathcal{P}$; and
                $\forall p \in \widetilde{\mathcal{P}}, p \longrightarrow \langle \infty, 0, 0 \rangle$.

At start time $t0$ at node $i$:
$Restart(1, i, t0)$
$Push(i, t0)$

At each cycle $t$ at node $i$:
$DetectConvergence(i, t)$
$Push(i, t)$

At event 'receive message $m$ from $j$' at node $i$:
$ResolveEpoch(i, t, m)$
if $m.reply$ then
    $\widehat{\mathcal{P}} = \emptyset$
    foreach $p \in \widetilde{\mathcal{P}}$ do                // Divide data elements and copy tuples
        $p \longrightarrow \langle p.\varsigma, \frac{p.v}{2}, \frac{p.w}{2} \rangle$,      $\widehat{\mathcal{P}} \cup p$
    Send $\langle \iota_i, \widehat{\mathcal{P}}_i, reply = false \rangle$ to $j$                    // a Pull to node $j$
if $m.\iota == \iota$ then                           // Update local tuples in all processes
    foreach $p \in \widetilde{\mathcal{P}}$ do
        if $m.p.\varsigma == p.\varsigma$ then $p \longrightarrow \langle p.\varsigma, m.p.v + p.v, m.p.w + p.w \rangle$
```

```
def avg($H = \{a_1, \ldots, a_n\}$): $\frac{1}{n}\sum a$                         // Average
def se($H = \{a_1, \ldots, a_n\}$): $\frac{1}{\sqrt{n}}\sqrt{\frac{1}{n-1}\sum(a - \text{avg}(H))^2}$        // Standard Error
procedure Restart($\iota$,$i$, $t$)
    if $\iota_i == \iota$ then return                              // Not a new epoch, stop
    $\iota_i = \iota$
    $phase_i$ = AGGREGATION
    $\mathcal{A}_i \longrightarrow \langle \mathcal{F}_\alpha(i,t), x_i, 1\rangle$                    // Reset processes
    $\mathcal{C}_i \longrightarrow \langle \infty, 0, 0\rangle$
    foreach $p \in \mathcal{P}_i$ do $p \longrightarrow \langle \mathcal{F}_\beta(i), 1, 1\rangle$
procedure Push($i$,$t$)
    $\widehat{\mathcal{P}} = \phi$
    foreach $p \in \widetilde{\mathcal{P}}_i$ do             // Divide data elements and copy tuples
        $p \longrightarrow \langle p.\varsigma, \frac{p.v}{2}, \frac{p.w}{2}\rangle$,      $\widehat{\mathcal{P}} \cup p$
    $j \longleftarrow getRandomPeer()$                               // Get random peer
    Send $\langle \iota_i, \widehat{\mathcal{P}}_i, reply = true\rangle$ to $j$                    // a PUSH to node $j$
procedure DetectConvergence($i$,$t$)
    switch $phase$ do
        case AGGREGATION do
            $\mathcal{Q}_i \cup \text{avg}(\{p.e : \forall p \in \mathcal{P}_i\})$        // Insert estimates average in $\mathcal{P}_i$
            if $\text{se}(\mathcal{Q}_i) < \epsilon_1$ for $\Upsilon$ cycles then         // Detect local convergence
                if $\text{se}(\{p.e : \forall p \in \mathcal{P}_i\}) > \epsilon_2$ then            // Detect divergence
                    Restart($\iota_i + 1, i, t$)                          // Start a new epoch
                else                          // Make transition to CONSENSUS phase
                    if $\mathcal{F}_\alpha(i,t) < \mathcal{C}_i.\varsigma$ then $\mathcal{C}_i \longrightarrow \langle \mathcal{F}_\alpha(i,t), 1, 1\rangle$
                    else $\mathcal{C}_i \longrightarrow \langle \mathcal{C}_i.\varsigma, \mathcal{C}_i.v + 1, \mathcal{C}_i.w\rangle$
                    phase=CONSENSUS
        case CONSENSUS do
            $\mathcal{Q}_i \cup \mathcal{C}_i.e$
            if $\text{se}(\mathcal{Q}_i) < \epsilon_1$ for $\Upsilon$ cycles then        // Detect global convergence
                Restart($\iota_i + 1, i, t$)                              // Start a new epoch
procedure ResolveEpoch($i$,$t$,$m$)
    if $m.\iota > \iota_i$ then Restart($m.\iota, i, t$)                   // New epoch discovered
    if $m.\iota == \iota_i$ then                                  // Resolve seed elements
        if $m.\mathcal{A}.\varsigma < \mathcal{A}_i.\varsigma$ then $\mathcal{A}_i \longrightarrow \langle m.\mathcal{A}.\varsigma, x_i, 0\rangle$
        if $m.\mathcal{C}.\varsigma < \mathcal{C}_i.\varsigma$ then
            $\mathcal{C}_i \longrightarrow \langle m.\mathcal{C}.\varsigma, 0, 0\rangle$ and if $phase$==CONSENSUS then $\mathcal{C}_i.v = 1$
        foreach $p \in \mathcal{P}_i$ do
            if $m.p.\varsigma < p.\varsigma$ then $p \longrightarrow \langle m.p.\varsigma, 1, 0\rangle$
```

The results in Figure 6.1 illustrate performance of the continuous epidemic pyrocoll, in particular, the seed selection method and restart mechanism behaviour. In these experiments, nodes churn is disabled and the result shows the protocol behaviour under stable conditions. The results for the processes $\mathcal{A}$ and $\mathcal{C}$ are distinguished for the clarity. The figure 6.1.a and 6.1.b show the variation in initial system mass $\mathcal{M}_v$, $\mathcal{M}_w$ over time. In the figures, data elements approach the correct value as a result of the selection method. Particularly, the figure 6.1.b presents the decrease in $\mathcal{M}_w$ value due to the selection of the correct seed and discarding of other seeds.

Figures 6.1.c, 6.1.d and 6.1.e illustrate the convergence in each phase and the correct detection of convergence. The results validate the efficiency of the protocol and the restart mechanism. The protocol in each node makes transition to the CONSENSUS phase after the detection of the convergence in the AGGREGATION phase. Also, it restarts the aggregation task after achieving convergence in the CONSENSUS phase. The figure 6.1.d shows the reduction in variance of nodes' estimates in the AGGREGATION phase. The estimation error continue decreasing among nodes although some nodes have converged and join the CONSENSUS phase. This behaviour is due to retaining data exchanging and computations for the processes $\mathcal{A}$ and $\mathcal{P}$ active even for nodes in the CONSENSUS phase. This is very important to allow nodes which still in the AGGREGATION phase to converge. Figure 6.1.e shows that 100% of nodes achieve and detect true convergence in both phases, and nodes restart and join the new epoch in a uniform asynchronous manner.

To demonstrate the probabilistic performance of the protocol, the results of 30 experiments are collected and used to export statistical summery of the protocol performance. Figure 6.2 presents the summery results. The results in Figure 6.2.a are collected using a dedicated oracle observer, and they show nodes that have achieved true-convergence in processes $\mathcal{A}$ and $\mathcal{C}$. The figure clearly shows that despite the probabilistic natural of the protocol, all nodes in process $\mathcal{A}$ achieve true-convergence before they make transition to the CONSENSUS phase. Also, nodes in the CONSENSUS phase achieve true-convergence before they restart. Figure 6.2.b confirms previous results and shows small variance in estimation errors in all epidemic processes. Further experiments are carried out for Figure 6.2.c, which shows results for various system sizes. The figure shows logarithmic increase in inter-restart times as system size increases. It also presents the variation in the inter-restart times for different sizes due to the probabilistic behaviour of the continuous protocol.

Results in figures 6.3.a and 6.3.b illustrate the behaviour of the protocol under dynamic conditions. Two experiments are carried out for nodes churn in the simulations. The first experiment examines the protocol sensitivity to a single node failure and to moderate churn rates. Figure 6.3.a shows the results for a single failure injected at cycle 5 followed by the failure of 30% of the system in cycles $[60 - 120]$. The second experiment tests the protocol under severe dynamical condition when a system loses 75% of its nodes during an epidemic task. Figure 6.3.b shows the failure of 75% of the system between cycles $[15, 195]$. In both experiments, results prove the ability of the protocol to detect divergence and it can restart for as small as a single failure. The protocol continues until the system stabilises before it can return a correct estimation and enters the CONSENSUS phase.

The impact of varying the parameter $l^{\mathcal{P}}$ has also been examined; however, The results have shown that the effect was negligible under stable conditions while causing a small increase in overhead to the underlying network. In dynamic conditions, the increase in the number of processes in $\mathcal{P}$ makes validation of convergence more accurate, especially, for the detection of divergence. Although precision is essential for the detection of a small amount of churn, e.g. single node failure, as shown in figure 6.3. The tolerance thresholds can also control the accuracy, and with low cost, and hence, the description of experiments on the parameter $l^{\mathcal{P}}$ are omitted. From another perspective, the amount of error that the protocol can tolerate corresponds to the lost portion of the initial system mass. In early cycles of an aggregation process, node failure may cause a major loss in the system mass, however, after convergence the impact of churn fades, see the analysis in Section5.2. Therefore, even large churn rates at late cycles can not result in divergence detection, the figure 6.3.b shows this scenario in the cycles $[15, 60]$.

Simulation results have validated the performance of the protocol in stable and dynamic conditions. Epidemic services and systems can adopt the restart mechanism and achieve efficient and reliable restart. The method is capable of restarting epidemic processes to adapt to changes in the system. Through periodic restarting, epidemic processes can detect and adapt to new conditions. In addition, services such as WSN can also track changes in sensor nodes. The change in captured values in this epoch can be recorded and aggregated for the followed epoch. Moreover, network failures and message loss can also be handled to some extent.

6.3 Discussions

In large and extreme-scale distributed systems, continuous epidemic protocols are useful services for systems consistency and adaptability. Through periodic restarting, epidemic processes can detect and adapt to new conditions. The work in this chapter has introduced our latest contribution, which is a novel continuous epidemic protocol with an adaptive restart mechanism. The protocol restarts either upon acquiring consensus on the global convergence of the epidemic task or upon the detection of divergence. Up to our knowledge, the ability of distinguishing convergence state, i.e., correct convergence or divergence, under dynamical conditions has not been addressed before in the literature. Typically, it is known that epidemic protocols do not converge in unstable systems [27, 33, 115]. The continuous epidemic protocol proposed in this chapter has the ability not only to converge under churn but also to detect churn impact on the results. Also, the protocol introduces a decentralised selection method for data aggregation tasks that require single-point initialisation. Furthermore, the detection accuracy of the protocol can be tuned according to the application preference for a good quick approximation or an accurate one that takes longer to compute. Moreover, the implemented restart mechanism has optimised communications overhead that can be piggybacked with regular message exchanging. Simulation results have validated the performance of the protocol under stable and dynamic conditions. The following discusses some relevant points:

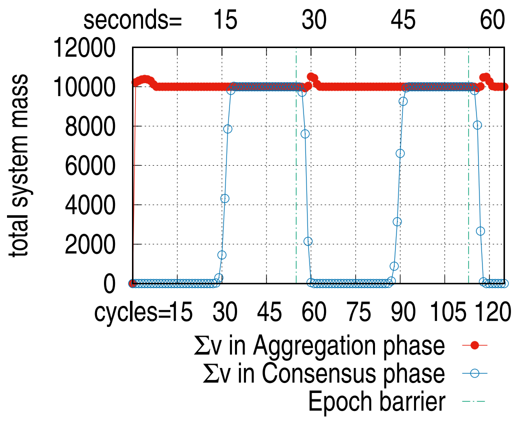
- The time complexity of the continuous epidemic protocol is $O(2\log(N))$ in stable conditions, as it has two phases. While it is $O(log(N))$ in dynamic conditions as the protocol restarts upon the divergence detection, which is considered an optimisation feature of the protocol rather than proceeding with an epidemic task that leads to wrong results. The protocol exhibit the complexity of the communications of PUSH-PULL protocols with overall overhead $O(2N)$.

- Considering the consensus and agreement protocols introduced in Chapter 4, e.g., ECP, the adaptive restart mechanism can be used in the agreement phases and achieve a decentralised decision-making mechanism with local results at each node. The mechanism enables the seamless detection of disagreements and withdraws, as nodes which decide not to proceed with the agreement can withhold itself from participating in the process, in addition to detecting the presence of failed nodes. The global tolerance threshold can be used to adjust the required majority for the agreement to make the transition to the next phase or, otherwise, to restart the agreement process. Another example on the advantage of the adaptive restart mechanism is that it has eliminated the need for the leader-election in the AGGREGATION phase in ECP. The seed selection method allows every node to participate in the agreement process, and the selection method decides which seed to remain in the process.

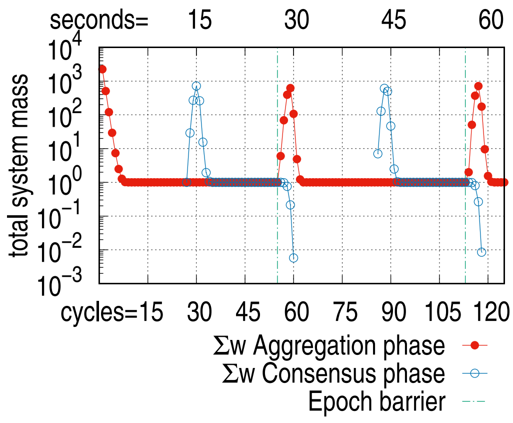
- The seed selection method exhibit a distinct feature over the leader election technique. Both techniques require propagation time before achieving the desired outcome. In seed selection method and at the start, distributed seeds propagate into the system, and each node selects a seed according to a predefined criterion that ensures a single seed to remain in the system. In decentralised leader election, candidates propagate into the system, and each node elects a candidate following a predefined criterion that elects a single leader in the system. However, a leader may fail and cause damage to the system. In consequence, the system needs additional steps and overhead to inform current leader failure and to elect another leader. In the seed selection method, the seed never fails, but the node which has generated the seed may fail. There are two cases of failure of seed originator node: (1) the originator fails before sharing the seed with any other node. In this case, the seed can be ignored and considered as never being in the system. The system will select the next in order seed. (2) The originator fails after sharing the seed with the system. The seed information, in this case, will propagate and will be selected regardless of the failure of its originator. The system does not require any further steps to detect originator failure or to propose a new seed. In some scenarios, a system may require a leader election step to perform a global action or decision using some aggregated information. In this case, the system can distribute the information with appropriate seed identifiers and allow each node to perform the designated action or decision using the information associated with the globally selected seed. It is information selection with decentralised action rather than decentralised selection with single-point action, which is subject to failure.

- One concealed issue in the seed selection method is that it requires distributed unique universal identifiers, not only that but also, the identifier has to preserve the natural order of numbers. Although such identifiers have been achieved in the practice of this research work, it might be infeasible to make universal identifiers with such requirements in some applications. In this case, random universal identifiers can be utilised. However, it is vital to note that using the random universal identifiers in the seed selection method may violate the single seed requirement in the mechanism, because there is a small probability that two random identifiers generated at different nodes or in various times can be the same. To clarify, the natural order of the seed identifiers ensures that the seed selection method will eventually converge to the globally lower seed proposed by any node in the system. Random identifiers do not preserve this principle. Potentially, a globally lower random seed can be generated in the system at the time when another seed was selected, and some nodes have detected the convergence on the oldest seed. In this case, two valid seeds exist in the system at the same time. This issue can be solved using another aggregation process of the *count* function to estimate the number of nodes that are using a particular seed, and proceed with the seed that has the majority. In case a new valid seed appears later in the system, nodes still select the oldest seed, which has propagated in the system and acquired the highest number of nodes.

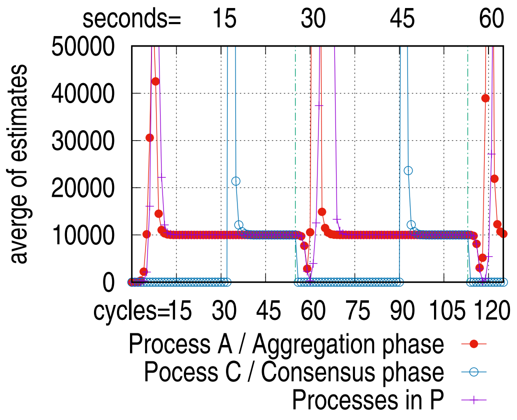
In the future, the research work may investigate how utilising multiple aggregation processes in parallel under churn can help to detect results quality. Also, how to use the aggregation results of different processes after convergence to compute an approximation of the true target value.



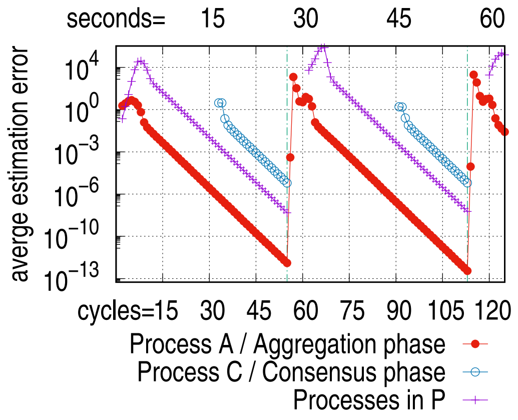
(a)



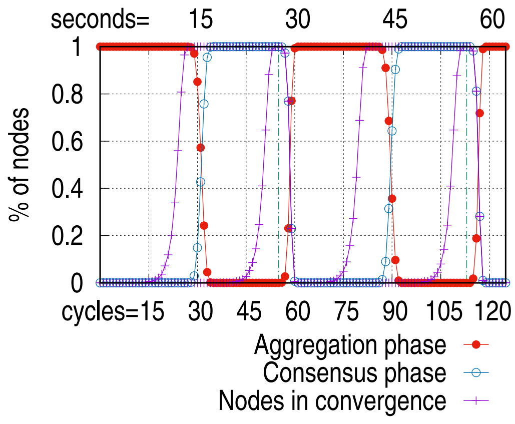
(b)



(c)

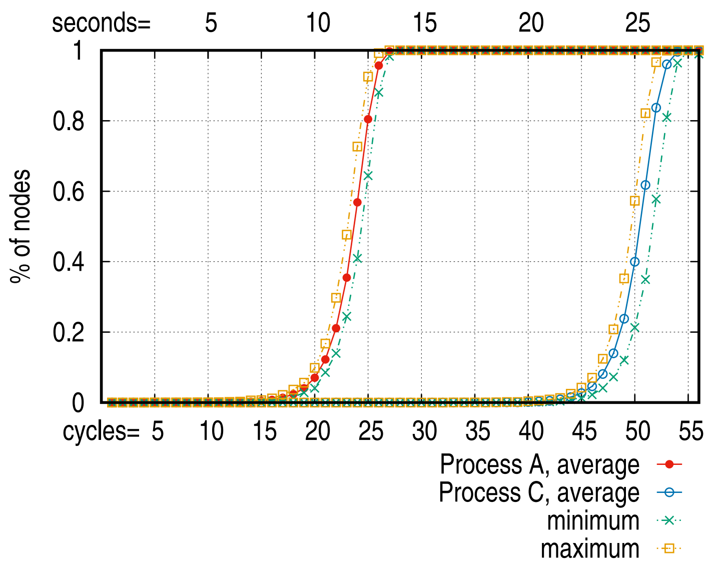


(d)

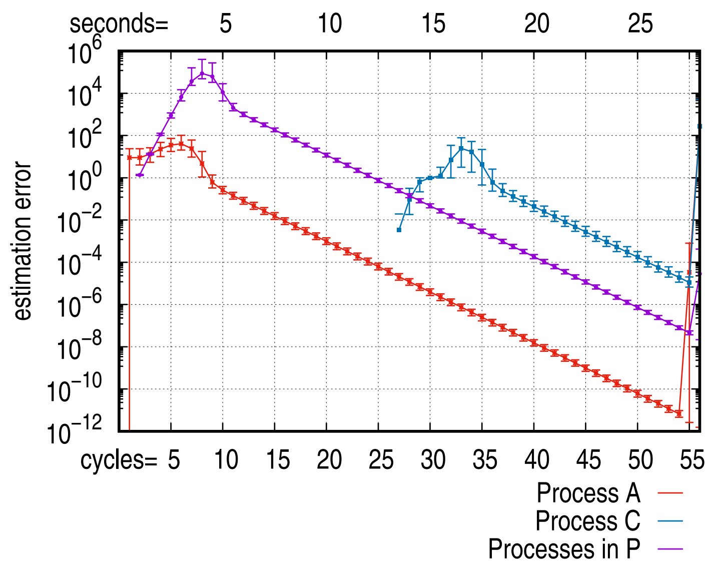


(e)

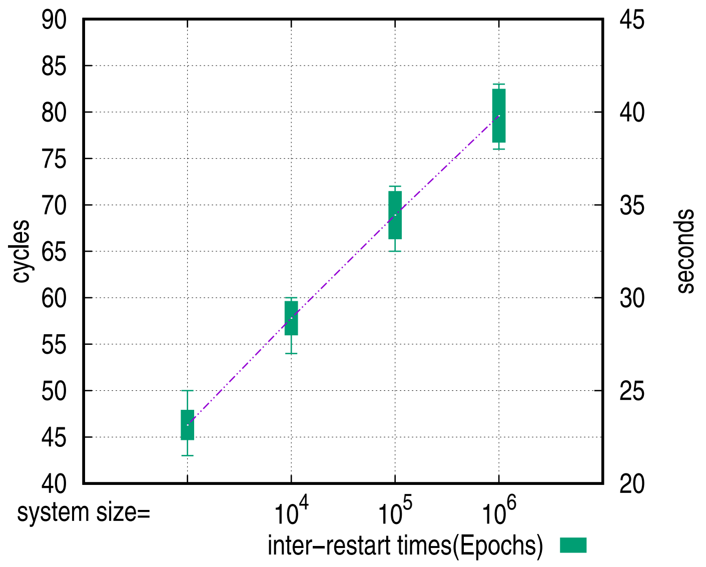
Figure 6.1: The continuous protocol and adaptive restart mechanism in stable conditions, $\mathcal{V} = 10^4$, $\epsilon_1 = 0.5$, $\epsilon_2 = 1$, $\Upsilon = 3$, $l^{\mathcal{P}} = 5$, $l^{\mathcal{Q}} = 10$, $\ddot{\mathfrak{T}} = 500ms$.



(a)

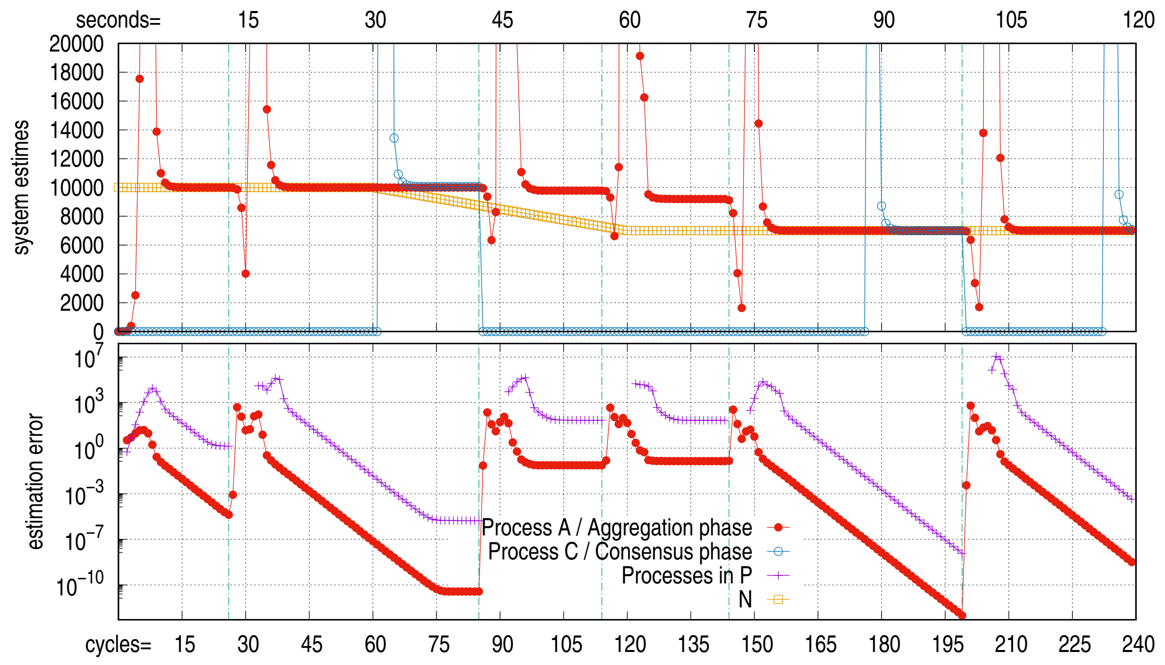


(b)

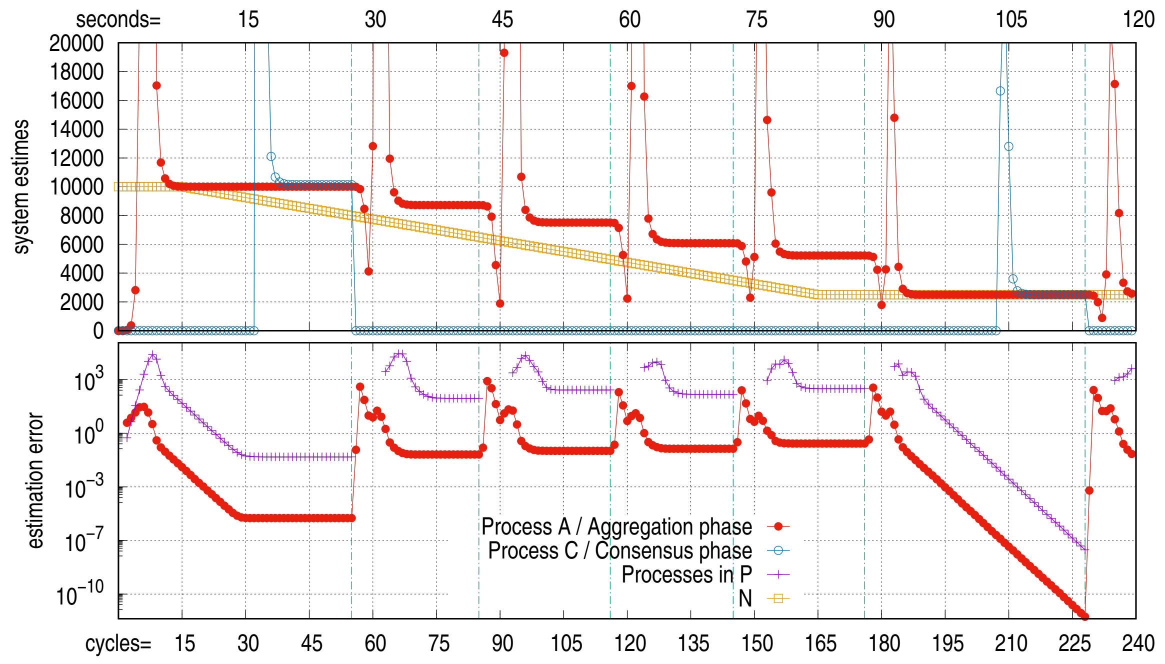


(c)

Figure 6.2: Probabilistic performance of the continuous protocol in stable conditions, $\mathcal{V} = 10^4$, $\epsilon_1 = 0.5$, $\epsilon_2 = 1$, $\Upsilon = 3$, $l^{\mathcal{P}} = 5$, $l^{\mathcal{Q}} = 10$, $\ddot{\mathfrak{T}} = 500ms$.



(a) Moderate churn



(b) severe churn

Figure 6.3: The continuous protocol and adaptive restart mechanism in dynamic conditions, $\mathcal{V} = 10^4$, $\epsilon_1 = 0.5$, $\epsilon_2 = 1$, $\Upsilon = 3$, $l^{\mathcal{P}} = 5$, $l^{\mathcal{Q}} = 10$,$\ddot{\mathfrak{T}} = 500ms$

Chapter 7

Conclusions and Further Work

This chapter presents a summary of the work introduced in this thesis and the aims and objectives which have been achieved. In the beginning, a short recap of the problem which has been addressed in the research project is provided. Section 7.2 demonstrates the main findings and achievements and concludes the research work. In Section 7.3, research ideas for further study are given.

7.1 Recap of the Research Problem

In the recap, decentralised algorithms have emerged as an alternative model of distributed systems in response to challenges in centralised approaches [12, 13]. Particularly, epidemic algorithms which adopt randomised communications and decentralised computations are found attractive to distributed systems due to their intrinsic characteristics such as the natural diffusion, fault-tolerance, scalability, and resilience.

Distributed services have used epidemic models to accomplish two essential tasks: information dissemination and data aggregation. Mainly, epidemic data aggregation has received wide attention and extensive research work, which made data aggregation a substantial component in the formation of distributed services. However, the consideration of real-world scenarios imposes important issues on the reliability and robustness of distributed systems, which are scalable, asynchronous and highly dynamic. Asynchrony and dynamism have a detrimental effect on the efficiency and the correctness of epidemic models [33, 34]. In real-world systems, the intrinsic properties of epidemic models cannot be ascertained and may get damaged from the unpredictable behaviour of nodes.

Theoretical analysis of epidemic models provided stochastic guarantees to the convergence of all nodes in stable systems. In addition, practical studies proved the ability of each node to detect convergence using only its local state [28, 32]. However, in dynamic systems where churn is usually present, the accuracy of the data aggregation cannot be guaranteed, and the results may significantly differ from the correct target of the aggregation [27, 115]. This detrimental impact of churn is a direct result of the violation of the mass-conservation invariant [26, 30], which lead to an estimation error of the target.

In another respect, modern large-scale distributed services demand continuous and decentralised solutions for achieving system consistency. Generally, coordination among the participants of a distributed system is required to make a globally consistent state in the system. Systems consistency allows the accomplishment of system-wide tasks such as termination, event ordering and decision-making. In epidemic systems, consistency supports the reliability and predictability of the system by making analogous state at all asynchronously acting nodes. However, achieving a globally consistent state is not a trivial problem that requires not only decentralised and resilient solutions but also, robust and continuous protocols that can adapt to changes in dynamic systems.

Systems consistency is attainable through distributed algorithms that ultimately converge to a global state. In particular, algorithms for distributed consensus are effective techniques to achieve system consistency [40]. Decentralised algorithms can achieve consensus in large-scale distributed systems, for which a collective decision-making process is accomplished targeting the agreement on some output. Likewise, the agreement in epidemic systems requires all nodes to achieve a particular state at the end of a data dissemination or aggregation task [22, 46, 48]. The agreement process in epidemic systems aims to compute a synopsis for initial system values and eventually, all system nodes converge to the target value. The similarity of agreement processes in distributed consensus and data aggregation process is an unsurprising finding due to the well-known guarantees on the convergence. The ultimate convergence of decentralised algorithms has motivated many researchers to use the attaining of convergence as a synonym to the reaching of agreement, primarily, when the target value is the interest of the consensus.

In stable systems, decentralised models for agreement assume that a system will eventually converge in a finite time, in which the agreement is entirely probabilistic, and a node has no explicit certainty on the convergence of system nodes. The heuristic detection of convergence is inadequate to decide the consensus. Typically, nodes achieve convergence at different times, and each node has no awareness about other nodes' state of convergence [51, 52]. Furthermore, computations in epidemic models are decentralised by nature, and the detection of convergence can only be obtained through the absolute reliance on local state at each node. Also, the certainty of global convergence cannot be directly presumed by local detection due to nodes asynchrony. In the lack of a central authority like in commitment protocols, it is a challenge to use only locally available information at each node in acquiring the awareness of other nodes' convergence.

The research work in this project has extended the distributed consensus problem to distributed data aggregation problem where the agreement process is an epidemic data aggregation process. In order to achieve agreement on a particular matter, nodes in a system have to receive a particular data item or compute a local estimate of a target, detect local convergence on the target, and acquire explicit awareness on the convergence of the system. Epidemic data aggregation is the core of the agreement process and therefore, the accuracy of the aggregation process is essential, and the process ability to achieve the convergence needs to be protected, especially in dynamic systems. In addition to the explicit detection of both the convergence and the agreement, the consideration of real-world scenarios requires continuous protocols to achieve system adaptability and consistency.

7.2 Conclusions

This section presents the list of conclusions deduced from the research work in this project. Each of the following points is a conclusion statement of an objective or a finding:

1) In distributed systems, the FLP result has motivated the identification of the minimal properties of distributed systems that are necessary to solve the consensus problem. In particular, the adoption of randomised communication reduces the potential causes of the impossible case. Also, the global agreement can be achieved in a partially synchronous model with prior defined bounds on the communication delays and processing times. Asynchronous protocols that can be adapted to terminate efficiently in practice can achieve consensus too. Epidemic systems naturally exhibit fault-tolerant and randomisation properties. Also, they lack the disastrous single point failure and implement significantly efficient diffusion process. Moreover, epidemic protocols eventually converge with high probability, which fulfils the termination condition in consensus problem.

2) The adopted model of the epidemic system in the project is a partial-synchrony model, in which the upper bound of communication delays is defined, and the processing cycle length is fixed. However, the model can be relaxed to complete asynchronous implementation. Epidemic protocols proposed in the project adopt asynchronous message exchange, i.e. the Symmetric-Push Sum model, in which non-atomic PUSH-PULL scheme is used. Thus, nodes do not lock waiting for a response, and interleaving messages are typically present. Also, communications among nodes are indeterministic due to the adopted overlay network manager that produces dynamic topologies. Although communication delays are generated randomly using statistical distribution with predefined parameters correspond to the RTT on the network diameter, there is no assumption restrains message delays to the defined upper bound. Moreover, the processing cycle length is fixed, and cycles are not synchronised, which implies that any two nodes may have different cycle counters at a particular time, and cycles of different nodes may overlap. In consequence, messages may cross boundaries between cycles of different nodes. Long delays make messages delivery to different cycles too. Thus, the defined upper bound of delays can be released as it has no impact on the performance of the protocols in the system, apart from small delay in the convergence. However, the epidemic system model in this project assumes no message loss.

3) Processing cycle length defines the convergence speed in the realistic implementation.

Applications need to adjust the cycle length to the adequate value to optimise the efficiency. Cycle length needs to be considered with the latency of the underlying network. Choosing a short length may cause a faster convergence due to the increase in communication rate per unit time. Also, a length that is too short will flood the underlying network with messages, and the convergence speed will be susceptible to communication delays, e.g. message queuing and congestion. The increase in transmitted messages will make them more vulnerable to network faults and may cause severe damage to epidemic tasks due to losing some system mass in the network. On the other hand, a large length makes the convergence slower and may expose epidemic tasks unnecessarily longer to system dynamics. The sufficient length of the cycle should allow a system to converge reasonably fast and with very high probability. For epidemic protocols that adopt pairwise message exchange such as Push-Pull and the Symmetric-Push Sum schemes, a cycle length that is long enough for most exchange transactions to complete within a cycle is recommended. In particular, setting the cycle length to the RTT on the diameter of the network is the most practical choice for epidemic services.

4) The local detection of convergence in epidemic systems is critical. It defines the point of which the local state is considered mature and can be admitted for a task or a service. It provides local synopsis for the global state, and it can be used to apply system-wide action, decision or event. Convergence detection methods in data aggregation are usually heuristic and require some application-specific parameters, i.e., error tolerance and consecutive cycles threshold. The settings of these parameters typically require global information on the system initialisation, e.g., system size or data distribution. In real-world systems, such information is unavailable or hard to obtain epically in the presence of dynamic conditions. The work in this project has evolved a novel heuristic convergence detection method with statistical formulas, mainly, the Standard Error (SE) is used; because SE formula does not require any prior information about the target value and provide more accurate convergence detection. Also, the method limits the setting of the application-specific parameters to the minimum amount that either well-know or easy to guess.

5) The work in Chapter 4 has introduced the Phase Transition Algorithm PTA, which is proposed to achieve globally consistent states in large and extreme-scale epidemic systems. Also, two innovative epidemic protocols, namely PTP and ECP, are proposed. Simulations have validated the protocols, and they have achieved the explicit agreement on the data dissemination and aggregation. The PTA is flexible, and the agreement phases in the algorithm can be cascaded to achieve further awareness on the target. The algorithm inherits the intrinsic properties of epidemic models and achieves the global agreement without deterministic communication or network structures. It is a typical decentralised, scalable and fault-tolerant solution to modern services in extreme-scale distributed systems such as consensus and consistency.

The protocols PTP and ECP have a general time complexity of $O(M\log(N))$, where M is the number of phases in the protocol and N is system size. They can commit in $t_{action} = M \times tc$ where $tc = \log(N) + \log(\frac{1}{\epsilon}) + \Upsilon$. Also, the PTA has higher overall communication overhead. Generally, the communication load is distributed over all system nodes. Protocols of PTA have an overall overhead complexity $O(2Nt_{action})$.

Action implementation after achieving global agreement in the PTA is a matter that requires global consideration. Nodes do not converge at the same time due to the asynchrony, and nodes have no explicit knowledge of the convergence of other nodes. Therefore, it is critical to all nodes in a particular phase to converge for a correctly global agreement. A node that promptly implements an action such as *stop* or *leave* after it detects global convergence locally can cause a detrimental effect on the aggregation process of the current phase and prevent other nodes in the same phase from achieving convergence. To avoid such effect, nodes must continue participating in the aggregation process for a sufficient time tc to allow other nodes in the phase to converge. At this stage, N is known to all nodes, and a reasonable value of tc can be computed and used by a node to continue active before implementing any local actions. For making a system-wide action or decision, e.g., *restart* or *terminate*. The first node that detects global agreement starts a propagation process for the action. However, other nodes may receive information on the action before they achieve convergence, and in this case, they are enforced to apply the action.

6) In Chapter 5, the problem of distributed data aggregation under nodes churn and network failures has been studied. Typically, the data aggregation process is susceptible to damage from the churn of nodes and may produce incorrect results or may not produce results at all. Our analysis of the data aggregation process presented in section 5.2 has led to the identification of three implicit phases in the process. Investigations have shown that nodes churn has a different impact on each phase. Also, the phase which is critical for the aggregation process is identified and has received further study, leading to introducing new robust data aggregation protocols that recognise the implicit phases, and achieve higher accuracy in the presence of nodes churn. Two protocols are introduced, REAP that implements the *Push-Release* model and REAP$^+$ which implements the *Pull-release* model. The protocols are robust epidemic data aggregation protocols that converge to a good approximation of the target value when churn is present. The protocols are lightweight, resilient and can be used in combination with any peer-sampling service or membership protocol. They can be used to build scalable decentralised services independent of the underlying topologies. Moreover, the protocols implement an innovative, distributed pairwise replication mechanism in addition to the distributed failure detection and instantaneous mass restoration mechanisms. In general, the robust data aggregation protocols proposed in this work achieve competitive accuracy in comparison to other aggregation protocols.

The protocols REAP and REAP$^+$ have preserved the intrinsic time complexity of

PUSH-PULL protocols, which is $O(\log(N))$. However, the communication complexity in the protocols increases by one message at the PROPAGATION phase, i.e., 3 messages per node per cycle. After convergence, messages number retains equivalent to the PUSH-PULL protocols, i.e., 2 messages per node. At $t < tc$ the overall overhead is $O(3Np)$ and $O(2Np)$ otherwise.

An important outcome of the work presented in Chapter 5 is the ability to detect the convergence in the aggregation process under all levels of churn. The robust protocols adopt asynchronous message exchanging and processing cycles, and because they can detect convergence, they can eventually and effectively terminate. This achievement satisfies the requirements for asynchronous distributed systems to make the agreement and acquire the consensus. Although, epidemic protocols eventually converge in high probability in stable networks, our work has practically proved that the robust epidemic protocols converge and detect convergence under dynamic conditions, and even under the worst churn scenarios. Thanks to the adopted heuristic method for the local detection of convergence which has enabled such essential feature. The method recognises the state of convergence in aggregation results when the target value is not available. In stable systems, data aggregation protocols that used the heuristic method have detected the convergence to the target value with a desired error that is tolerated. In dynamic conditions, the method detects the convergence on a good estimation of the target value, and the estimation accuracy depends on the level of the node churn.

Also, Chapter 5 presented a survey on the node churn, particularly in P2P networks and systems. The survey aimed to adopt the typical churn model for studying the epidemic data aggregation process. During the survey, it was difficult to extract accurate churn rates for short time intervals due to the acquisition methods which are used in the research studies. However, and with general assumptions on node departure, estimation for expected departure rates were obtainable from the statistical distributions of the session duration provided in the studies. The survey has shown that in normal system conditions, 30% of nodes churn in the average should be expected. This result implies that churn rates are usually moderate, and data aggregation protocols are required to provide acceptable accuracy in this level of churn. Higher churn levels are also considered; however, the higher levels have a low probability of occurring. Additionally, it has limited global impact on decentralised systems which are deployed over a large-scale network.

7) In large and extreme-scale distributed systems, continuous epidemic tasks are useful models for monitoring and maintaining system consistency. Through periodic restarting, epidemic processes can detect and adapt to new conditions. The work in Chapter 6 introduced a novel continuous epidemic protocol with an adaptive restart mechanism. The process restarts either upon acquiring consensus on the global convergence of the epidemic task or upon the detection of divergence. Moreover, the mechanism preserve

optimised communications overhead that can be piggybacked with regular message exchanging. Also, the protocol introduces a decentralised selection method for data aggregation tasks that require single-point initialisation. Simulations validated the performance of the protocol under static and dynamic conditions. Furthermore, the detection accuracy of the protocol can be tuned according to the application preference for a good quick approximation or an accurate one that takes longer to compute.

The seed selection method addresses the determination of a single node problem in the data aggregation initialisation. For instance, the global summation function requires a single node (*seed* node) to set the initial aggregate of weights. The determination of the seed node is a known challenge that requires a leader election process beforehand. However, the seed node is subject to single-point failure problem. The seed selection method is proposed to overcome the single-point initialisation in the data aggregation, and in distributed and epidemic systems in general. Apart from the universally unique identifiers, the method does not require any specific initialisation. All nodes are initially seed nodes, the diffusion process in the data aggregation selects only one seed in the system. During the diffusion process, seeds propagate in the system following a random-walk fashion. In line with seeds propagation, the selection process is carried out, targeting the oldest seed or the minimum seed identifier. In this way, nodes failure does not affect the aggregation process. In case a node with the minimum seed identifier has failed before it propagates, the system will select the seed with the prior in order identifier.

The seed selection method requires distributed unique universal identifiers, not only that but also, the identifier has to preserve the natural order of numbers. Although such identifiers have been achieved in the practical work in this project, it might be infeasible to make universal identifiers with such requirements in some applications. In this case, random universal identifiers can be utilised. However, it is vital to note that using the random universal identifiers in the seed selection method may violate the single seed requirement in the data aggregation process. To clarify, the natural order of the seed identifiers ensures that the seed selection method will eventually converge to the globally lower seed proposed by any node in the system. Random identifiers do not preserve this principle. Potentially, globally lower random seed can be generated in the system at the time when another seed was selected, and nodes have detected the convergence on the oldest seed. In this case, two seeds exist in the system at the same time, which invalidates the single seed requirement. By accompanying the seed selection method with another aggregate of the *count* function to count the number of nodes that using a particular seed, and proceed with the seed that has the majority is a practical solution for using random identifiers in the seed selection method.

Finally, protocols proposed in this project provide variety of selections to build epidemic large scale services and applications for stable and dynamic systems. A particular epidemic service can combine the robust data aggregation protocols with the PTA and achieve the ultimate epidemic protocol that achieves agreement in dynamic systems. The ECP for instance, may use the robust data aggregation protocol in the agreement process and achieve consensus in the presence of node churn. However, the decision-making process in the agreement phases need to tolerate higher error and achieved consensus on the majority rather than on a complete agreement. Also, ECP may implement the adaptive restart mechanism in the agreement phases. The mechanism enables seamless detection of nodes that wishes to disagree or withdraw, in addition to detecting the presence of failed nodes. The global tolerance threshold can be used to adjust the required majority for the agreement to make the transition to the next phase or, otherwise, to restart the agreement process or even the whole task.

This section has described the achieved findings and concluded the accomplished work. Although the work in this project was challenging and complicated mission, the presented work has achieved all the stated aims and objectives. Overall, the research work in this thesis makes noticeable contributions to the discipline of epidemic systems. The introduced protocols provide fundamental solutions for many applications and services in the modern distributed systems.

7.3 Further work

The following points are recommended suggestions for the future work as an extension to this research project:

1) Nowadays, information systems in private and public sectors utilise Blockchain technology to obtain a more secure and transparent transactional recording. It is a new decentralised way to deal with assets, markets, and financial ecosystems. However, the distributed ledger in the Blockchain requires several minutes to approve and achieve consensus on a particular transaction, especially when a full-decentralisation is a requirement, for example, the Proof-of-Work (PoW) in Bitcoin systems. Settling a transaction in minutes limits the capabilities of large-scale systems, which usually make hundreds of transactions per minute, e.g., systems used in healthcare, retailers, and governmental services. Moreover, distributed systems in WSN and IoT demand flexible algorithms suitable for their limited capabilities [5].

   This research work has introduced the PTA framework and, in particular, its implementation of the protocol PTP, which can achieve consensus among millions of nodes within seconds in addition to being scalable, lightweight, and fully-decentralised. This fast global-agreement can significantly improve the Blockchain technology for Internet-scale applications. Mainly, the protocol PTP and as described in Section 4.4.2 achieves the explicit agreement on each information item (i.e., data block or transaction) through its intrinsic properties. The protocol continuously disseminates new information items into the system using randomised communications. It also makes the transition into phases for each item following the PTA to build the global awareness on the reception of that particular item, i.e., achieving global agreement on the item. Each phase targeting the awareness among system nodes in the protocol PTP is an aggregation process that counts the number of nodes detecting an event locally (e.g. convergence, reception of an item, or verifying a block). Ultimately, after sufficient time the aggregation process in the current phase will converge to the count of nodes in the phase. After the convergence, each node can merely detect achieving the global agreement via verifying its local estimate of count to the system size or to a majority threshold. The system size can be obtained using an epidemic protocol such as SSEP described in Section 4.3.

The PTP can also be adapted to acquire the consistency of information, especially when the total ordering of transactions is required, e.g., attaining certainty on data immutability in the distributed ledger in the Blockchain systems [38]. Each transaction information in the Blockchain platform needs to be applied at most once at each node of the distributed ledger, and any two updates have to be applied in the same order at all ledger nodes. The PTP and via continuous communication among ledger nodes exchange the new transactions, and the agreement process (i.e., the subsequent phases) can acquire the certainty on each transaction. For instance, each node and during the aggregation (i.e., decision-making) process of a phase can either verify the transaction or abort the process due to incorrectness of the transaction or due to violation of transaction order. Upon the detection of convergence, each node can verify its local estimate to a given threshold and decide to validate or invalidate a transaction.

In general, introducing an epidemic consensus service for Blockchain systems requires further research. Precisely, investigations are needed to address the adversary behaviour of some nodes and potential ways to achieve Byzantine agreement in epidemic protocols, e.g., the PTP protocol. Also, the new research work needs to propose a novel epidemic protocol that can verify transactions in a distributed ledger and achieve agreement faster than the cutting-edge consensus protocols, for instance, the Avalanche protocols [126].

2) The protocol ECP is approved to achieve the consensus in stable epidemic systems. The features of protocol can be extended to dynamic systems by integrating the adaptive restart mechanism. In essence, agreement phases in ECP can use the restart mechanism and achieve decentralised decision-making with local results at each node. The mechanism enables the seamless detection of disagreements and withdraws, as nodes which decide not to proceed with the agreement can withhold itself from participating in the process, in addition to detecting the presence of failed nodes. The global tolerance threshold can be used to adjust the required majority for the agreement to make the transition to the next phase or, otherwise, to restart the agreement process.

3) Additionally, work may investigate how running multiple aggregation processes in parallel can help to detect results quality and use the results to compute an estimation

for the lost mass during the aggregation process. The estimation can then be used in the restoration procedure, maybe after the detection of convergence to correct the local estimates to the true target.

4) In general, epidemic protocols have been extensively studied, and many solutions have been proposed. However, and despite the intrinsic features they provide for large and extreme-scale distributed systems, the impact of the protocols in real-world applications and systems is limited. It is recommended that future research should involve potential engagement of the research into realistic applications or projects.

References

[1] G. Coulouris et al. *Distributed Systems: Concepts and Design*. 5th ed. USA: Addison-Wesley Publishing, 2011. ISBN: 0-13-214301-1 (cit. on pp. 1, 5, 31).

[2] F. Schneider et al. 'Understanding Online Social Network Usage from a Network Perspective'. In: *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*. IMC '09. Chicago, Illinois, USA: ACM, 2009, pp. 35–48. ISBN: 978-1-60558-771-4. DOI: 10.1145/1644893.1644899 (cit. on p. 1).

[3] A. Malatras. 'State-of-the-art survey on P2P overlay networks in pervasive computing environments'. In: *Journal of Network and Computer Applications* 55 (2015), pp. 1–23. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2015.04.014 (cit. on p. 1).

[4] P. Costa and J. Leitão. 'Practical Continuous Aggregation in Wireless Edge Environments'. In: *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. Oct. 2018, pp. 41–50. DOI: 10.1109/SRDS.2018.00015 (cit. on pp. 2, 5, 120).

[5] J. Bonneau et al. 'SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies'. In: *2015 IEEE Symposium on Security and Privacy*. May 2015, pp. 104–121. DOI: 10.1109/SP.2015.14 (cit. on pp. 2, 5, 140).

[6] C. Weinstock and J. Goodenough. *On System Scalability*. Tech. rep. CMU/SEI-2006-TN-012. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2006. URL: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=7887 (cit. on p. 2).

[7] J. Albrecht et al. 'Loose Synchronization for Large-scale Networked Systems'. In: *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*. ATEC '06. Boston, MA: USENIX Association, 2006, pp. 28–28 (cit. on p. 2).

[8] D. Peng et al. 'A loosely synchronized gossip-based algorithm for aggregate information computation'. In: *2008 33rd IEEE Conference on Local Computer Networks (LCN)*. Oct. 2008, pp. 451–455. DOI: 10.1109/LCN.2008.4664203 (cit. on pp. 2, 45).

[9] M. J. Fischer. 'The Consensus Problem in Unreliable Distributed Systems (A Brief Survey)'. In: *Proceedings of the 1983 International FCT-Conference on Fundamentals of Computation Theory*. London, UK, UK: Springer, 1983, pp. 127–140. ISBN: 3-540-12689-9. DOI: 10.1007/3-540-12689-9_99 (cit. on pp. 2, 5, 56).

[10] I. Gupta, T. D. Chandra and G. S. Goldszmidt. 'On Scalable and Efficient Distributed Failure Detectors'. In: *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*. PODC '01. Newport, Rhode Island, USA: ACM, 2001, pp. 170–179. ISBN: 1-58113-383-9. DOI: `10.1145/383962.384010` (cit. on p. 2).

[11] M. Jelasity et al. 'Gossip-based Peer Sampling'. In: *ACM Transactions on Computer Systems* 25.3 (2007). ISSN: 0734-2071. DOI: `10.1145/1275517.1275520` (cit. on pp. 2, 20, 26).

[12] P. T. Eugster et al. 'Epidemic Information Dissemination in Distributed Systems'. In: *IEEE Computer Society Press* 37.5 (May 2004), pp. 60–67. ISSN: 0018-9162. DOI: `10.1109/MC.2004.1297243` (cit. on pp. 2, 3, 18, 133).

[13] A. Montresor, M. Jelasity and O. Babaoglu. 'Robust aggregation protocols for large-scale overlay networks'. In: *International Conference on Dependable Systems and Networks, 2004*. June 2004, pp. 19–28. DOI: `10.1109/DSN.2004.1311873` (cit. on pp. 2, 19, 26, 28, 32, 48, 93, 115, 133).

[14] S. Guha and N. Daswani. *An experimental study of the skype peer-to-peer voip system*. Tech. rep. Cornell University, 2005 (cit. on pp. 2, 86).

[15] D. Stutzbach and R. Rejaie. 'Understanding Churn in Peer-to-peer Networks'. In: *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*. IMC '06. Rio de Janeriro, Brazil: ACM, 2006, pp. 189–202. ISBN: 1-59593-561-4. DOI: `10.1145/1177080.1177105` (cit. on pp. 2, 84–86).

[16] S. Voulgaris, D. Gavidia and M. van Steen. 'CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays'. English. In: *Journal of Network and Systems Management* 13.2 (2005), pp. 197–217. ISSN: 1064-7570. DOI: `10.1007/s10922-005-4441-x` (cit. on pp. 2, 26, 101).

[17] M. Jelasity et al. 'The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations'. In: *Middleware 2004*. Ed. by H.-A. Jacobsen. Berlin, Heidelberg: Springer, 2004, pp. 79–98. ISBN: 978-3-540-30229-2. DOI: `10.1007/978-3-540-30229-2_5` (cit. on p. 2).

[18] S. Rhea et al. 'Handling Churn in a DHT'. In: *Proceedings of the Annual Conference on USENIX Annual Technical Conference*. ATEC '04. Boston, MA: USENIX Association, 2004, pp. 10–10. URL: `http://dl.acm.org/citation.cfm?id=1247415.1247425` (cit. on p. 2).

[19] L. Nyers and M. Jelasity. 'Spanning Tree or Gossip for Aggregation: A Comparative Study'. In: *Euro-Par 2014 Parallel Processing: 20th International Conference, Porto, Portugal, August 25-29, 2014. Proceedings*. Ed. by F. Silva, I. Dutra and V. Santos Costa. Cham: Springer, 2014, pp. 379–390. ISBN: 978-3-319-09873-9. DOI: `10.1007/978-3-319-09873-9_32` (cit. on pp. 2, 56).

[20] R. Makhloufi et al. 'Decentralized Aggregation Protocols in Peer-to-Peer Networks: A Survey'. In: *Proceedings of the 4th IEEE International Workshop on Modelling Autonomic Communications Environments*. MACE '09. Venice, Italy: Springer, 2009, pp. 111–116. ISBN: 978-3-642-05005-3. DOI: `10.1007/978-3-642-05006-0_10` (cit. on pp. 2, 56, 83).

[21] K. P. Birman et al. 'Bimodal Multicast'. In: *ACM Transactions on Computer Systems* 17.2 (May 1999), pp. 41–88. ISSN: 0734-2071. DOI: `10.1145/312203.312207` (cit. on p. 3).

[22] D. Shah. 'Gossip Algorithms'. In: *Foundations and Trends in Networking* 3.1 (2009), pp. 1–125. ISSN: 1554-057X. DOI: `10.1561/1300000014` (cit. on pp. 3, 6, 8, 18, 31, 134).

[23] A. Demers et al. 'Epidemic Algorithms for Replicated Database Maintenance'. In: *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*. PODC '87. Vancouver, British Columbia, Canada: ACM, 1987, pp. 1–12. ISBN: 0-89791-239-X. DOI: `10.1145/41840.41841` (cit. on pp. 3, 18, 29, 36, 62, 93).

[24] D. Kempe, J. Kleinberg and A. Demers. 'Spatial Gossip and Resource Location Protocols'. In: *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*. STOC '01. Hersonissos, Greece: ACM, 2001, pp. 163–172. ISBN: 1-58113-349-9. DOI: `10.1145/380752.380796` (cit. on pp. 3, 18).

[25] S. Boyd et al. 'Randomized Gossip Algorithms'. In: *ACM Transactions on Networking* 14.SI (2006), pp. 2508–2530. ISSN: 1063-6692. DOI: `10.1109/TIT.2006.874516` (cit. on pp. 3, 19, 36, 43, 45).

[26] D. Kempe, A. Dobra and J. Gehrke. 'Gossip-based computation of aggregate information'. In: *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings*. Oct. 2003, pp. 482–491. DOI: `10.1109/SFCS.2003.1238221` (cit. on pp. 3, 4, 8, 21, 31, 37–39, 56, 59–61, 68, 72, 78, 83, 87, 88, 101, 109, 113, 115, 117, 120, 133).

[27] M. Jelasity, A. Montresor and O. Babaoglu. 'Gossip-based Aggregation in Large Dynamic Networks'. In: *ACM Transactions on Computer Systems* 23.3 (2005), pp. 219–252. ISSN: 0734-2071. DOI: `10.1145/1082469.1082470` (cit. on pp. 3, 4, 6, 8, 19, 22, 31, 33, 35–37, 48, 58, 61, 68, 83, 87, 89, 113, 116, 117, 119, 120, 127, 133).

[28] P. Jesus, C. Baquero and P. S. Almeida. 'Dependability in Aggregation by Averaging'. In: *The Computing Research Repository (CoRR)* abs/1011.6596 (2010). URL: `http://arxiv.org/abs/1011.6596` (cit. on pp. 3, 4, 20, 22, 32, 37, 133).

[29] I. Rao, A. Harwood and S. Karunasekera. 'Gossip-based asynchronous and robust aggregation protocol - A pessimistic approach'. In: *Consumer Communications and Networking Conference (CCNC), 2011 IEEE*. 2011, pp. 543–548. DOI: `10.1109/CCNC.2011.5766539` (cit. on pp. 3, 22, 32, 48, 49).

[30] F. Blasa et al. 'Symmetric Push-Sum Protocol for decentralised aggregation'. In: *Proceedings of AP2PS 2011, the Third International Conference on Advances in P2P Systems*. IARIA Journals, 2011, pp. 27–32. ISBN: 9781612081731 (cit. on pp. 3, 23, 26, 27, 37, 38, 48, 49, 52, 59–61, 65, 72, 83, 87, 101, 109, 117, 120, 133).

[31] W. R. Heinzelman, J. Kulik and H. Balakrishnan. 'Adaptive Protocols for Information Dissemination in Wireless Sensor Networks'. In: *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*. MobiCom '99. Seattle, Washington, USA: ACM, 1999, pp. 174–185. ISBN: 1-58113-142-9. DOI: `10.1145/313451.313529` (cit. on pp. 3, 35, 36).

[32] J. M. Bahi, S. Contassot-Vivier and R. Couturier. 'An Efficient and Robust Decentralized Algorithm for Detecting the Global Convergence in Asynchronous Iterative Algorithms'. In: *High Performance Computing for Computational Science - VECPAR 2008*. Ed. by J. M. L. M. Palma et al. Berlin, Heidelberg: Springer, 2008, pp. 240–254. ISBN: 978-3-540-92859-1 (cit. on pp. 4, 32, 35, 45, 47, 72, 102, 133).

[33] P. Poonpakdee and G. D. Fatta. 'Robust and efficient membership management in large-scale dynamic networks'. In: *Future Generation Computer Systems* (2017). ISSN: 0167-739X. DOI: `10.1016/j.future.2017.02.033` (cit. on pp. 4, 20, 23, 26, 28, 32, 33, 86, 93, 101, 127, 133).

[34] M. M. Ayiad and G. Di Fatta. 'Robust Epidemic Aggregation Under Churn'. In: *Internet and Distributed Computing Systems*. Ed. by G. Fortino et al. Cham: Springer, 2018, pp. 173–185. ISBN: 978-3-319-97795-9. DOI: `10.1007/978-3-319-97795-9_16` (cit. on pp. 4, 31, 38, 41, 42, 80, 84, 133).

[35] Y. Cao et al. 'An Overview of Recent Progress in the Study of Distributed Multi-Agent Coordination'. In: *IEEE Transactions on Industrial Informatics* 9.1 (Feb. 2013), pp. 427–438. ISSN: 1551-3203. DOI: `10.1109/TII.2012.2219061` (cit. on pp. 4, 68).

[36] V. Rapp and K. Graffi. 'Continuous Gossip-Based Aggregation through Dynamic Information Aging'. In: *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*. July 2013, pp. 1–7. DOI: `10.1109/ICCCN.2013.6614118` (cit. on pp. 4, 115, 120).

[37] S. Abshoff and F. Meyer auf der Heide. 'Continuous Aggregation in Dynamic Ad-Hoc Networks'. In: *Structural Information and Communication Complexity*. Ed. by M. M. Halldórsson. Cham: Springer, 2014, pp. 194–209. ISBN: 978-3-319-09620-9 (cit. on pp. 4, 120).

[38] A. Litke, D. Anagnostopoulos and T. Varvarigou. 'Blockchains for Supply Chain Management: Architectural Elements and Challenges Towards a Global Scale Deployment'. In: *Logistics* 3.1 (2019). ISSN: 2305-6290. DOI: `10.3390/logistics3010005` (cit. on pp. 5, 81, 141).

[39] R. Baldoni et al. 'Unconscious Eventual Consistency with Gossips'. In: *Stabilization, Safety, and Security of Distributed Systems*. Ed. by A. K. Datta and M. Gradinariu. Berlin, Heidelberg: Springer, 2006, pp. 65–81. ISBN: 978-3-540-49823-0. DOI: 10.1007/978-3-540-49823-0_5 (cit. on pp. 5, 29, 36, 45, 68, 93).

[40] P. Bailis and A. Ghodsi. 'Eventual Consistency Today: Limitations, Extensions, and Beyond'. In: *ACM Communications* 56.5 (May 2013), pp. 55–63. ISSN: 0001-0782. DOI: 10.1145/2447976.2447992 (cit. on pp. 5, 8, 30, 35, 134).

[41] N. A. Lynch. *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1996. ISBN: 1558603484 (cit. on p. 5).

[42] G. Weikum and G. Vossen. *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery*. Elsevier, 2001 (cit. on pp. 5, 45, 47, 56, 57).

[43] R. Guerraoui and A. Schiper. 'The generic consensus service'. In: *IEEE Transactions on Software Engineering* 27.1 (Jan. 2001), pp. 29–41. ISSN: 0098-5589. DOI: 10.1109/32.895986 (cit. on pp. 5, 56, 57).

[44] C. Dwork, N. Lynch and L. Stockmeyer. 'Consensus in the Presence of Partial Synchrony'. In: *Journal of the ACM* 35.2 (Apr. 1988), pp. 288–323. ISSN: 0004-5411. DOI: 10.1145/42282.42283 (cit. on pp. 5, 31, 32, 56, 116).

[45] R. Guerraoui et al. 'Consensus in Asynchronous Distributed Systems: A Concise Guided Tour'. In: *Advances in Distributed Systems: Advanced Distributed Computing: From Algorithms to Systems*. Ed. by S. Krakowiak and S. Shrivastava. Berlin, Heidelberg: Springer, 2000, pp. 33–47. ISBN: 978-3-540-46475-4. DOI: 10.1007/3-540-46475-1_2 (cit. on pp. 5, 6, 45, 47).

[46] B. S. Chlebus and D. R. Kowalski. 'Gossiping to Reach Consensus'. In: *Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures*. SPAA '02. Winnipeg, Manitoba, Canada: ACM, 2002, pp. 220–229. ISBN: 1-58113-529-7. DOI: 10.1145/564870.564908 (cit. on pp. 5, 6, 29, 56, 134).

[47] T. C. Aysal et al. 'Broadcast Gossip Algorithms for Consensus'. In: *IEEE Transactions on Signal Processing* 57.7 (July 2009), pp. 2748–2761. ISSN: 1053-587X. DOI: 10.1109/TSP.2009.2016247 (cit. on pp. 5, 20).

[48] A. Olshevsky and J. N. Tsitsiklis. 'Convergence Speed in Distributed Consensus and Averaging'. In: *SIAM Journal on Control and Optimization (SICON)* 48.1 (Feb. 2009), pp. 33–55. ISSN: 0363-0129. DOI: 10.1137/060678324 (cit. on pp. 5, 6, 8, 29, 37, 56, 68, 83, 134).

[49] B. Charron-Bost, M. Függer and T. Nowak. 'Approximate Consensus in Highly Dynamic Networks: The Role of Averaging Algorithms'. In: *Automata, Languages, and Programming*. Ed. by M. M. Halldórsson et al. Berlin, Heidelberg: Springer, 2015, pp. 528–539. ISBN: 978-3-662-47666-6. DOI: 10.1007/978-3-662-47666-6_42 (cit. on pp. 5, 6, 30, 36, 56).

[50] C. Georgiou et al. 'Asynchronous Gossip'. In: *Journal of the ACM* 60.2 (2013), pp. 1–42. ISSN: 0004-5411. DOI: 10.1145/2450142.2450147 (cit. on pp. 7, 8, 32, 37).

[51] J. M. Bahi et al. 'A decentralized convergence detection algorithm for asynchronous parallel iterative algorithms'. In: *IEEE Transactions on Parallel and Distributed Systems* 16.1 (Jan. 2005), pp. 4–13. ISSN: 1045-9219. DOI: 10.1109/TPDS.2005.2 (cit. on pp. 7, 35, 37–40, 43, 45–47, 134).

[52] M. Ayiad, A. Katti and G. Di Fatta. 'Agreement in Epidemic Information Dissemination'. In: *Internet and Distributed Computing Systems: 9th International Conference, IDCS 2016, Wuhan, China, September 28-30, 2016, Proceedings*. Ed. by W. Li et al. Vol. 9864. Cham: Springer, 2016. Chap. 9, pp. 95–106. ISBN: 978-3-319-45940-0. DOI: 10.1007/978-3-319-45940-0_9 (cit. on pp. 7, 35, 36, 43, 45, 49, 55, 65, 83, 98, 121, 134).

[53] A. Montresor and M. Jelasity. 'PeerSim: A scalable P2P simulator'. In: *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*. Sept. 2009, pp. 99–100. DOI: 10.1109/P2P.2009.5284506 (cit. on pp. 11, 19).

[54] M. Ebrahim, S. Khan and H. Mohani. 'Peer-to-Peer Network Simulators: an Analytical Review'. In: *arXiv e-prints* (May 2014). arXiv: 1405.0400 [cs.NI] (cit. on p. 11).

[55] S. Naicken et al. 'The State of Peer-to-peer Simulators and Simulations'. In: *ACM SIGCOMM Computer Communication Review* 37.2 (Mar. 2007), pp. 95–98. ISSN: 0146-4833. DOI: 10.1145/1232919.1232932 (cit. on p. 11).

[56] R. Karp et al. 'Randomized rumor spreading'. In: *Proceedings 41st Annual Symposium on Foundations of Computer Science*. 2000, pp. 565–574. DOI: 10.1109/SFCS.2000.892324 (cit. on pp. 18, 29, 32, 36, 62).

[57] P. T. Eugster et al. 'Lightweight Probabilistic Broadcast'. In: *ACM Transactions and Computer Systems* 21.4 (Nov. 2003), pp. 341–374. ISSN: 0734-2071. DOI: 10.1145/945506.945507 (cit. on pp. 18, 62).

[58] M. Jelasity and A. Montresor. 'Epidemic-style proactive aggregation in large overlay networks'. In: *Distributed Computing Systems, 2004. Proceedings*. 2004, pp. 102–109. DOI: 10.1109/ICDCS.2004.1281573 (cit. on pp. 19, 22, 37, 56, 68).

[59] M. Jelasity. 'Gossip'. In: *Self-organising Software: From Natural to Artificial Adaptation*. Ed. by G. Di Marzo Serugendo, M.-P. Gleizes and A. Karageorgos. Berlin, Heidelberg: Springer, 2011, pp. 139–162. ISBN: 978-3-642-17348-6. DOI: 10.1007/978-3-642-17348-6_7 (cit. on pp. 19, 48).

[60] M. Jelasity, A. Montresor and O. Babaoglu. ‘Detection and removal of malicious peers in gossip-based protocols’. In: *In Proceedings of FuDiCo II: Workshop on Future Directions in Distributed Computing* (2004) (cit. on pp. 19, 33).

[61] A. Montresor and A. Ghodsi. ‘Towards robust peer counting’. In: *Peer-to-Peer Computing, 2009.* 2009, pp. 143–146. DOI: 10.1109/P2P.2009.5284525 (cit. on p. 19).

[62] S. Madden et al. ‘TAG: A Tiny AGgregation Service for Ad-hoc Sensor Networks’. In: *SIGOPS - Operating Systems Review* 36.SI (Dec. 2002), pp. 131–146. ISSN: 0163-5980. DOI: 10.1145/844128.844142 (cit. on p. 19).

[63] M. Dam and R. Stadler. ‘A generic protocol for network state aggregation’. In: 3 (2005), p. 411 (cit. on p. 19).

[64] K. Iwanicki, M. van Steen and S. Voulgaris. ‘Gossip-Based Clock Synchronization for Large Decentralized Systems’. In: *Self-Managed Networks, Systems, and Services.* Ed. by A. Keller and J.-P. Martin-Flatin. Berlin, Heidelberg: Springer, 2006, pp. 28–42. ISBN: 978-3-540-34740-8 (cit. on p. 19).

[65] B. Ghit, F. Pop and V. Cristea. ‘Epidemic-Style Global Load Monitoring in Large-Scale Overlay Networks’. In: *2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing.* Nov. 2010, pp. 393–398. DOI: 10.1109/3PGCIC.2010.62 (cit. on p. 19).

[66] G. D. Fatta et al. ‘Fault tolerant decentralised K-Means clustering for asynchronous large-scale networks’. In: *Journal of Parallel and Distributed Computing* 73.3 (2013). Models and Algorithms for High-Performance Distributed Data Mining, pp. 317–329. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2012.09.009 (cit. on pp. 20, 68, 83).

[67] R. Gupta and Y. Singh. ‘Reputation Aggregation in Peer-to-Peer Network Using Differential Gossip Algorithm’. In: *Knowledge and Data Engineering, IEEE Transactions on* 27.10 (2015), pp. 2812–2823. ISSN: 1041-4347. DOI: 10.1109/TKDE.2015.2427793 (cit. on p. 20).

[68] A. Katti et al. ‘Epidemic failure detection and consensus for extreme parallelism’. In: *The International Journal of High Performance Computing Applications* 0.0 (2017), p. 1094342017690910. DOI: 10.1177/1094342017690910. eprint: http://dx.doi.org/10.1177/1094342017690910 (cit. on pp. 20, 23, 55, 59, 68, 83, 93).

[69] M. Jelasity, A. Montresor and O. Babaoglu. ‘T-Man: Gossip-based Fast Overlay Topology Construction’. In: *Computer Networks* 53.13 (Aug. 2009), pp. 2321–2339. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2009.03.013 (cit. on p. 20).

[70] R. Azimi and H. Sajedi. ‘Peer sampling gossip-based distributed clustering algorithm for unstructured P2P networks’. In: *Neural Computing and Applications* (July 2017). ISSN: 1433-3058. DOI: 10.1007/s00521-017-3119-0 (cit. on p. 20).

[71] M. Bawa et al. *Estimating Aggregates on a Peer-to-Peer Network*. Technical Report 2003-24. Stanford InfoLab, 2003 (cit. on p. 20).

[72] H.-G. Roh and C. L. Ignat. *Rapid and Round-free Multi-pair Asynchronous Push-Pull Aggregation*. Research Report RR-8044. INRIA, 2012. URL: `https://hal.inria.fr/hal-00724232` (cit. on pp. 22, 48, 49, 115, 120).

[73] P. Jesus, C. Baquero and P. S. Almeida. 'Fault-Tolerant Aggregation by Flow Updating'. In: *Proceedings of the 9th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems*. DAIS '09. Lisbon, Portugal: Springer, 2009, pp. 73–86. ISBN: 978-3-642-02163-3. DOI: `10.1007/978-3-642-02164-0_6` (cit. on p. 24).

[74] P. Jesus, C. Baquero and P. Almeida. 'Fault-Tolerant Aggregation for Dynamic Networks'. In: *Reliable Distributed Systems, 2010 29th IEEE Symposium on*. 2010, pp. 37–43. DOI: `10.1109/SRDS.2010.13` (cit. on p. 24).

[75] P. Jesus, C. Baquero and P. S. Almeida. 'Flow updating: Fault-tolerant aggregation for dynamic networks'. In: *Journal of Parallel and Distributed Computing* 78 (2015), pp. 53–64. ISSN: 0743-7315. DOI: `10.1016/j.jpdc.2015.02.003` (cit. on pp. 26, 93, 115, 120).

[76] E. Ogston and S. A. Jarvis. 'Peer sampling with improved accuracy'. In: *Peer-to-Peer Networking and Applications* 2.1 (6th Nov. 2008), p. 24. ISSN: 1936-6450. DOI: `10.1007/s12083-008-0017-3` (cit. on pp. 26, 93).

[77] C. Cyrus Moallemi and B. Van Roy. 'Consensus Propagation'. In: *IEEE Transactions on Information Theory* 52 (Nov. 2006), pp. 4753–4766. DOI: `10.1109/TIT.2006.883539` (cit. on p. 29).

[78] W. Ren, R. W. Beard and E. M. Atkins. 'Information consensus in multivehicle cooperative control'. In: *IEEE Control Systems* 27.2 (Apr. 2007), pp. 71–82. ISSN: 1066-033X. DOI: `10.1109/MCS.2007.338264` (cit. on pp. 29, 68).

[79] F. Fagnani and S. Zampieri. 'Randomized consensus algorithms over large scale networks'. In: *IEEE Journal on Selected Areas in Communications* 26.4 (May 2008), pp. 634–649. ISSN: 0733-8716. DOI: `10.1109/JSAC.2008.080506` (cit. on p. 29).

[80] D. Dolev et al. 'Reaching Approximate Agreement in the Presence of Faults'. In: *J. ACM* 33.3 (May 1986), pp. 499–516. ISSN: 0004-5411. DOI: `10.1145/5925.5931`. URL: `http://doi.acm.org/10.1145/5925.5931` (cit. on p. 30).

[81] A. Katti and D. J. Lilja. 'Efficient and Fast Approximate Consensus with Epidemic Failure Detection at Extreme Scale'. In: *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. Mar. 2018, pp. 267–272. DOI: `10.1109/PDP2018.2018.00045` (cit. on p. 30).

[82] M. J. Fischer, N. A. Lynch and M. S. Paterson. 'Impossibility of Distributed Consensus with One Faulty Process'. In: *Journal of the ACM* 32.2 (Apr. 1985), pp. 374–382. ISSN: 0004-5411. DOI: 10.1145/3149.214121 (cit. on pp. 31, 36).

[83] R. Guerraoui and A. Schiper. 'Consensus: the big misunderstanding [distributed fault tolerant systems]'. In: *Distributed Computing Systems, 1997., Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of.* Oct. 1997, pp. 183–188. DOI: 10.1109/FTDCS.1997.644722 (cit. on p. 31).

[84] D. Dolev, C. Dwork and L. Stockmeyer. 'On the Minimal Synchronism Needed for Distributed Consensus'. In: *Journal of the ACM* 34.1 (Jan. 1987), pp. 77–97. ISSN: 0004-5411. DOI: 10.1145/7531.7533 (cit. on p. 31).

[85] M. Ben-Or. 'Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols'. In: *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing.* PODC '83. Montreal, Quebec, Canada: ACM, 1983, pp. 27–30. ISBN: 0-89791-110-5. DOI: 10.1145/800221.806707 (cit. on p. 31).

[86] L. Lamport. *Generalized Consensus and Paxos.* Tech. rep. MSR-TR-2005-33. Microsoft Research, Mar. 2005, p. 60. URL: https://www.microsoft.com/en-us/research/publication/generalized-consensus-and-paxos/ (cit. on pp. 31, 57, 58, 116).

[87] T. D. Chandra and S. Toueg. 'Unreliable Failure Detectors for Reliable Distributed Systems'. In: *Journal of the ACM* 43.2 (1996), pp. 225–267. ISSN: 0004-5411. DOI: 10.1145/226643.226647 (cit. on p. 31).

[88] R. Van Renesse, Y. Minsky and M. Hayden. 'A Gossip-Style Failure Detection Service'. In: *Proceeding Middleware '98 Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing.* Ithaca, NY, USA: Cornell University, 1998, pp. 55–70 (cit. on p. 31).

[89] M. Pease, R. Shostak and L. Lamport. 'Reaching Agreement in the Presence of Faults'. In: *Journal of the ACM* 27.2 (Apr. 1980), pp. 228–234. ISSN: 0004-5411. DOI: 10.1145/322186.322188 (cit. on pp. 31, 116).

[90] C. Georgiou et al. 'On the Complexity of Asynchronous Gossip'. In: *Proceedings of the 27th ACM Symposium on Principles of Distributed Computing.* PODC '08. Toronto, Canada: ACM, 2008, pp. 135–144. ISBN: 978-1-59593-989-0. DOI: 10.1145/1400751.1400771 (cit. on pp. 31, 116).

[91] P. Poonpakdee and G. Di Fatta. 'Expander Graph Quality Optimisation in Randomised Communication'. In: *Data Mining Workshop (ICDMW).* Dec. 2014, pp. 597–604. DOI: 10.1109/ICDMW.2014.150 (cit. on p. 32).

[92] M. M. Ayiad and G. D. Fatta. 'Agreement in Epidemic Data Aggregation'. In: *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS).* Dec. 2017, pp. 738–746. DOI: 10.1109/ICPADS.2017.00099 (cit. on pp. 32, 35, 37, 39, 40, 43, 45, 49, 55, 83, 88, 98, 102, 116).

[93] D. P. Bertsekas and J. N. Tsitsiklis. ‘Convergence Rate and Termination of Asynchronous Iterative Algorithms’. In: *Proceedings of the 3rd International Conference on Supercomputing*. ICS ’89. Crete, Greece: ACM, 1989, pp. 461–470. ISBN: 0-89791-309-4. DOI: 10.1145/318789.318894 (cit. on pp. 35, 45).

[94] P. Poonpakdee, N. Orhon and G. Di Fatta. ‘Convergence Detection in Epidemic Aggregation’. English. In: *Euro-Par 2013: Parallel Processing Workshops*. Ed. by D. an Mey et al. Vol. 8374. Lecture Notes in Computer Science. Springer, 2014, pp. 292–300. ISBN: 978-3-642-54419-4. DOI: 10.1007/978-3-642-54420-0_29 (cit. on pp. 35, 39, 40, 43, 46, 70, 95, 102, 116).

[95] M. Femminella et al. ‘Gossip-based signaling dissemination extension for next steps in signaling’. In: *2012 IEEE Network Operations and Management Symposium*. Apr. 2012, pp. 1022–1028. DOI: 10.1109/NOMS.2012.6212024 (cit. on p. 36).

[96] N. T. H. Linh et al. ‘Convergence time evaluation of a gossip algorithm over signed graphs’. In: *Control Conference (ASCC), 2015 10th Asian*. 2015, pp. 1–5. DOI: 10.1109/ASCC.2015.7244534 (cit. on pp. 43, 45).

[97] G. Shi et al. ‘Finite-Time Convergent Gossiping’. In: *ACM Transactions on Networking* 24.5 (Oct. 2016), pp. 2782–2794. ISSN: 1063-6692. DOI: 10.1109/TNET.2015.2484345 (cit. on p. 45).

[98] F. M. D. Fave et al. ‘Decentralised Coordination of Unmanned Aerial Vehicles for Target Search using the Max-Sum Algorithm’. Event Dates: 10 - 14 May. May 2010. URL: https://eprints.soton.ac.uk/270812/ (cit. on p. 45).

[99] D. J. Buehrer and C.-Y. Wang. ‘Deco: A Decentralized, Cooperative Atomic Commit Protocol’. In: *Journal of Computer Networks and Communications* 2012 (2012) (cit. on pp. 45, 57).

[100] L. Tang et al. ‘Empirical Study on the Evolution of PlanetLab’. In: *Sixth International Conference on Networking, 2007. ICN ’07*. 2007, pp. 64–64. DOI: 10.1109/ICN.2007.40 (cit. on p. 50).

[101] J. Peltotalo et al. ‘Peer-to-Peer Streaming Technology Survey’. In: *Seventh International Conference on Networking (icn 2008)*. Apr. 2008, pp. 342–350. DOI: 10.1109/ICN.2008.86 (cit. on p. 50).

[102] D. Lee et al. ‘Has Internet Delay Gotten Better or Worse?’ In: *Proceedings of the 5th International Conference on Future Internet Technologies*. CFI ’10. Seoul, Korea: ACM, 2010, pp. 51–54. ISBN: 978-1-4503-0230-2. DOI: 10.1145/1853079.1853094 (cit. on p. 50).

[103] S. Zander and G. Armitage. ‘Minimally-intrusive frequent round trip time measurements using Synthetic Packet-Pairs’. In: *38th Annual IEEE Conference on Local Computer Networks*. Oct. 2013, pp. 264–267. DOI: 10.1109/LCN.2013.6761245 (cit. on p. 50).

[104] G. Alexander and J. R. Crandall. ‘Off-path round trip time measurement via TCP/IP side channels’. In: *2015 IEEE Conference on Computer Communications (INFOCOM)*. Apr. 2015, pp. 1589–1597. DOI: 10.1109/INFOCOM.2015.7218538 (cit. on p. 50).

[105] P. Romirer-Maierhofer et al. ‘Network-Wide Measurements of TCP RTT in 3G’. In: *Traffic Monitoring and Analysis*. Ed. by M. Papadopouli, P. Owezarski and A. Pras. Berlin, Heidelberg: Springer, 2009, pp. 17–25. ISBN: 978-3-642-01645-5. DOI: 10.1007/978-3-642-01645-5_3 (cit. on p. 50).

[106] B. K. Soorty et al. ‘Performance Comparison of Category 5e vs. Category 6 Cabling Systems for both IPv4 and IPv6 in Gigabit Ethernet’. In: *2010 10th IEEE International Conference on Computer and Information Technology*. June 2010, pp. 1525–1529. DOI: 10.1109/CIT.2010.271 (cit. on p. 50).

[107] C. P. Kruger and G. P. Hancke. ‘Implementing the Internet of Things vision in industrial wireless sensor networks’. In: *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*. July 2014, pp. 627–632. DOI: 10.1109/INDIN.2014.6945586 (cit. on p. 50).

[108] C. Pei et al. ‘WiFi can be the weakest link of round trip network latency in the wild’. In: *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. Apr. 2016, pp. 1–9. DOI: 10.1109/INFOCOM.2016.7524396 (cit. on p. 50).

[109] J. Y. Yu and M. Rabbat. ‘Performance comparison of randomized gossip, broadcast gossip and collection tree protocol for distributed averaging’. In: *2013 5th IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*. Dec. 2013, pp. 93–96. DOI: 10.1109/CAMSAP.2013.6714015 (cit. on pp. 56, 58, 73).

[110] W. Ren, R. W. Beard and E. M. Atkins. ‘A survey of consensus problems in multi-agent coordination’. In: *Proceedings of the 2005, American Control Conference, 2005*. June 2005, pp. 1859–18643. DOI: 10.1109/ACC.2005.1470239 (cit. on p. 56).

[111] R. Van Renesse and D. Altinbuken. ‘Paxos Made Moderately Complex’. In: *ACM Computing Surveys* 47.3 (Feb. 2015), 42:1–42:36. ISSN: 0360-0300. DOI: 10.1145/2673577 (cit. on pp. 57, 58).

[112] L. Lamport. ‘The Part-time Parliament’. In: *ACM Transactions on Computer Systems* 16.2 (May 1998), pp. 133–169. ISSN: 0734-2071. DOI: 10.1145/279227.279229 (cit. on p. 58).

[113] L. Chitnis, A. Dobra and S. Ranka. ‘Aggregation Methods for Large-scale Sensor Networks’. In: *ACM Transactions on Networking* 4.2 (Apr. 2008), 9:1–9:36. ISSN: 1550-4859. DOI: 10.1145/1340771.1340775 (cit. on p. 68).

[114] J. Hursey et al. 'A Log-Scaling Fault Tolerant Agreement Algorithm for a Fault Tolerant MPI'. In: *Recent Advances in the Message Passing Interface: 18th European MPI Users' Group Meeting, EuroMPI 2011, Santorini, Greece, September 18-21, 2011. Proceedings*. Ed. by Y. Cotronis et al. Berlin, Heidelberg: Springer, 2011, pp. 255–263. ISBN: 978-3-642-24449-0. DOI: 10.1007/978-3-642-24449-0_29 (cit. on p. 73).

[115] D. Kostoulas et al. 'Decentralized Schemes for Size Estimation in Large and Dynamic Groups'. In: *Fourth IEEE International Symposium on Network Computing and Applications*. July 2005, pp. 41–48. DOI: 10.1109/NCA.2005.15 (cit. on pp. 83, 119, 127, 133).

[116] R. Bhagwan, S. Savage and G. M. Voelker. 'Understanding Availability'. In: *Peer-to-Peer Systems II*. Ed. by M. F. Kaashoek and I. Stoica. Berlin, Heidelberg: Springer, 2003, pp. 256–267. ISBN: 978-3-540-45172-3 (cit. on pp. 84, 85).

[117] Z. Yao. 'Understanding Churn in Decentralized Peer-to-Peer Networks'. PhD thesis. Texas A&M University, Aug. 2009. URL: http://hdl.handle.net/1969.1/ETD-TAMU-2009-08-906 (cit. on p. 84).

[118] F. Benevenuto et al. 'Characterizing User Behavior in Online Social Networks'. In: *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*. IMC '09. Chicago, Illinois, USA: ACM, 2009, pp. 49–62. ISBN: 978-1-60558-771-4. DOI: 10.1145/1644893.1644900 (cit. on pp. 85, 86).

[119] S. Saroiu, K. P. Gummadi and S. D. Gribble. 'Measuring and analyzing the characteristics of Napster and Gnutella hosts'. In: *Multimedia Systems* 9.2 (1st Aug. 2003), pp. 170–184. ISSN: 1432-1882. DOI: 10.1007/s00530-003-0088-1 (cit. on p. 85).

[120] B. Javadi et al. 'Discovering Statistical Models of Availability in Large Distributed Systems: An Empirical Study of SETI@home'. In: *IEEE Transactions on Parallel and Distributed Systems* 22.11 (Nov. 2011), pp. 1896–1903. ISSN: 1045-9219. DOI: 10.1109/TPDS.2011.50 (cit. on p. 85).

[121] H. Pham. 'System Reliability Concepts'. In: *System Software Reliability*. London: Springer, 2006, pp. 9–75. ISBN: 978-1-84628-295-9. DOI: 10.1007/1-84628-295-0_2 (cit. on p. 85).

[122] N. Basher et al. 'A Comparative Analysis of Web and Peer-to-peer Traffic'. In: *Proceedings of the 17th International Conference on World Wide Web*. WWW '08. Beijing, China: ACM, 2008, pp. 287–296. ISBN: 978-1-60558-085-2. DOI: 10.1145/1367497.1367537 (cit. on p. 86).

[123] M. Conti et al. 'Trusted Dynamic Storage for Dunbar-Based P2P Online Social Networks'. In: *On the Move to Meaningful Internet Systems: OTM 2014 Conferences*. Ed. by R. Meersman et al. Berlin, Heidelberg: Springer, 2014, pp. 400–417. ISBN: 978-3-662-45563-0. DOI: 10.1007/978-3-662-45563-0_23 (cit. on p. 86).

[124] R. Kumar and A. Tomkins. ‘A Characterization of Online Browsing Behavior’. In: *Proceedings of the 19th International Conference on World Wide Web*. WWW ’10. Raleigh, North Carolina, USA: ACM, 2010, pp. 561–570. ISBN: 978-1-60558-799-8. DOI: `10.1145/1772690.1772748` (cit. on p. 86).

[125] I. Rao, A. Harwood and S. Karunasekera. ‘Impacts of Asynchrony on Epidemic-Style Aggregation Protocols’. In: *Parallel and Distributed Systems (ICPADS)*. 2010, pp. 601–608. DOI: `10.1109/ICPADS.2010.130` (cit. on p. 101).

[126] M. Yin et al. ‘Scalable and Probabilistic Leaderless BFT Consensus through Metastability’. In: *Computing Research Repository* abs/1906.08936 (2019). Team Rocket. arXiv: `1906.08936`. URL: `http://arxiv.org/abs/1906.08936` (cit. on p. 141).