Adaptability in Dynamic Wireless Networks

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op donderdag 15 november 2012 om 10:00 uur

door **Venkatraman Ganeshan IYER**
Master of Science, Computer Science
geboren te Chennai, India

Dit proefschrift is goedgekeurd door de promotor:
Prof. Dr. K.G. Langendoen

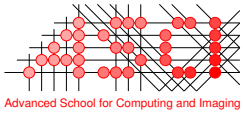
samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof. Dr. K.G. Langendoen	Technische Universiteit Delft, promotor
Dr. S.O. Dulman	Technische Universiteit Delft, copromotor
Prof. Dr. Ir. M. van Steen	Vrije Universiteit Amsterdam
Prof. Dr. Ir. T. Basten	Technische Universiteit Eindhoven
Prof. Dr. Ir. D.H.J. Epema	Technische Universiteit Delft, Technische Universiteit Eindhoven
Prof. Dr. K.U. Römer	Universität zu Lübeck, Germany
Ing. F. van der Wateren	CHESS, Haarlem
Prof. Dr. C. Witteveen	Technische Universiteit Delft, reservelid

ISBN: 978-94-6186-067-5

Printed in the Netherlands by: Ipskamp Drukkers Print Service.

This work was funded by the ALWEN project, a Point-One project funded by SenterNovem.



This work was carried out in the ASCI graduate school.

It is, of course, a trifle, but there is nothing so important as trifles.

– Sherlock Holmes, in *The Man with the Twisted Lip*

Acknowledgements

When you have committed yourself to a process of learning that stretches your thinking and takes you beyond your comfort zone, it is then that you understand and cherish the support that you receive from people, both close and far. I can firmly say that my progress towards the completion of my PhD thesis stands testimony to this observation. While words would barely suffice to express my gratitude, I would very much like to acknowledge the inner circle of people who have cheered me on to the finishing line through all these years.

Firstly, I would like to thank Koen Langendoen - my promotor and supervisor, for offering me the opportunity to conduct research as a PhD candidate. His experimental, get-your-hands-dirty-with-details outlook to research has always been something that I admire and incorporate in my work as often as possible. You have played a major role in helping me make the transition from a student to a researcher. This transition did not come about in a day, and I can never thank you enough for being observant, patient and often understanding in the past four years. I would also like to thank my co-promotor, Stefan Dulman, for agreeing to supervise my research at a crucial stage of my PhD study. I learnt a lot under your supervision, especially with several ideas we bounced off each other at weekly reading sessions, or even a cup of coffee. It has been a truly wonderful experience working with you both, and I hope to keep in touch, if not collaborate, in the years to come. Let me add in a word of thanks also to the committee members, for spending their time over reading the thesis draft, and providing constructive feedback.

I would also like to thank the members of the ALwEN project for their support, suggestions and feedback about my research from time to time. A special thanks to Frits van der Wateren, who assisted me early on with my experimental efforts for the PhD track. In no particular order, I would like to acknowledge Milos Blagojevic, Majid Nabi, Edwin Rijpkema, Teun Hendricks, Daniela Gavidia, Freek Van Polen, and Matthew Dobson for their opinions and comments on several discussions and presentations we have had.

Among colleagues here at Delft, I would like to thank Matthias Woehrle for his close supervision during the initial years of my PhD research. It was a truly remarkable experience working with you. I owe a lot of my success to the endless technical discussions we have had, which often ended with me going back to the drawing board. You have also been a wonderful host, and an excellent comrade on a hiking trip. I would also like to acknowledge my office roommates Alexander Feldman, Andrei Pruteanu and Niels Brouwers for their support and understanding in these four years. I thoroughly enjoyed our frequent discussions regarding research, and otherwise, the wide world around us. I would also like to thank my colleague Andreas Loukas for being patient and critical of new ideas or suggestions that I would come up with. Again, in no

particular order, thanks to Otto Visser, Philipp Glatz, Kavitha Muthukrishnan, Gertjan Halkes, Marco Cattani, Qingzhi Liu, and Martin Bor for being such awesome colleagues.

I would like to thank my friends Shreyas Raghunathan, Venkatesh Seshan, Gunjan Singh, Ashutosh Shembekar, Anoop Jassal, Krishna Kowlgi, Murthy Dharmapura, and others for being understanding of my rather reclusive habit at times, and providing the much needed company and support when I felt I had to unwind. A special thanks to my cousins, Venkatraman and Chidambaresan, for their presence and encouragement. I wish you both success with your study and research, and am sure that you will continue the good work you are currently doing.

Lastly, I would like to thank my family, namely my parents, brother, sister-in-law and relatives for being supportive throughout these years. Thank you very much for providing the voice of reason and wisdom when I needed them the most. More importantly, thanks for standing by with me through times both joyous and difficult. Your support has definitely kept me going on about my work tirelessly, and with renewed vigor. A special acknowledgement to late K. G. Balakrishnan, for encouraging me to come this far.
Thank you all.

Venkatraman Iyer
September 15, 2012
Delft, The Netherlands

Contents

Chapter 1

Introduction

The proliferation of applications involving networked embedded devices has rapidly increased over the last decade. The application domain for wireless embedded devices has significantly outgrown its beginnings in military and defence scenarios. For example, wireless sensor networks (WSNs) are deployed in both indoor and outdoor settings for monitoring relevant phenomena such as temperature, humidity, and human movement patterns. Networked embedded devices have also been employed in mobility-based scenarios such as intelligent transport systems and crowd management. More often, the networked wireless applications on these devices are expected to run for a prolonged period without manual intervention. This implies that the embedded software must create and sustain the network topology in an ad-hoc manner, i. e., without the help of external infrastructure. Additionally, the software running on these devices faces challenges from dynamics such as changing environmental phenomena affecting wireless link quality, and changing topologies caused by node mobility. This demands self-adjusting and adaptive mechanisms on part of the embedded software.

In this thesis, we investigate adaptability in dynamic wireless networks characterized by intermittent wireless links and node mobility. To do so, we identify specific issues concerning communication reliability in the face of changing wireless neighborhoods and link failures, and design decentralized mechanisms that adapt to topology dynamics at runtime. Using our research works as a basis, we observe the commonalities in our solutions, and remark on the general approach to be favored in order to achieve decentralized adaptability.

1.1 Application Scenarios for Wireless Networking

The research efforts presented in this thesis are funded by the ALWEN project, a Point-One project funded by AgentschapNL [1]. The ALWEN project focused mainly on the research challenges that included monitoring the daily activities of elderly patients, detecting events of interest such as elevated rate of body signals, and providing timely delivery of patient data to medical personnel. The challenges together address what is commonly known as *Assisted Daily Living*, a system wherein elderly people are monitored by embedded devices that form a wireless network. The contributions in this thesis, however, apply also to other classes of applications involving wireless networks. In the following, we present relevant application scenarios for wireless embedded networked devices:

Environmental Monitoring - Wireless nodes are deployed in a region of interest to continuously monitor relevant phenomena and report deviations from normal observations. Typical examples include monitoring of volcanic and seismic activity [142], temperature and humidity in an agricultural field [68], and geophysical data in high-altitude environments [6]. These applications are implemented by initially deploying the wireless nodes in the environment (manually, in most cases). Once deployed, the nodes connect to a central basestation either directly or via multiple wireless hops.

Assisted Daily Living - With an increase in the number of elders suffering from chronic ailments, it becomes unfeasible to manually track the progress of every patient. As an alternative, researchers and medical personnel have investigated the benefits of monitoring patients using a network of embedded sensors [118, 141, 145]. The so-called Body Sensor Network not only monitors the vital signs of a patient in real-time, but also communicates with ambient devices to maintain a record of the daily activities [89, 97]. The application is typically designed to notify medical emergencies, and provide constructive feedback to patients.

Asset Tracking - Wireless systems are increasingly used to monitoring inventory items in a warehouse for incidences of theft. Typically, the network infrastructure comprises a central *reader* node that co-ordinates with a set of *tags* that are attached to the monitored objects, as in [96, 125, 147]. Surveillance is achieved with the reader node keeping track of the number of tags within its communication range. Additionally, inventory items may also be fitted with movable components as in the case of the *Kiva robots* [28], that make it easier to transport them from one shelf to the other.

Intelligent Transport Systems - Vehicular networks are seen as a promising technology to assist the implementation of intelligent transport systems [20, 85]. Every vehicle is equipped with a wireless telemetry device, which can then be used to both signal and resolve traffic congestion in real-time. Typical research challenges in vehicular networks are related to minimizing the so-called *age of information* that is disseminated throughout the network [56].

Studying Human Interaction in Large Groups - Recent advances in wearable computing [60] have made it possible to study the interaction of humans in large social groups. Typically, groups of people volunteer to participate in a so-called social game, in which every participant is equipped with a wireless device. The communication encounters between device pairs are periodically logged over an extended period, and the traces are mined to extract social interaction patterns. For example, the MIT reality mining project [26] collected data traces from Bluetooth radio encounters and e-mail exchanges between human participants for over a year. Based on experimental findings, the work by [108] has identified the presence of so-called honest signals that could be used to predict and influence the outcomes of day-to-day interactions between people [106].

While application scenarios for networked embedded devices are diverse, the embedded software that runs them perform mostly similar tasks. We next describe the important components that form an essential part of embedded software for wireless network applications.

1.2 Building Blocks for Networked Embedded Software

Throughout this thesis, we focus on the embedded software for networked wireless systems. We start in this section by identifying fundamental building blocks that make networked embedded software, and describing research challenges related to their functioning that demand run-time adaptivity. A wireless application for networked embedded systems is structured as a set of *components*, implemented as software modules. A software module may address a set of specific application requirements, or several modules could collectively address a single requirement. For example, a wireless sensor network application for environmental monitoring may impose several requirements, such as *(i)* the need for a tree structure rooted at the basestation for gathering data, *(ii)* time synchronization for co-ordinating sensing and communication between neighboring sensor nodes, *(iii)* aggregation protocols that filter out relevant context from raw sensor samples, and *(iv)* dissemination protocols for performing code updates and explicit control signalling. In the following, we list four main important components that mainly involve node-to-node communication. We provide a brief description on how they work, the scenarios to which they apply, and the research challenges that they bring along.

Dissemination Message dissemination is used mainly for performing software updates in wireless networks, and explicit control signalling to kick-start a specific process such as gathering data. Typical examples where dissemination is used include *(i)* WSN applications, wherein software executables and routing tables are regularly updated over time, and *(ii)* Vehicular Networks, in which events such as road congestion and traffic blocks are disseminated along the line of vehicles on a highway for safety reasons. The underlying premise of dissemination is the existence of at least a single originating node that wishes to replicate a context of interest; i. e., either a software update, an interesting event, or simply a trigger for co-ordinated activity. Using the broadcast nature of wireless communication, the context is propagated throughout the network with every node relaying the message that it receives. Since it is naturally desired to disseminate information as quickly as possible, ensuring efficiency in communication is of paramount importance. In its simplest implementation, dissemination takes the form of a flooding mechanism, in which every node continuously transmits the information it receives. A downside to the flooding approach is the build up of the so-called *broadcast storm* problem [102], in which neighboring nodes compete inefficiently for channel access. To mitigate this problem, efforts have been made to limit the number of communication hops [137], adaptively limit the message transmission rate [77], and to transmit messages in a probabilistic manner [32]. As a first step towards achieving efficient communication, nodes require knowledge of their neighborhood size. The neighborhood of a node is typically the set of nodes to which it can communicate reliably. However, in dynamic and mobile network topologies, the neighborhood size of a node may change continuously with time.

Data Gathering Data Gathering forms an important component of most wireless sensing and monitoring applications, in which nodes periodically sense data and report them to a basestation. Applications that involve data gathering span different network topologies, and therefore demand different approaches to protocol design. For example, data gathering in a static wireless sensor network is usually accomplished by setting up a tree structure rooted at a so-called data sink. Routing protocols such as Collection Tree Protocol [33] and Backpressure Collection Protocol [88] use the underlying tree structure to relay data messages to the sink. In contrast, for mobile network topologies, data gathering algorithms takes the form of epidemic routing mechanisms [149], probabilistic mechanisms [80] and replication schemes [122] for delay-tolerant networks (DTNs). Especially for DTN routing, data reliability is preferred at the expense of an increased latency in data acquisition at the sink. An important requirement for most data gathering protocols is that of maintaining a neighborhood table locally on each node. The neighborhood table contains not only the identities of a node's neighbors, but also the link quality between the node and its neighbors. Neighborhood information is gathered by the exchange of broadcast messages that include specific routing information such as the number of hops from the sink, and the residual node energy. Gathering neighborhood information in a static wireless network is straightforward, and also easily maintained in the face of changing link qualities. In contrast, discovering neighbors in a mobile network topology is a non-trivial matter, as a node's neighborhood, i. e., both the neighbors and the neighborhood size change continuously with time. Estimating the neighborhood size at runtime is therefore, an important requirement for applications involving mobile networks.

In-Network Data Processing In-Network processing is performed in wireless networks to compress and filter the large volume of data that it generates. Alternatively, it could also be used to collaboratively compute an aggregate of interest. Typical scenarios where in-network processing is used include data aggregation in WSNs [83], RFID counting of tags in inventory warehouses [140], and estimating properties of dynamic networks such as churn rate [110], and communication error rate [111]. Data aggregation in wireless sensor networks is primarily motivated by the need to compress and filter out relevant information from raw sensor observations. Protocols that perform data aggregation in static networks either leverage the tree structure that is constructed at network bootstrap [119], or setup a tree structure online in a completely distributed manner [30]. Having setup the tree structure, aggregation proceeds in bottom-up manner, with each node computing a partial aggregate and relaying it to the next node along the tree hierarchy. Tree-based aggregation protocols work best on a static network topology in which node and communication failures either do not occur or are inconsequential. Therefore, approaches such as Synopsis Diffusion [99] have been studied to address the issue of faulty aggregates that accumulate in a network topology subjected to communication errors. However, for mobile networks wherein wireless links change continuously, an aggregation structure for in-network processing would be unfeasible to maintain. Therefore, researchers have looked at gossip-based protocols [57, 110, 111] to compute aggregates without the need for any structure. Although gossip-based in-network processing is simple and lightweight in its implementation, it is also extremely sensitive to communication errors. Currently, research efforts are being made to meet this challenge.

Time Synchronization Time Synchronization finds application in wireless networks for scenarios in which the communication and sensing of deployed nodes have to be co-ordinated. For example, a time-synchronized schedule for communication is desired for bulk transfer of bursty data in WSNs [113], or to disseminate information as quickly as possible [29, 81]. Time synchronization is also crucial for low-power networks such as WSNs where there is a strong need to conserve energy, without significantly affecting communication efficiency. By scheduling neighboring nodes to wake up at the same time, the issue of communication efficiency and end-to-end latency is addressed. Protocols for time synchronization in wireless networks either use the underlying network structure [84], or rely on a purely decentralized approach, using broadcast message exchanges between neighbors to converge to a common time reference [143]. Challenges faced in time synchronization range from hardware issues such as clock drift and skew, to more complicated factors such as node mobility and fragmented networks. While the hardware differences have been somewhat addressed by recent advances in crystal oscillator circuitry, the more complicated issues of node mobility and simultaneous network bootstrapping leading to disjoint networks still remain subjects of active research.

1.3 Reconfigurable Software: Necessity and Challenges

We have thus far identified building blocks of inter-node communication in networked embedded software. Real-world wireless applications that run in ad-hoc networking scenarios typically face a number of challenges that affect their operation. These challenges include network downtime caused by node or basestation failure [142], theft or vandalism [3], and to a large extent, failures in the embedded software itself [68]. Additionally, the network operation is continuously subject to topological dynamics, i. e., fluctuations in wireless link quality and node mobility. It is therefore, necessary to ensure that the software accounts for these network dynamics at runtime, and incorporates features for robustness to system failures. In this thesis, we focus particularly on two issues that arise under topological dynamics: *(i)* maintaining communication robustness between nodes, and *(ii)* reasoning about network connectivity at runtime. Guaranteeing reliability and robustness to failures for wireless network applications touches issues beyond inter-node communication, such as hardware failures, faulty calibration, and power outages. We maintain that ensuring reliability at all levels of a wireless application is a rather broad and generalized topic, and beyond the scope of this thesis. Therefore, we restrict our study to the problem of ensuring reliable communication between nodes in the face of failing wireless links and node mobility.

Achieving communication reliability in dynamic wireless networks warrants a preliminary discussion on the manner in which it must be achieved, and its efficiency. We discuss these concerns in the following:

Adaptability by Automatic Reconfiguration As described earlier, a wireless application for networked embedded devices relies on a set of communication components or *protocols* for its operation. The protocols are built from algorithmic components that comprise a set of parameters. For example, a routing software for WSNs such as the Collection Tree Protocol (CTP) [33] relies on a threshold for performing a routing switch, and timing parameters for transmitting

routing information, the so-called *beacons*, as well as data retransmissions. Another example is that of medium access control (MAC) protocols for wireless networks that fix the beaconing time interval for inter-node communication. Generally speaking, protocol parameters take a range of values that are decided upon during system design. These decisions are based upon relevant considerations such as the expected number of nodes in a neighborhood, the average or worst-case expected traffic flow, and the worst-case wireless link quality. However, when dynamics such as link degradation and node mobility scale to proportions that the allowable parameter range cannot handle, the system software either crashes or suffers a loss of performance. Therefore, automatic reconfiguration of protocol parameters *at runtime* is a means for achieving adaptability in networked embedded software.

Decentralized Operation Reconfiguration of protocol parameters could be either restricted to a local neighborhood of nodes, or may prove necessary for the entire network. In either case, the adaptive action is triggered in response to observed phenomena such as an increase in channel activity, or a sudden loss of packet reception. Additionally, these observations could apply either to a single node or a region of nodes within the network. For reasons of achieving scalable solutions, it is desirable to design and develop decentralized algorithms that rely only on local information to trigger the desired parameter reconfiguration. While decentralized approaches to adaptability may not always achieve the same level of performance as their centralized counterparts, they preclude the need for additional overhead in acquiring the global knowledge that is necessary for reconfiguration. In this regard, this thesis explores the possibilities of designing decentralized algorithms that adapt relevant parameters in the face of dynamics such as wireless link failures and topology changes caused by node mobility.

Isolating Applications from Adaptability Decentralized algorithms for adaptability demand an exchange of control messages between nodes for collaborative processing. Since both the network application and the adaptability component share the same medium for inter-node communication, care must be taken to ensure that their operations do not interfere with one another. One way of achieving the required isolation would be to piggyback the control messages on application payload. However, there are situations in which an adaptive algorithm may impose real-time requirements on communication, which may not be satisfied without changing application-specific parameters such as the data transmission rate. In such cases, some form of communication scheduling becomes necessary. We re-iterate this issue throughout the thesis, and provide suggestions towards isolating the application from the software that is responsible for adaptability.

1.4 Problem Statement

The objective of this thesis is *to perform a constructive study of decentralized algorithms for adaptability in dynamic wireless networks characterized by failing links, intermittent connectivity and continuous topological changes.* To do so, we identify four specific, and yet, relevant problems that are researched in the wireless community. Having identified the challenges, we design, develop, and implement on hardware wherever possible, decentralized algorithms and quantify

their performance under diverse scenarios of topology dynamics. Finally, we summarize the commonalities in all the solutions that we develop, and make suggestions or recommendations for a practical implementation on real-world networks. The specific chosen problems relate mostly to the scheduling of the communication channel between nodes, and the reachability of information in the case of intermittently connected networks. In the following, we describe the problems in more detail.

1.4.1 Desynchronization in Dynamic Networks

A fundamental requirement of most wireless sensing applications is the co-ordination of communication and sampling processes between nodes that monitor a common phenomenon. To achieve co-ordination, researchers have looked at approaches that attempt to maximize the sensing coverage of a neighborhood over time, while keeping the sampling process or energy consumption on a node at a minimum [4, 41, 58]. While such approaches are novel, lightweight and easily implementable on current hardware platforms, they relate more to the co-ordination of sensing processes between neighboring nodes, than to inter-node communication. In particular, arbitrating the wireless communication channel between nodes remains largely a difficult problem. This so-called MAC (medium access control) problem has been investigated with focus on diverse aspects, including maximizing throughput [51, 151], ensuring fairness [53], and reducing the number of collisions in a network [67, 115]. In general, when timely delivery of sensed data is important, as with real-time and safety-critical applications, it is considered desirable to space apart nodes' channel access uniformly in time. As wireless link quality changes over time, and as nodes move, there is a strong need to design a self-organizing, self-adaptive mechanism that achieves a fair schedule, while maintaining a desired level of channel utilization.

Recently, desynchronization of node communication in wireless networks has been proposed as a solution to the aforementioned problem [14, 18, 19]. The main idea of desynchronization is to have nodes continuously adapt their transmission events, the so-called *firing phases* - using a temporal ordering of their neighbors' phases. This leads to a self-organizing schedule in which the firing phase of nodes are uniformly spaced apart in time. The desynchronization algorithm can adapt to gradual changes in a node's neighborhood, while precluding any prior requirement such as time synchronization. The seminal work on deysnchronization was evaluated on fully-connected networks [19], and shown to converge to a state in which message transmissions in a neighborhood are spaced at equal intervals of time. However, in wireless networks that are deployed in the real world, the desynchronization problem is complicated by two factors; namely *(i)* the multihop setting of node neighborhoods, which leads to the classic hidden terminal problem, and *(ii)* node mobility, which can induce an oscillatory behavior in the network. Current extensions to the desynchronization algorithm [18, 91] propose incremental mechanisms that focus on specific aspects of the aforementioned problems. However, there is little intuition on the following two questions: *(i)* what are the factors that influence the performance of the desynchronization algorithm?, and *(ii)* how well does the desynchronization algorithm adapt to node mobility? We believe that the answers to these questions hold the key for identifying the set of parameters that can be adapted under network dynamics. This would then provide the necessary base for designing adaptive and self-organizing algorithms for communication in dynamic networks.

1.4.2 Neighborhood Context in Dynamic Networks

One of the essential building blocks for an application running on large-scale, complex networks is the ability to *discover the neighboring nodes* within communication range; algorithms often presume a ready availability of neighborhood information. Such a service proves valuable for the cause of imparting so-called *self-** system properties, i. e., allowing the system to correct or adapt its behavior at runtime in the face of dynamic and ad-hoc communication scenarios. Typical examples include routing in large-scale mobile ad-hoc networks [93], timely dissemination of information in vehicular networks [117], and estimation of churn in overlay networks [110]. Sometimes, discovery of wireless devices constitutes a major part of the application, such as in [96]. Discovering the neighborhood in a dense network is tightly linked to sharing of the communication channel. This problem aggravates especially at high traffic loads, where message collisions prevent any form of meaningful co-ordination between participating nodes. Furthermore, with the case of node mobility, communication links tend to become intermittent, and only persist for short durations. Both these problems demand a self-adaptive behavior within the wireless network, in which every node adjusts its communication parameters in response to changing circumstances in its local neighborhood topology. Existing approaches for neighborhood discovery in wireless networks [9, 24, 54, 86, 132] either fix the communication parameters at deployment [124], target static networks [15] where node density does not change significantly over time, or focus on a specific application such as routing information to a reference node [124]. They are therefore, not suited for mobile, ad-hoc networking scenarios. While there are adaptive approaches for mobile networks, they are either system-specific [56], or require knowledge of the neighborhood [44, 121].

1.4.3 Temporal Connectivity in Dynamic Networks

An important information about a deployed wireless network concerns the connectivity between nodes. Especially, how reachable a node is from every other node in the network provides a wealth of information regarding the spread of information [34], the best route to send a message from source to destination [87, 114], and relationship between participants in a large crowd [101]. Existing work analyzes the dissemination delay in mobile wireless networks from a purely spatial perspective [35, 62, 65]. That is, the network is always connected at any point in time. In reality, a network could be composed of several disjoint components, and yet the nodes could be connected to each other over time. Recent work studies the connectivity in mobile network topologies in a temporal sense, i. e., the time taken by messages to reach a destination node from a source. The fundamental metric for connectivity is expressed in a spatial-temporal context, and referred to as a so-called temporal distance [116, 127, 128]. The temporal distance is defined as the shortest actual time taken by the message to reach the destination from a source. Using the set of observed temporal distances, it is possible to compute aggregate metrics of connectivity such as the temporal efficiency and temporal closeness. However, at present, these temporal metrics are computed either offline over real-world mobility traces [128], or in a centralized manner [116] on simplistic connectivity graphs such as the Erdos-Renyi model. In contrast, a localized view on aggregated temporal connectivity would provide valuable feedback information

in specific application scenarios such as managing personal interactions in large groups, and adapting storage parameters for messages in transit.

1.4.4 Mitigating External Interference in Low-power Networks

An important requirement for wireless sensor network deployments is that of autonomous operation, i. e., nodes should be able to adapt their communication to changing conditions in the wireless channel. Specifically, sensor nodes radios operate in the unlicensed ISM bands, and often face the problem of having to share the communication medium with other devices in the neighborhood. In particular 802.15.4 radios, which operate in the 2.4 GHz range, interfere with prominent communication technologies like Wi-Fi (802.11) and Bluetooth. Especially, the predominant class of data collection applications for WSNs tend to suffer in data reliability on account of frequent occurrences of external interference. Interestingly, sensor node transceivers offer communication on different non-overlapping frequency channels. This multichannel feature could be leveraged to ensure a seamless operation of sensornets in the face of severe external interference.

1.5 Thesis Contributions

In this thesis, we address the four research problems described previously, and conclude our work by listing commonalities in our approaches, and avenues for future work. Specifically, our contributions can be summarized as follows:

Characterizing Desynchronization and Resolving Message Collisions (Chapter 2) We quantify the benefits of using the desynchronization algorithm in multihop and mobile wireless networks, and compare its performance against a plain, randomized communication mechanism. To do so, we develop a novel metric for quantifying the performance of desynchronization, and characterize it under diverse conditions of node mobility and multihop settings. Additionally, we present *CollisionDetect*, a decentralized algorithm that resolves the difficult problem of hidden terminal collisions in multihop and mobile wireless networks. The novelty of *CollisionDetect* stems from the fact that it does not rely on restrictive assumptions as other collision resolution mechanisms in the literature. Our experimental results indicate that while desynchronization works well in sparse and static networks, it would be advisable to stick to randomized communication for dense and mobile network topologies. Additionally, a collision resolution algorithm such as *CollisionDetect* reduces the number of collisions upto a factor of two over a randomized scheme.

Neighborhood Discovery in Dynamic Networks (Chapter 3) We address the importance of discovering neighbors efficiently in dynamic network topologies characterized by node mobility. This problem touches the combined concerns of estimating the neighborhood density at runtime, and adapting the transmission parameter on nodes in order to maximize communication efficiency. We present *Nest*, an adaptive and decentralized algorithm that estimates the neighborhood density and adapts the so-called communication beaconing parameter at runtime. *Nest* uses only local interactions between neighbors, and relies on a statistical estimation of node

density to gather neighborhood information as quickly as possible. We showcase the performance of *Nest* through extensive simulations under diverse network topologies, and real-world mobility traces. Additionally, we present our experiences from a practical implementation of *Nest* on real hardware.

Online Estimation of Temporal Connectivity (Chapter 4) We present *PathDetect*, an adaptive algorithm that characterizes the temporal connectivity in a mobile network, using only local message exchanges between nodes. *PathDetect* combines distributed consensus with a statistical representation of temporal distances to infer the average temporal connectivity in the network. We showcase, through extensive simulations, the ability of *PathDetect* to reflect changes in connectivity owing to random node and link failures, and slow changes in network topology. Particularly, we discuss issues and challenges to be faced towards a practical implementation of *PathDetect* on real hardware, and in realistic mobility settings.

Mitigating External Interference in Low-power Networks (Chapter 5) We present *Chrysso*, a multichannel protocol that mitigates the effect of external interference on low-power sensor networks. We focus specifically on the class of data gathering applications in wireless sensor networks, and leverage the existing tree structure to perform collaborative decision making. We present an exhaustive evaluation of *Chrysso*, including experimental results from three different testbeds. Our evaluation features detailed testcases with artificially-induced jamming, real-world WiFi interference, experimental runs over a long timescale, and on a large setup of nodes. Additionally, we compare the performance of *Chrysso* with that achieved by *MuChMAC*, a state-of-the-art channel hopping protocol for sensor networks, and show that *Chrysso* exhibits greater data reliability and stability.

The research contributions in this thesis apply to different application scenarios for wireless networks. Notably, the research contributions in Chapters 2, 3 and 4 apply largely to mobile, ad-hoc networks. The research contribution in Chapter 5, however, applies to the specific application domain of Wireless Sensor Networks.

1.6 Thesis Outline

The outline of the thesis is as follows: we start our research by studying the performance of desychronization in dynamic wireless networks (cf. Chapter 2). Next, we address the problem of estimating neighborhood density in dynamic network topologies (cf. Chapter 3). We then extend our research towards estimating the connectivity between nodes in mobile networks (cf. Chapter 4). As a final contribution, we present an experimental evaluation of our adaptive multichannel protocol for mitigating external interference in wireless sensor networks (cf. Chapter 5). Chapter 6 concludes the thesis. As a note, we cover the related work in every research chapter. The publications associated with the research contributions in this thesis are listed as follows:

1. V.G. Iyer, A. Pruteanu, and S.O. Dulman. Netdetect: Neighborhood discovery in wireless networks using adaptive beacons. In Proceedings of IEEE Self-Adaptive and Self-Organizing Systems (SASO) Conference , 2011, pages 31-40.

2. V.G. Iyer, M. Woehrle, and K.G. Langendoen. Chrysso - a multi-channel approach to mitigate external interference. In Proceedings of IEEE Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2011, pages 422-430.

3. V.G. Iyer, M. Woehrle, and K.G. Langendoen. Chamaeleon - exploiting multiple channels to mitigate interference. In Proceedings of IEEE Networked Sensing Systems (INSS), 2010, pages 65-68.

Chapter 2

Desynchronization in Dynamic Wireless Networks

We begin our research by focusing on the distributed coordination problem in wireless neighborhoods. A fundamental requirement of many wireless network applications is the coordination of node communication and sensing in a neighborhood. Arbitrating the communication channel access between neighboring nodes is necessary in order to provide per-node fairness. However, achieving a fair and efficient channel access between nodes is a non-trivial problem for real-world networks, mainly because of inter-node contention and simultaneous message transmissions leading to collisions. Traditional scheduling mechanisms for communication involve either centralized computation [53], rely on pre-determined structures at runtime [115], or focus specifically on competing traffic flows in a multihop network [51]. However, the inclusion of topological dynamics such as intermittent links and node mobility make the afore-mentioned solutions unsuitable for scheduling inter-node communication. The fact that a wireless neighborhood could change continuously suggests that algorithms designed for channel access need to reconfigure specific parameters related to node communication, notably the transmission event, on every node. This could be carried out in a continuous and adaptive manner on each node, and by relying on collaboration between neighbors, i. e., nodes within communication range.

In recent years, researchers have looked at Desynchronization [14, 18, 19], an adaptive and decentralized algorithm that arbitrates channel access across nodes in a neighborhood uniformly over time. The core idea behind desynchronization is to have every node continuously adapt its so-called transmission phase based on the observed temporal ordering of its neighbors' phases. This seemingly simple local rule ensured convergence to a state in which the transmission phases of nodes in a neighborhood was uniformly spaced over time. Desynchronization is perceived to be a self-adaptive, self-organizing algorithm that can adapt to slow changes in neighborhood topology without explicit requirements on time synchronization. However, a direct application of the desynchronization algorithm to real-world settings faces the following challenges: *(i)* hidden terminal nodes in a multihop network that lead to message collisions, and *(ii)* fast-changing neighborhoods in a mobile network that make it unfeasible to establish a uniformly spaced transmission schedule. It is therefore of interest to study the factors that influence the performance of desynchronization algorithm in multihop, static as well as mobile network settings. Furthermore, given that the desynchronization algorithm incurs overhead on communication and processing,

it is also important to determine whether a naive and yet simple communication mechanism serves as a lightweight alternative for the case of mobile networks. Such a study would offer insights over the choice of parameters to adapt at runtime. This in effect would set the tone for the design of decentralized, and adaptive algorithms for communication.

We study the desynchronization problem, by looking at how well-spaced node transmissions are in a wireless neighborhood, and the number of message collisions that are witnessed in a network. As a prerequisite, we formulate a novel metric, the so-called *desync spaceout* that describes how uniformly node transmissions are spaced in a wireless neighborhood. Interestingly, recent works on desynchronization [16, 91] address the desychronization problem with the assumption that a collision resolution mechanism is in place. Although message collision resolution mechanisms exist in practice, most approaches are constructive, i. e., they rely on time sychronization and bitmaps [134] that represent slot allocations, or node identifiers [31]. In contrast, we present *CollisionDetect*, a collision resolution mechanism that does not rely on either time synchronization or on explicit slot allocation bitmaps. Nodes that run *CollisionDetect* detect collisions in real-time and notify their neighbors to change their slot allocations, thereby addressing the more difficult case of the hidden terminal problem. *CollisionDetect* combines the detection of message collision with relative timestamp ordering of received messages to notify message collisions. Evaluation of *CollisionDetect* shows a reduction in message collisions of upto a factor of two, compared to a naive randomized communication scheme. Our proposed mechanism is general, and applicable to both asynchronous and time-synchronized modes of communication in wireless networks.

We evaluate the desynchronization algorithm through simulations involving various network topologies. Typically, we look at multihop static as well as mobile network topologies. For every network topology, we compare and contrast the performance of desynchronization with a naive approach in which nodes transmit messages randomly within a given time interval, on a periodic basis. A randomized approach might not offer the desired performance, but entails minimal overhead compared to a desynchronization protocol. In our study, we address the question of whether a desynchronization algorithm is necessary for most topological scenarios, or what topologies favor a randomized mechanism over desynchronization.

The rest of the chapter is structured as follows: Section 2.1 describes the desynchronization algorithms and introduces the desync spaceout metric. Additionally, it also presents a mechanism for resolving message collisions. We present simulation results that characterize the performance of desynchronization in Section 2.2. Section 2.3 discusses related work, and Section 2.4 concludes the chapter.

2.1 Desynchronization in Wireless Networks

In the following, we first describe the desynchronization algorithm with the help of a system model. Thereafter, we introduce the desync spaceout metric that quantifies desynchronization. Additionally, as an extension to a state-of-the-art desynchronization algorithm, we present *CollisionDetect*, a mechanism for detecting and resolving message collisions.

2.1.1 System Model and Problem Statement

We start by defining a wireless neighborhood I for a node as the set of nodes that are within communication distance. Nodes communicate with each other by transmitting short messages called *beacons* that minimally convey their identities. We are interested in characterizing how well-spaced nodes' communications are in a neighborhood of N nodes, each labelled $(n_1, n_2, .., n_N)$. Unless otherwise stated, nodes access the channel in periodic intervals of time, hereafter referred to as a *round*. For the sake of convenience, we represent a round as an integral number of so-called time slots, each slot wide enough to hold a message transmission. We denote the round size by R. Note that we do not assume the existence of time synchronization; nodes operate asynchronously with respect to one another. A neighbor n_i transmits a beacon at a randomly occuring slot within a round, that bears an offset ϕ_i from the *virtual* start of the round. In this regard, we refer to a so-called firing phase for node n_i. Since nodes in a neighborhood may fire at different instances, we refer by Δ_{ϕ_i} the phase difference between the firing phase of nodes n_i and n_j, where node n_j is the node that fires immediately after n_i. Therefore, $\Delta_{\phi_i} = \text{mod}(\phi_j - \phi_i, R)$. The objective of a desynchronization primitive is to arrange the firing phases of nodes in a neighborhood such that $\forall i \in (1, N), \Delta_{\phi_i} = \frac{R}{N}$. That is, the transmission events of the nodes are to be spaced apart uniformly over round R. Note that as $\frac{R}{N}$ is a real-value, it might not be possible to space apart transmission events in an exact sense. Nonetheless, it provides a baseline for observing how well-spaced message transmissions are in a neighborhood.

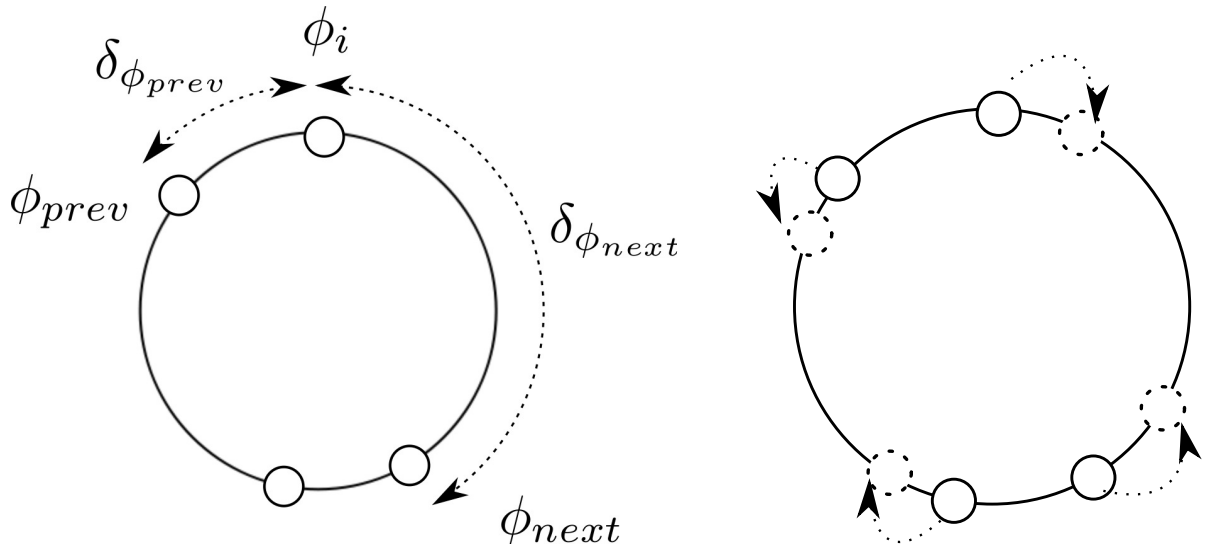
2.1.2 The Desync Algorithm

The desynchronization algorithm by Degesys et al. [19], referred to hereafter as *Desync* tries to space apart node transmission in a neighborhood iteratively over rounds. As mentioned under Subsection 2.1.1, nodes transmit periodically over a time interval. The ordering of transmission events in a neighborhood of nodes can be represented by so-called *firing phases* that bear an offset from a virtual start of the beaconing round. Specifically, a node n_i transmits at an offset ϕ_i from the start of the round. The phase offsets can be visualized as dots over a circle of length R, as shown in Figure 2.1a. A node running the Desync algorithm maintains information regarding the previous and next firing phases of its neighbors, denoted as ϕ_{prev} and ϕ_{next} respectively. The phase values are ordered as shown: $\phi_{next} > \phi_i > \phi_{prev}$. A node running *Desync* updates its firing phase ϕ_i as soon as it observes ϕ_{prev} and ϕ_{next}. The updated value for a node is a function of the phase difference between its own and that of its next and previous firing phases. Specifically, using the locally recorded values for ϕ_{prev} and ϕ_{next}, every node computes a new value for its firing phase as shown:

$$\phi_i' = \phi_i + \frac{1}{2}(\delta_{\phi_{next}} - \delta_{\phi_{prev}}). \tag{2.1}$$

where $\delta_{\phi_{prev}} = \text{mod}(\phi_i - \phi_{prev}, R)$ and $\delta_{\phi_{next}} = \text{mod}(\phi_{next} - \phi_i, R)$, are the phase difference between the nodes own firing phase and that of its previous and next neighbor respectively. Finally, the node updates its firing phase using a linear combination of its current firing phase ϕ_i, and the computed ϕ_i', as shown:

$$\phi_{inew} = \alpha_{desync}\phi_i + (1 - \alpha_{desync})\phi_i'. \tag{2.2}$$



(a) Round t **(b)** Round $t+1$

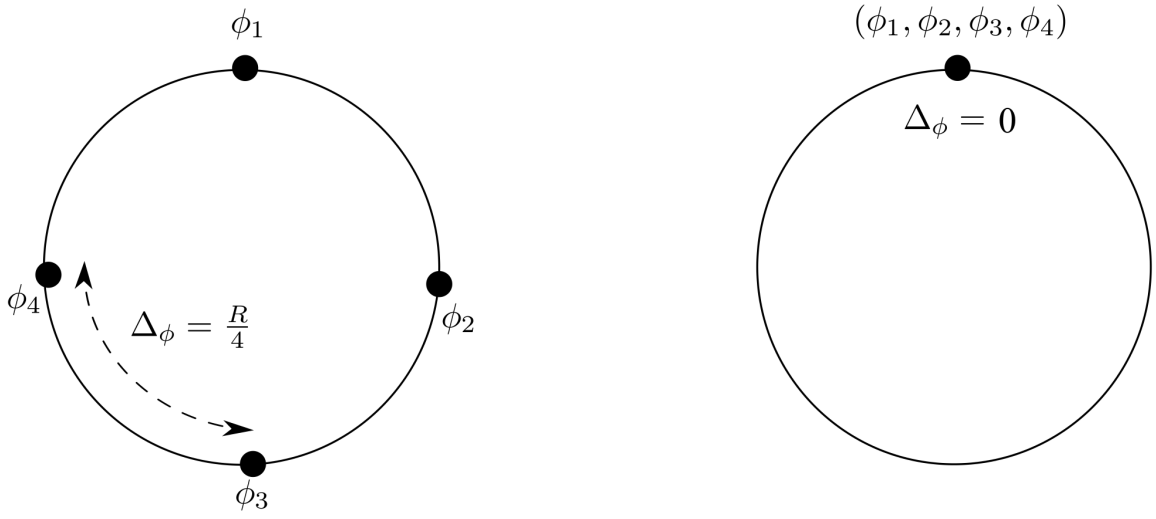
Figure 2.1: Desynchronization step at round t. Dotted circles represent newly computed firing phases.

Equation 2.2 summarizes the Desync operation, which is essentially a distributed controller that operates in discrete rounds. The term α_{desync} represents the controller coefficient that decides the convergence speed and stability of the Desync algorithm. The work by Degesys et al. [19] recommends setting α_{desync} to a large value, i. e., $\alpha_{desync} > 0.5$, with the effect that changes to firing pulses are very gradual. This ensures a stable operation of the distributed controller. Figure 2.1b shows a snapshot of a Desync operation, in which the participating nodes revise their transmission slots (or firing phases) based upon their previous and next firing phases in line. The Desync algorithm was initially evaluated on a clique, and was found to converge in $O(N^2)$ rounds, where N is the number of nodes in the clique.

In what follows, we extend the current work on desynchronization with a metric that quantifies its performance, and a novel decentralized mechanism for resolving message collisions.

2.1.3 Understanding Desynchronization Performance

The Desync algorithm described in Section 2.1.2 relies on a simplistic assumption, i. e., nodes in a neighborhood initially choose unique or non-overlapping firing phases. However, in practice, the initial configuration of firing phases (or transmission slots) is often a random selection over the round size. This means that the slots chosen by the nodes are not strictly unique. Applying the Desync algorithm over such a configuration results to a converging state in which a firing pulse may still be shared between nodes. Furthermore, the evaluation of Desync was initially restricted to a fully-connected topology [19], and later extended to multihop wireless networks [18]. Specifically, the existence of hidden terminals in multihop settings complicate the evolution and convergence of the Desync algorithm. To mitigate this problem, the works presented in [18, 94] propose extending the desynchronization process to apply over a two hop neighborhood. However, doing so deviates from the original objective of splitting the channel usage uniformly between nodes in a wireless neighborhood. In general, while related work



(a) desync spaceout = 0

(b) desync spaceout ≈ 1, for large neighborhoods

Figure 2.2: Desync spaceout for fully-desynchronized (left) and fully synchronized (right) states

implements desynchronization in one form or the other, it still remains to be seen how well-spaces message transmissions are in a wireless neighborhood. Therefore, it is necessary to formulate a metric that quantifies desynchronization performance.

In order to do so, we consider a local neighborhood of N nodes, transmitting once within a round of size R. As mentioned earlier in Section 2.1.1, having nodes perfectly desynchronized implies that the absolute phase difference between the consecutive firing phases of nodes is $\frac{R}{N}$. We denote this phase difference by $\bar{R}$. Let $\Delta_\phi = (\Delta_{\phi_1}, \Delta_{\phi_2}, .., \Delta_{\phi_N})$ be the set of phase differences (computed modulo R) observed for all nodes $n_i \in (1, N)$. Note that Δ_ϕ represents a distribution of phase differences that are computed over *unique* values of ϕ. For example, given a round size $R = 10$, and a set of phase values $(1, 2, 2, 5, 8)$, Δ_ϕ is computed as $(\text{mod}(1 - 8 + 10, R), 2 - 1, 2 - 1, 5 - 2, 8 - 5)$, or equivalently, $(3, 1, 1, 3, 3)$. We now express the performance of the Desync algorithm by the *root mean square* deviation of the phase differences with respect to the best-case value, i. e., $\bar{R}$. The resulting metric - the *desync spaceout* is normalized to the round size R, and expressed as follows:

$$\text{desync spaceout} = \frac{1}{R}\sqrt{\frac{\sum_i (\Delta_{\phi_i} - \bar{R})^2}{N}}. \tag{2.3}$$

The desync spaceout is computed locally for every node. In a multihop setting, this means that the desync spaceout would differ between neighbors. Therefore, as a network-wide metric for desynchronization, we consider the average over all desync spaceout. A desync spaceout value of 0 represents a best-case scenario (shown in Figure 2.2a), in which all nodes of a neighborhood transmit uniformly apart in time. Figure 2.2b shows an example in which the nodes have

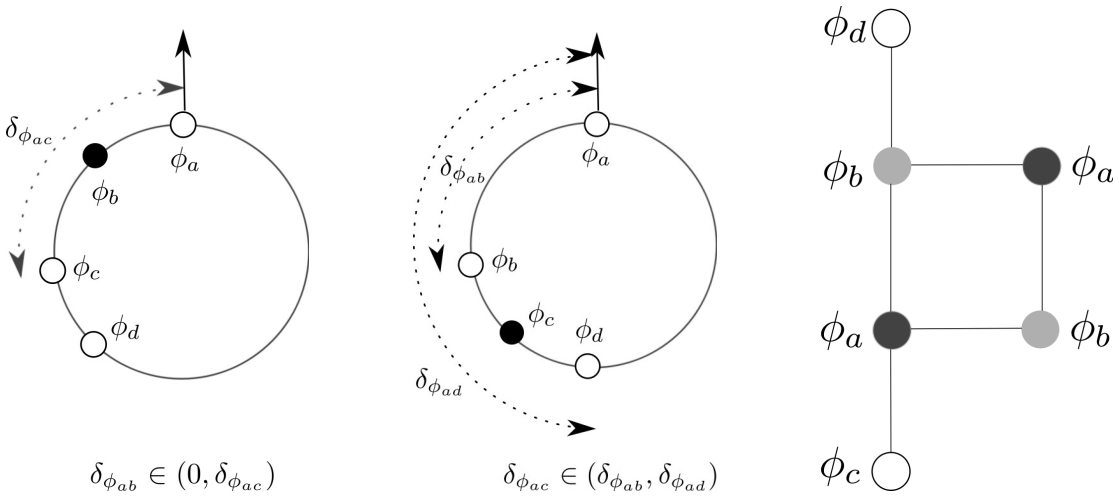
synchronized phases, such that their phase differences are zero. In this case, the computed desync spaceout equals $\frac{1}{N}$, which approaches 0 for large values of N. However, Figure 2.2b describes a worst-case scenario for desychronization, which is not represented by the computed spaceout value of $\frac{1}{N}$. Therefore, we map every observed phase difference of 0 to R, and recompute the desync spaceout defined in Equation 2.3. Doing so places an upper bound of $1 - \frac{1}{N}$ for the desync spaceout, where N is the neighborhood size. For large neighborhood sizes ($N > 10$), the spaceout value approaches 1. As observed from Equation 2.3, the desync spaceout is dependent on both the round size and the neighborhood density. The dependencies are elaborated upon later in Section 2.2.

2.1.4 CollisionDetect - Resolving Collisions

Recently, the work by Motskin et al. [91] has modeled desynchronization as a vertex coloring problem, in which nodes are assigned contiguous subintervals over an interval of length R. The assignment of the subintervals should be such that no two subintervals between neighboring nodes overlap. Therefore, with N nodes in a neighborhood, every node gets a share equalling $\frac{R}{N}$. However, for multihop networks, the neighborhood density varies across nodes, such that the share allotted to a node is a function of its neighborhood density. Therefore, every node n_i monitors its neighborhood density continuously, and determines the maximum node degree d_i in its one-hop neighborhood. Having performed this step, the node n_i chooses a subinterval of length $\frac{R}{2(d_i+1)}$ randomly over R. Furthermore, as a check to meet the constraints of the vertex coloring problem, the authors assume the existence of a collision detection mechanism among nodes. However, the collision detection mechanism proposed by [91] is simplistic in nature, i. e., nodes can discover overlapping intervals only with their neighbors. The more complicated case of hidden terminal collisions [131] has not been addressed.

In what follows, we present *CollisionDetect*, a distributed mechanism for detecting message collisions and resolving them in real-time in a neighborhood. Unlike conventional approaches for collision resolution [115, 134], *CollisionDetect* does not rely on time synchronization, encoding of slot allocation using bitmaps, node identifiers, and network infrastructures such as connected trees to mitigate the hidden terminal problem. *CollisionDetect* works asynchronously, and in an iterative manner on all nodes. Nodes that run *CollisionDetect* self-organize their communication into an almost collision-free schedule, by relying only on a temporal ordering of relevant communication channel events, i. e., message reception and collision.

In order to explain how *CollisionDetect* works, we consider a scenario in which nodes transmit only once within a round, with an initially fixed phase offset ϕ. Additionally, nodes are able to distinguish between specific channel events, i. e., idle, message reception and message collision. In particular, nodes monitor the channel continuously for message collisions, and record the firing phases of neighbors that successfully transmit a message. On detecting a collision, a node determines a time interval that encloses the event. For convenience, nodes use the phase values from received messages as the time reference. Figure 2.3a shows the example of a node that fires with a phase ϕ_a, and also records a collision at phase ϕ_b. The node additionally records the firing phases ϕ_c and ϕ_d of its neighbors, and bounds the collision event over an interval $(0, \delta_{\phi_{ac}})$, where the values 0 and $\delta_{\phi_{ac}}$ represent the phase difference between ϕ_a and itself, and ϕ_c respectively. At the next firing event of the node, an explicit collision notification



(a) Example 1 **(b)** Example 2 **(c)** Lost Notification

Figure 2.3: Nodes running *CollisionDetect* notify their neighbors regarding observed message collisions. A collision may occur just after the node fires (example 1), or at random phases within the round R (example 2). The figure on the right shows a topology in which the notification mechanism is deadlocked.

is transmitted with the value $\delta_{\phi_{ac}}$ piggybacked on the payload. On receiving the message, the colliding nodes with phase ϕ_b determine that their phase difference with respect to ϕ_a falls within the bound, i. e., $\delta_{\phi_{ab}} \in (0, \delta_{\phi_{ac}})$. Only nodes whose messages collide pick a new transmission slot at random over unoccupied portions of round R. Figure 2.3b shows another example of a collision notification, in which the node with firing phase ϕ_a bounds a message collision over a non-adjacent interval $(\delta_{\phi_{ab}}, \delta_{\phi_{ad}})$.

The ability of *CollisionDetect* to detect and resolve message collisions is strongly influenced by the round size R. Particularly, for large values of R (i. e., several multiples of neighborhood size), *CollisionDetect* would eventually lead the network to a state without any collisions. However, for small round sizes, *CollisionDetect* would degenerate to a setting in which nodes choose their firing phases at random over R. We would like to mention here that mitigating message collisions in a wireless network is essentially a two-step process, the first step being to adjust the round size on every node as a function of its neighborhood size. Our recent works [46, 47] have shown that this is achievable even for dynamic wireless networks. *CollisionDetect* forms the second step of the process, and essentially provides the fine-adjustment necessary to ensure a collision-free schedule. Additionally, while *CollisionDetect* can detect and notify collisions, resolving them is not strictly guaranteed. Specifically, it would not work on topologies in which the collision notifiers and the colliding nodes form a ring, as shown in Figure 2.3c. In the example, although the nodes with phases ϕ_a and ϕ_b collide, their collision notifications collide as well, resulting in a deadlocked state. However, such cases form a tiny fraction of collision occurrences.

In our assessment of the desynchronization problem, we adapt the *CollisionDetect* algorithm to mimic the work by Motskin et al. [91]. Additionally, we allow *CollisionDetect* to choose discrete and non-necessarily contiguous slots within a round. This is done in order to characterize the desynchronization process with the spaceout metric defined earlier in Section 2.1.3.

2.2 Characterizing Desynchronization

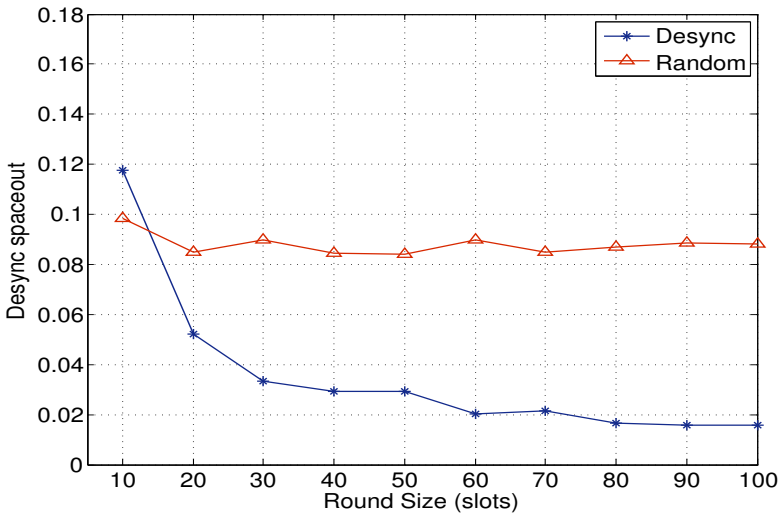
Given the multihop setting of real-world wireless networks, and the existence of mobile nodes in a network, we are interested in finding out whether desynchronization provides any benefit to the cause of efficient and fair channel access. In order to do so, we compare and contrast the performance of desychronization (i. e., both Desync and *CollisionDetect*) against a naive approach in which nodes transmit randomly within a round. Note that our research objective is to study the relative advantages of desychronization over randomized beaconing, and not to perform a quantitative comparison between different desynchronization algorithms.

All simulations are performed in Matlab. As metrics, we study the desync spaceout and the average message collisions recorded in the network. Specifically, we look at network topologies representing multihop mesh networks and mobile networks. We present our findings as follows: Firstly, we study the Desync algorithm and compare its performance against a randomized communication mechanism under diverse scenarios of network topology. Specifically, we look at multihop static as well as mobile networks, and characterize the influence of node density and mobility on the Desync algorithm. We repeat the characterization for a neighborhood density-aware desynchronization algorithm (i. e., *CollisionDetect*), and compare its performance with its corresponding randomized counterpart.

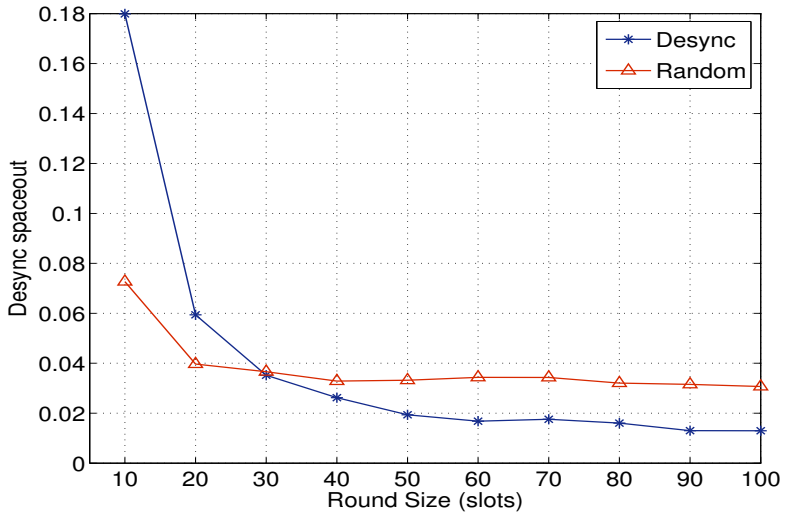
2.2.1 Desync vs. Randomized Beaconing

In this subsection, we study the influence of the round size R and neighborhood density on desync spaceout. This gives an insight on when Desync should be favored over randomized communication. Next, we extend our comparison to static multihop network settings, and characterize the influence of round size on both desync spaceout and message collisions.

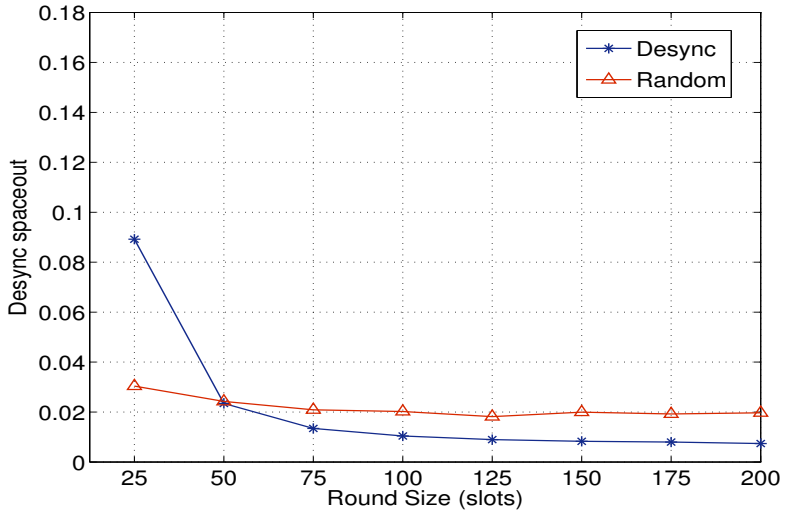
1. Fully-connected Networks We consider fully-connected neighborhoods of varying sizes. For every value of neighborhood density, we vary the round size and plot the resulting desync spaceout. Figure 2.4 shows the average desync spaceout values for both the Desynchronization algorithm and randomized beaconing, over varying round sizes, and for densities 10, 30 and 50 nodes. Since the standard deviation in the observed values is less than 10% of the mean in all cases, we do not plot them here. A couple of observations emerge from the values shown in Figure 2.4. Firstly, the desync spaceout for both Desynchronization and randomized beaconing levels off beyond a certain round size. A supporting observation from inspecting the data reveals that the phase differences (Δ_{ϕ_i} from Equation 2.3) scale almost linearly with the mean interval $\frac{R}{N}$. For a fixed value of N, this results in an almost constant value for desync spaceout. Secondly, we note that the desync spaceout for the Desynchronization algorithm is greater than that achieved by randomized beaconing, for small round sizes. Generally, at low round sizes, the number of nodes that pick the same firing phase is greater than what is observed at bigger values of R. In such cases, Desync relies on the few non-colliding firing phases of nodes to space each other out in time. In particular, neighboring nodes that collide observe the same $(\phi_{prev}, \phi_{next})$, and therefore collide again in the following round. The increased overlapping of transmission results in an increased value for desync spaceout (cf. Figure 2.2b in Section 2.1.3). This problem however, is mitigated with a randomized beaconing scheme, in which the firing phases of nodes are strictly independent of one another. Lastly, we note that for large round sizes, the absolute



(a) N = 10



(b) N = 30



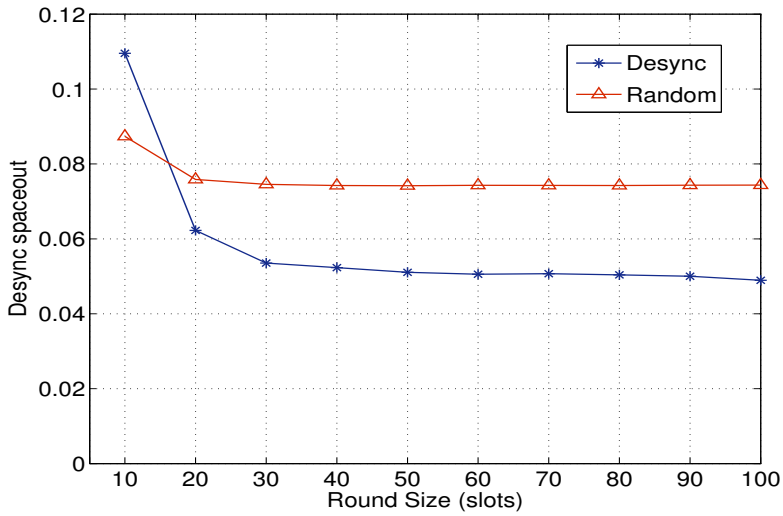
(c) N = 50

Figure 2.4: Average desync spaceout, for different neighborhood sizes, over varying values of round size R.

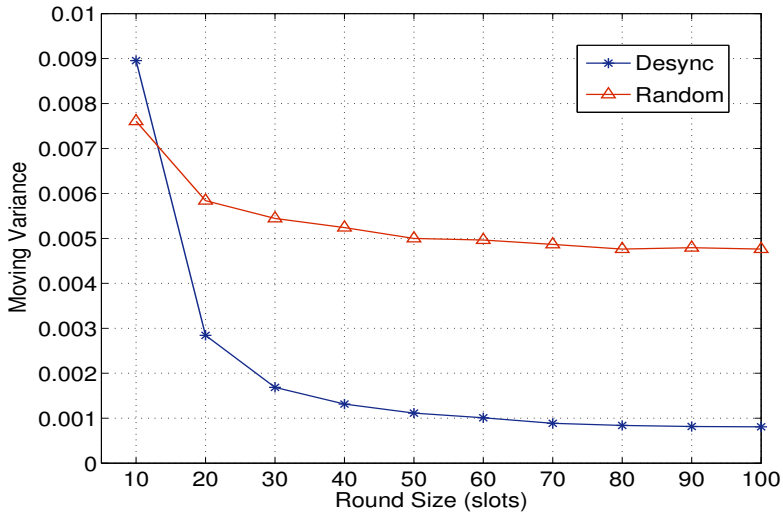
difference between the desync spaceout values for Desync and randomized beaconing begins to decrease as the neighborhood density grows in size. For example, for a round size of 100 slots, and for a neighborhood density of 10 nodes, Desync achieves a desync spaceout of 2%, as opposed to 9% achieved by randomized beaconing. However, for a neighborhood of 50 nodes, we find that the randomized beaconing achieves a mean desync spaceout of approximately 2%, while the desync spaceout achieved by Desync drops down to a mere 1%. This suggests that a randomized beaconing scheme improves in its ability to desynchronize nodes' firing phases at high neighborhood densities. As such, with only a marginal difference between the two, the randomized beaconing mechanism may be favored over Desync, especially in dense regions of a network.

2. Static Mesh Networks We evaluated both Desync and randomized beaconing on network topologies that comprise 119 nodes in a 1000m by 1000m area. The transmission range is set to 200m, which results in an average node density of approximately 15 nodes within a communication radius. We vary the round size in the range 10 to 100 slots in steps of 10 slots. This represents a transition from a state of high channel contention to one in which message collisions are kept to a minimum. Figure 2.5 shows the averaged desync spaceout, its moving variance over time, and the number of message collisions witnessed by a multihop network over different round sizes. Each point in the graph represents an average over 50 simulation runs. Specifically, from Figure 2.5a, we can attribute the stable trend in desync spaceout and the absolute differences in their values for Desync and randomized beaconing to the same reasoning as applied to fully-connected networks. Note that the standard deviation for desync spaceout between runs is less than 10% of the mean in all cases, so we do not plot the errorbars. Figure 2.5b shows the moving deviation in the network-wide computed average desync spaceout over time. The moving deviation is an indication of convergence for the network-wide average desync spaceout. From the figure, we see that the deviation achieved by both Desynchronization and randomized beaconing is negligible, which suggests that the desync spaceout achieved by the network converges to within a very small interval. Moreover, the Desynchronization exhibits a greater stability in desync spaceout than randomized beaconing. This is to be expected for a static topology, because nodes that actively desynchronize their transmission slots gradually reduce the magnitude of slot change over rounds. On the matter of message collisions, we note from Figure 2.5c that Desync performs only marginally better than randomized beaconing. However, as the round size is increased, both approaches achieve almost the same percentage of message collisions, leaving not much to choose between them.

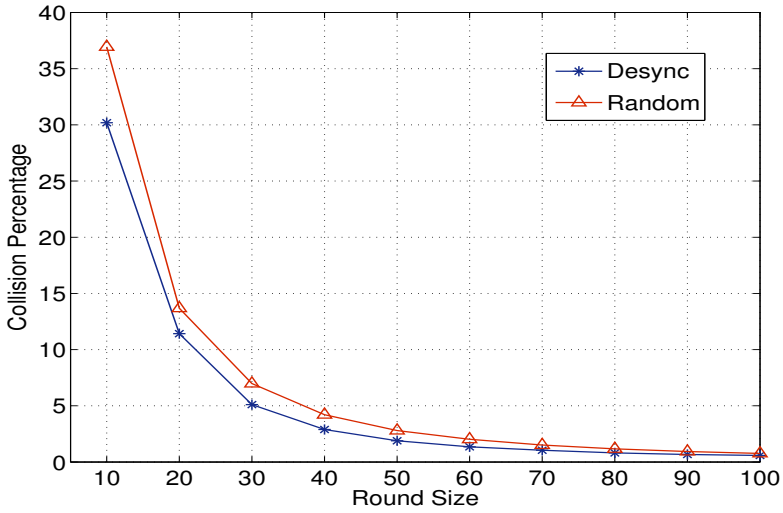
3. Mobile Networks We evaluate Desync and the randomized beaconing mechanism on a mobile network topology that is constructed using the RandomWalk mobility model. The transmission range for the nodes is set to 200m, while the maximum velocity of the nodes is varied in the range 1 m/s to 512 m/s, in increasing powers of two. Note that while the absolute values for node velocities do not reflect practical settings, our objective is to characterize the desynchronization performance when nodes could cover a distance beyond a transmission range per slot. Figure 2.6 shows the desync spaceout achieved by both Desync and randomized beaconing for varying values of maximum node speed, and for different round sizes. From the graphs, we see that in all the three cases shown, the desync spaceout begins to level off between velocities of 64 and 128m/s. This range relates to a mean velocity of roughly half the



(a) Mean Normalized desync spaceout

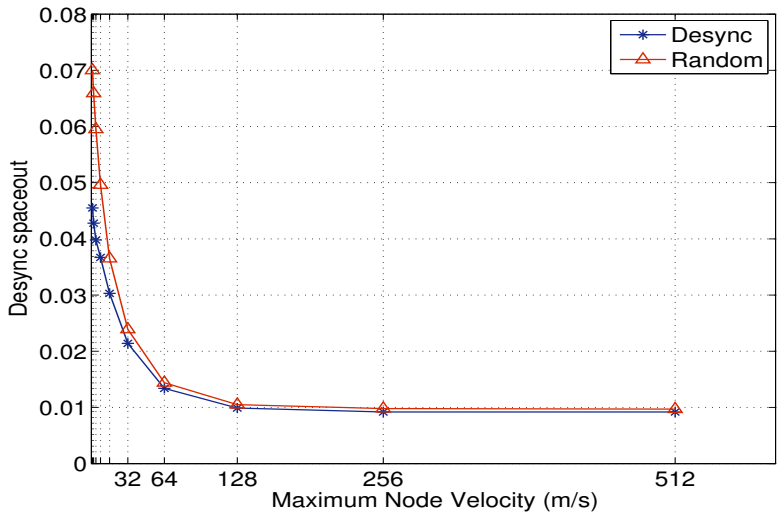


(b) Deviation

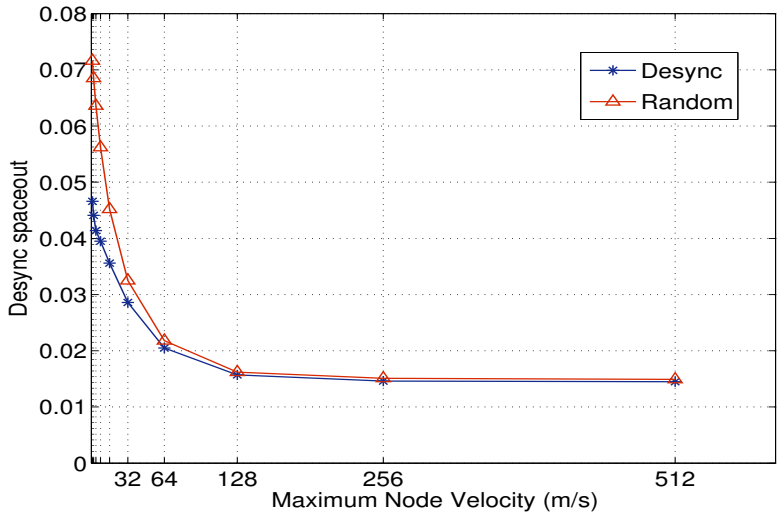


(c) Message collisions (%)

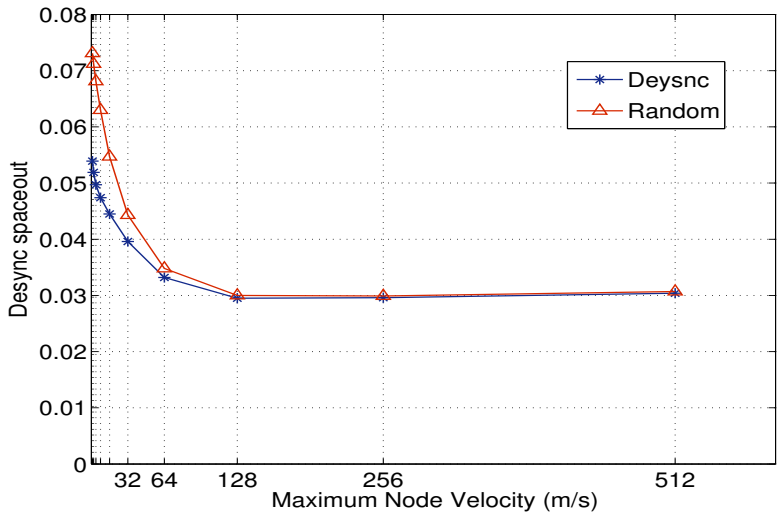
Figure 2.5: Average desync spaceout (top), moving variance in desync spaceout over time (middle), and message collisions (bottom) in static mesh networks over varying values of round size R.



(a) $R = 100$

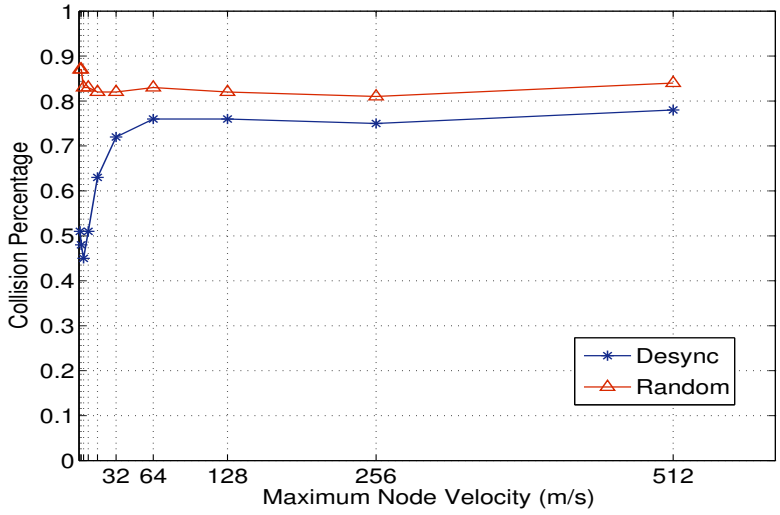


(b) $R = 50$

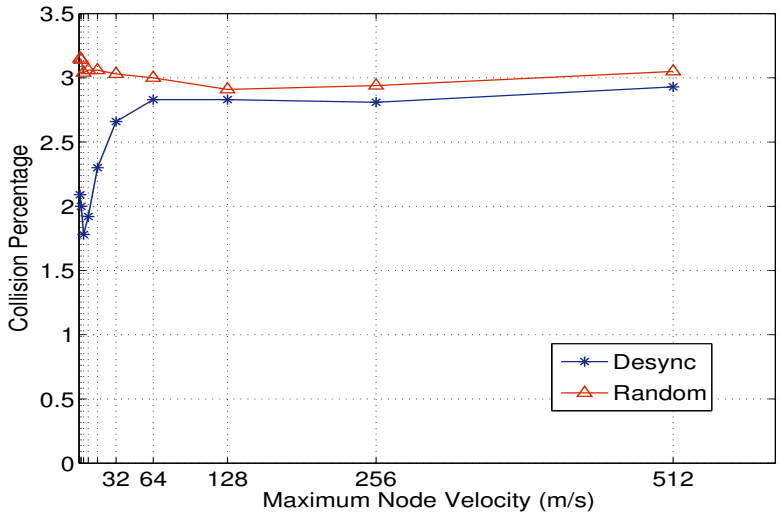


(c) $R = 25$

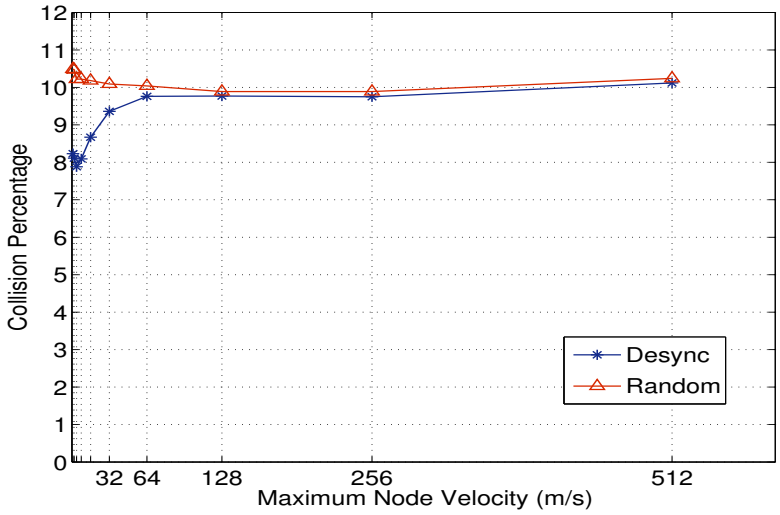
Figure 2.6: Desync spaceout over varying node velocity, and for different values of round size R. Each point in the graph is averaged over 20 simulation runs.



(a) $R = 100$



(b) $R = 50$



(c) $R = 25$

Figure 2.7: Message collisions observed over varying node velocity, and for different values of round size R. Each point in the graph is averaged over 20 simulation runs.

transmission range per slot. Furthermore, we note that the Desync algorithm performs better than randomized beaconing for network topologies that feature a maximum node velocity of less than 128m/s. Beyond this value, there is no notable difference between the normalized desync spaceout achieved by both Desync and randomized beaconing. This leads us to conclude that in highly mobile networks, resorting to randomized beaconing yields almost the same level of desynchronization performance as a Desync algorithm. An equally interesting observation relates to the registered message collisions in the network, shown in Figure 2.7. We notice that the message collisions for Desync initially increase and begin to level off at almost the same interval of maximum velocity as for desync spaceout, i. e., at 128m/s. An explanation could be that the node density on average increases as the velocity is increased up to a critical point (approximately half the transmission range), beyond which it stays almost a constant. The collisions recorded by randomized beaconing however, is nearly constant across the range of velocities that we evaluated. This is attributed to the uniform mobility of nodes that ensures that the average node degree in the network remains constant. Nevertheless, we note that the fraction of collisions recorded for the Desync algorithm is only marginally smaller than that achieved by randomized beaconing. The preference for Desync over randomized beaconing for the case of a mobile network, is therefore not very strongly motivated. In fact, for networks with very high node velocities, beaconing at random not only yields a performance comparable to Desync, but also precludes the need for any control overhead.

2.2.2 *CollisionDetect* vs Randomized Beaconing

We now characterize and contrast the desynchronization performance of *CollisionDetect* with a naive randomized beaconing mechanism that does not involve any form of collision detection and resolution. As a background, we assume that nodes have a ready access to their neighborhood size, and can collaboratively determine the maximum neighborhood density in their one-hop range. Nodes still operate with a fixed round size R. However, the number of transmission slots within a round chosen by a node n_i depends on its perceived maximum one-hop neighborhood density, denoted by d_i. Typically, a node chooses $\lfloor \frac{R}{kd_i} \rfloor$ slots within one round, where k is a positive integer. The slot allocation policy differentiates *CollisionDetect* from a randomized approach, i. e., with *CollisionDetect*, nodes start off by choosing slots at random, and change their schedule only when they receive collision notifications. In our evaluation, we study the influence of k on the performance of *CollisionDetect* and the randomized beaconing mechanism. We perform the study separately for static multihop and mobile networks.

1. Static Mesh Networks Figure 2.8 shows the normalized desync spaceout and the recorded message collisions for both *CollisionDetect* and the randomized beaconing mechanism. We use the same multihop network setting as in Section 2.2.1. The parameter k is varied from 1 to 10, and for every value of k, we average the results over 50 simulation runs. Figure 2.8a shows that while the randomized beaconing achieves a nearly stable value of desync spaceout across increasing values of k, *CollisionDetect* improves in its ability to desynchronize with increasing round size. However, the maximum absolute difference in desynchronization performance between *CollisionDetect* and the randomized approach is only marginal - approximately 3% of the round size. On the matter of message collisions, however, *CollisionDetect* outperforms the randomized beaconing scheme even for small values of k. Notably, for a setting of $k = 2$, which

is what the work by Motskin et al. [91] recommends, *CollisionDetect* achieves a reduction in collisions by more than a factor of 2 (5% for the randomized beaconing scheme versus 1.8% for *CollisionDetect*). Although, the relative benefits of using a mechanism such as *CollisionDetect* diminishes as k is increased. This is mainly because, at large values of k, the offered load on the channel gets lowered, such that even a randomized beaconing scheme would achieve a collision-free schedule. To summarize, *CollisionDetect* not only achieves a marginally better performance in desynchronization, but also reduces the number of collisions, particularly for small k.

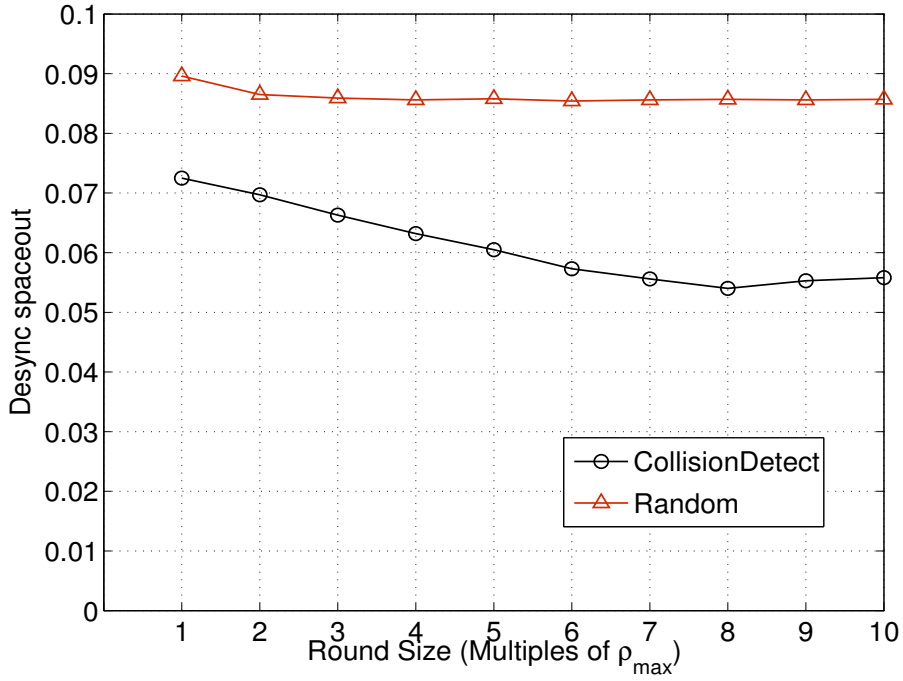
2. Mobile Networks Figures 2.9 and 2.10 show the desync spaceout and collision percentage respectively in mobile network topologies, for increasing values of maximum node speed. We find that the characterization of desync spaceout follows a trend similar to what we observed in Figure 2.6, i. e., the desynchronization performance levels off after the critical velocity of 128 m/s. Additionally, we also find the randomized beaconing scheme to marginally outperform *CollisionDetect* for all values of node velocities. On the matter of resolving collisions, *CollisionDetect* witnesses an increase in message collisions as the maximum node velocity is increased. This is typically attributed to the fact that collision notifications usually fail to get conveyed to the right set of nodes. Nevertheless, *CollisionDetect* still continues to perform better, although marginally than the randomized beaconing scheme. We therefore maintain that from the perspective of achieving desynchronization, a randomized beaconing scheme is a preferred choice for mobile networks.

2.3 Related Work

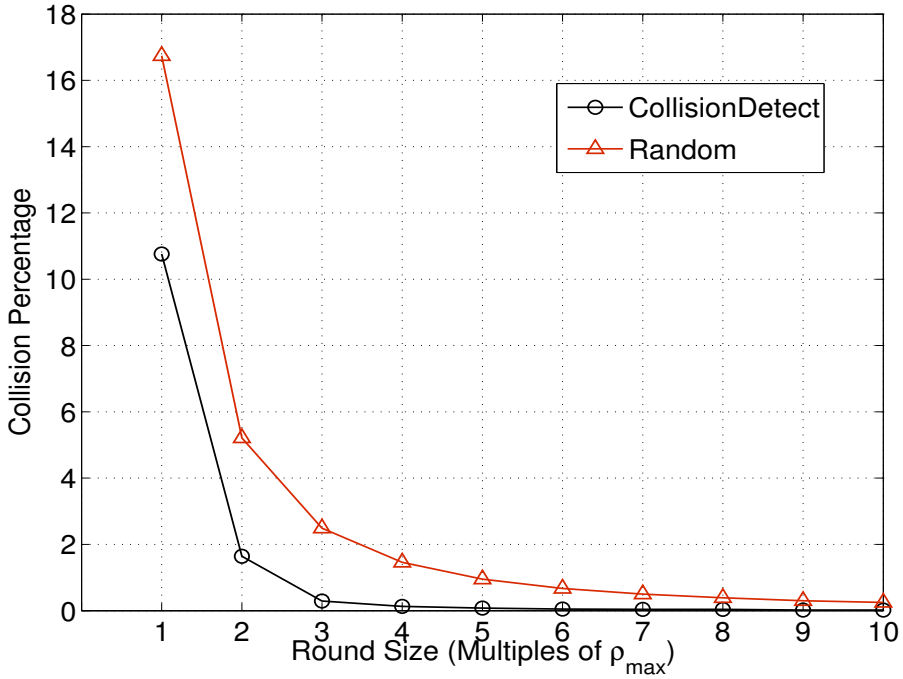
Desynchronization is considered an adaptive, self-organizing mechanism for scheduling collaborative node actions in a neighborhood. Typical application scenarios for desynchronization include scheduling of node sensing or wake up times in a neighborhood [4, 107] and coordinating radio communication in a one-hop range [51, 151]. In the following, we cover the related work on these problems in detail.

2.3.1 Collaborative Sensing

There have been numerous approaches to perform collaborative sensing [4, 13, 39, 41, 58, 107] in wireless networks. The objective of collaborative sensing is to primarily maximize the sensing coverage or minimizing sensing delay, while keeping the energy consumption per node to a minimum. However, most of the approaches either rely on an underlying network infrastructure [4, 58], and involve significant overhead on computation and communication [13, 107]. For example, the work by Keshavarzian et al. [58] is tailored to design a wake-up schedule that minimizes the end-to-end delay in a flow-based communication from sensor nodes to a sink. While the work by Baryshnikov et al. [4] is novel and lightweight, the proposed algorithm requires global time synchronization, and the presence of so-called artificial nucleating centre nodes that ensure continuity in the periodic wakeup pattern of neighboring nodes. The work presented by Cao et al. [13] motivates the need for a desynchronized state of node sensing in a neighborhood, and proposes a distributed algorithm that computes the sensing phase of every node in a manner that minimizes the average detection delay. However, the proposed algorithm incurs a significant

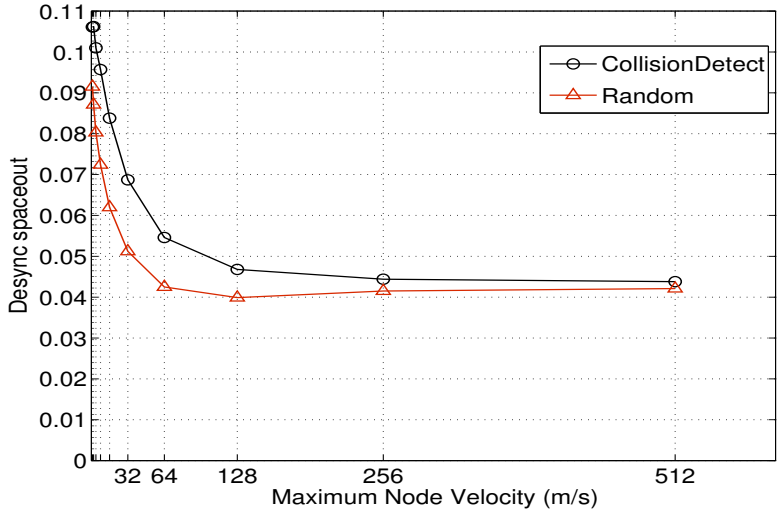


(a) Normalized desync spaceout

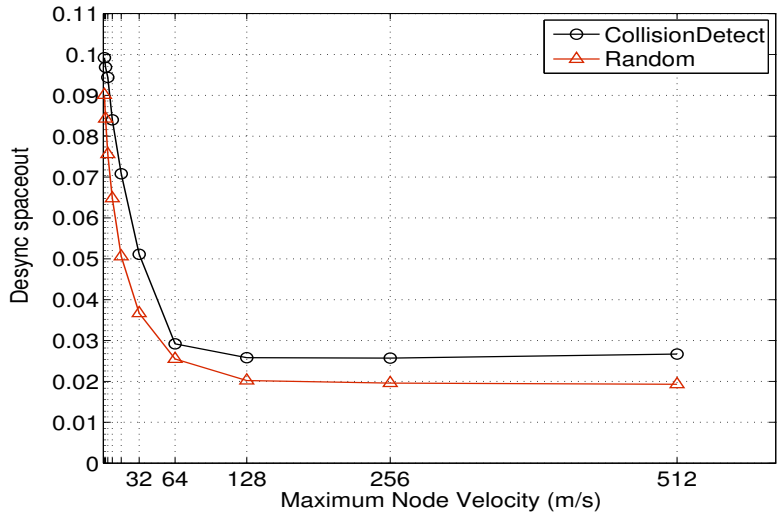


(b) Message collisions (%)

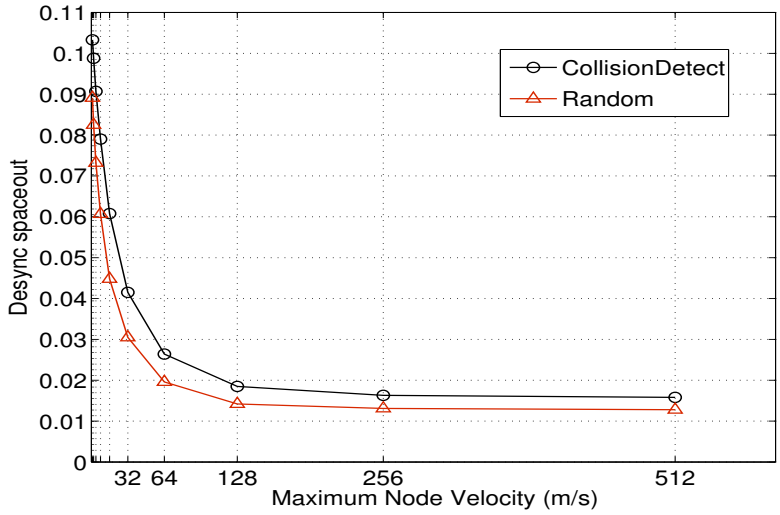
Figure 2.8: Desync spaceout and Collision recorded by *CollisionDetect* and randomized beaconing in static mesh networks. The round size for every node is set to $k\rho_{max}$, where ρ_{max} is the maximum local neighborhood density of a node. Parameter k is varied from 1 to 10.



(a) $k = 1$

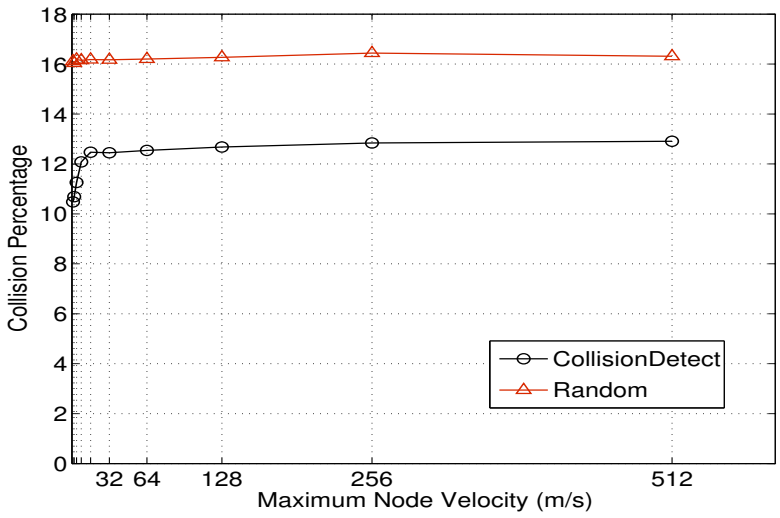


(b) $k = 2$

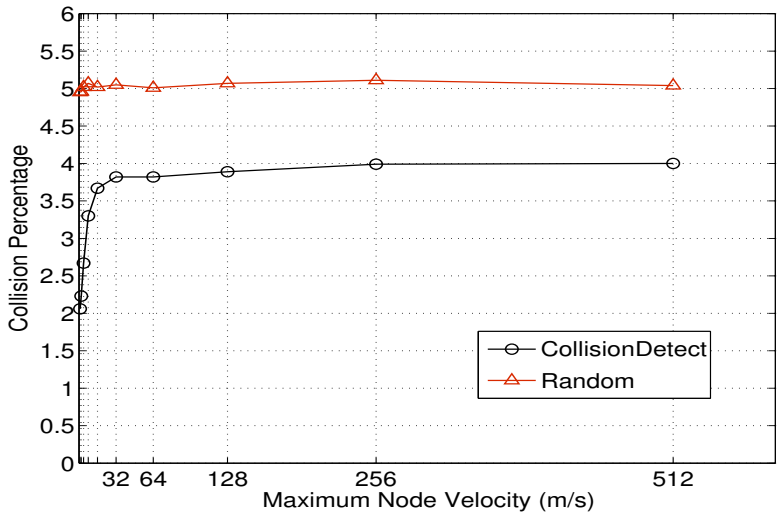


(c) $k = 3$

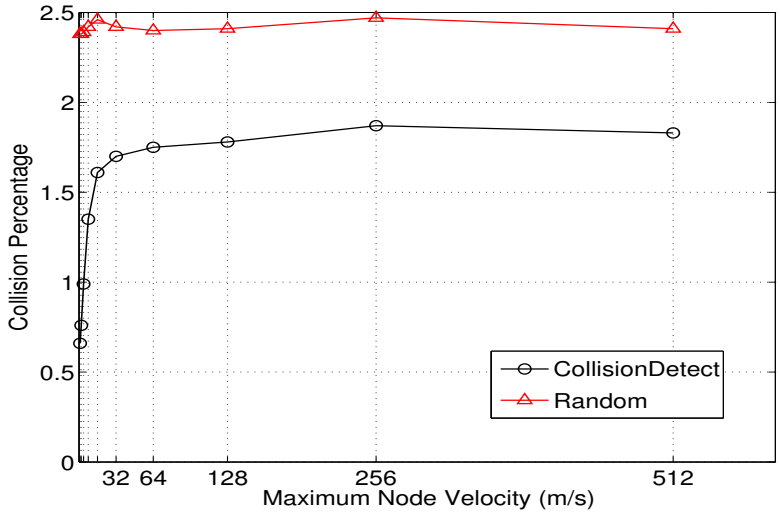
Figure 2.9: Desync spaceout over varying node velocity. Every node n_i transmits $\lfloor R \cdot k{d_i}^{-1} \rfloor$ times within a round, where d_i is the maximum node degree registered within its one-hop neighborhood. The results are averaged over 20 runs.



(a) $k = 1$



(b) $k = 2$



(c) $k = 3$

Figure 2.10: Message collisions observed over varying node velocity. Every node n_i transmits $\lfloor R \cdot k{d_i}^{-1} \rfloor$ times within a round, where d_i is the maximum node degree registered within its one-hop neighborhood. The results are averaged over 20 runs.

computational overhead. Alternatively, the work by Pemmaraju et al. [107] attempts to minimize the overall energy consumption in the network by partitioning the nodes into a collection of k-dominating sets. However, the implementation of the algorithm is inherently complicated and involves an explicit exchange of neighborhood information. While probabilistic approaches to maximizing sensing coverage [41] are simple and straightforward to implement, the resulting performance may not be optimal in the sense of minimizing average energy consumption per node. Lastly, an observation underlining these related works is that they have all been evaluated in network scenarios representing a static topology. To the best of our knowledge, a characterization of the proposed schedules under the influence of node mobility still needs to be addressed.

2.3.2 Communication Scheduling

Scheduling radio communication evenly between nodes in a neighborhood is complicated by the problem of message collisions, i. e., a receiving node cannot decipher messages from two neighboring nodes that transmit simultaneously. To this end, researchers have looked at approaches that maximize throughput [47, 151], ensure fairness [53, 98], or address a joint optimization of the two metrics [51, 123]. Desynchronization of node communication can been perceived as special case of fairness in which every node in a neighborhood gets an equal share of the channel. The afore-cited approaches either rely on explicit protocol mechanisms such as backoff timers [98], and the use of randomized mechanisms for transmitting messages on the channel [47]. In most cases, the communication mechanisms are either fixed, or adapt specific parameters according to changing dynamics such as long-term link changes and node mobility. In most cases, either the policy for ensuring fairness is integrated with the underlying mechanism, or the communication mechanisms are randomized, which may not strictly yield an optimal performance. In our study, we deviate from these approaches, and compare the performance of an explicit mechanism ensuring fairness, i. e., desynchronization, with respect to randomized communication mechanisms. On a secondary note, there are explicit mechanisms that handle the issue of the so-called hidden-terminal collisions [67, 112, 134] in multihop network topologies. Most of these mechanisms require time synchronization, and present an overhead on communication, i. e., some require explicit bitmaps of time slot allocations. In this chapter, we tackle the collision problem in a multihop setting, while precluding any of these requirements, and propose a novel mechanism to reduce message collisions.

2.3.3 Desynchronization Algorithms

Desynchronization in wireless networks was initially presented by Degesys et al. [19]. The proposed Desync algorithm worked on a clique of n nodes, and converged in $O(n^2)$ time. Extension of the Desync algorithm to multihop networks revealed interesting challenges, caused notably by message collisions [18]. To this end, a recent work by Motskin et al. [91] has revisited the desynchronization problem with a different approach. Specifically, the authors modeled desynchronization as a specialized case of a graph vertex coloring problem, and formulated an algorithm that converges in $O(\Delta \log n)$ time, where Δ is the maximum node degree in a network

of n nodes. However, the work does not characterize the performance of their desynchronization algorithm with regards to the well-spacedness of node communication. Moreover, the work presumes the existence of a collision resolution mechanism. The literature on desynchronization therefore, leaves the reader with open questions concerning the applicability of the algorithm to a dynamic network in which nodes may move at random.

2.4 Conclusion

In this chapter we studied the relative advantages of desynchronization over a randomized communication scheme in dynamic wireless networks characterized by node mobility. To this end, we formulated the desync spaceout metric, and characterized it under diverse topological scenarios that feature multihop static and mobile networks. Furthermore, we extended the current state-of-the-art on desynchronization by designing a collision resolution algorithm. Simulation results indicate that desynchronization should be favored mainly in static and sparse network topologies. In dense neighborhoods, and in scenarios marked by high node mobility, desynchronization achieves almost the same performance as a randomized communication mechanism. Additionally, our collision resolution algorithm, called *CollisionDetect*, reduces the number of collisions upto a factor of two over a randomized scheme.

To summarize, our findings in this chapter recommend the use of a randomized communication mechanism in the face of network dynamics. However, we are yet to address the issues of communication efficiency and fairness, which are heavily influenced by the offered load on the channel. In particular, having all nodes transmit randomly within the same time interval may not work efficiently in dense neighborhoods in which message collisions dominate most of the activities on the channel. This strongly motivates the need for a self-adaptive and self-regulating communication mechanism on all nodes that adjusts the transmission parameters according to the neighborhood density. However, knowing the number of neighbors in a dynamic, mobile wireless network is a non-trivial issue. Therefore, some form of neighborhood size estimation is required for achieving efficient communication in dynamic wireless network topologies. We address the combined problem of estimating neighborhood size as well as adaptive communication in the following chapter.

Chapter 3

Neighborhood Size Estimation in Dynamic Wireless Networks*

Our study of desynchronization in the previous chapter suggests the use of randomized beaconing mechanisms for dynamic wireless networks. The beaconing parameter for a node is typically a time interval over which it transmits at randomized phases. Alternatively, when nodes communicate in discrete time slots, the beaconing parameter is expressed as a probability value, i. e., nodes transmit a message in every slot with a given beaconing probability. The beaconing parameter has a strong influence on the communication efficiency i. e., the number of messages received by a node per time unit. Particularly, every node must set it in proportion to the number of nodes within its communication range, in order to maximize efficiency. While adapting the beaconing parameter is straightforwardly achieved for the case of a static and sparsely connected network topology, it poses critical challenges for dense and dynamic, mobile networks. Especially in mobile networks, the neighborhood of a node changes over time, which means that the beaconing parameter has to be adjusted continuously. The naivest way to acquire neighborhood size information involves registering all neighbors one-by-one. Therefore, the need for neighborhood discovery precludes the requirement for efficient communication. Furthermore, neighborhood discovery forms an important building block for most wireless applications such as routing in large-scale ad-hoc networks [93], timely dissemination of information in vehicular networks [117], and the estimation of churn in overlay networks [110]. Sometimes, discovery of wireless devices constitutes a major part of the application, such as in [96].

Quickly discovering neighbors requires an efficient beaconing mechanism in place. There is therefore, a cyclic dependency between the concerns of knowing the neighborhood size, discovering the neighbors, and communicating efficiently. As a solution, we propose addressing the combined problem of maximizing efficiency and neighborhood discovery by designing for a self-adaptive behavior within the network, wherein every node adapts its beaconing parameter in response to changes in its neighborhood size. For dense and mobile wireless neighborhoods, it is quicker for a node to estimate how many neighbors there are, than to register their identities. Thus, a continuous estimation of neighborhood size is a valuable piece of information that the

*A preliminary version of this work was published at *IEEE SASO'11*, Ann Harbor, MI, titled 'NetDetect: Neighborhood Discovery in Wireless Networks Using Adaptive Beacons', and authored jointly by V.G. Iyer, A. Pruteanu and S.O. Dulman.

beaconing mechanism could benefit from. The software that performs the estimation should not only be accurate, but also track changes in neighborhood size on account of intermittent link failures and node mobility.

We present the *Ne*ighborhood *Est*imator Algorithm (*Nest*), a distributed, online adaptive algorithm that estimates the neighborhood size in diverse network topologies ranging from static to mobile mesh networks. By dynamically changing the transmission parameter on every node at runtime, *Nest* seeks to maximize the number of message receptions, and thereby also the number of discoveries made per time unit. In order to enable adaptivity in the communication mechanism, each node gathers channel-utilization statistics at runtime and derives an estimate of the local neighborhood size. Additionally, *Nest* is designed to work with duty-cycled network layers, which are vital to the domain of low-power networking.

We evaluate *Nest* on different network topologies to study the latency in discovery; in particular, we look at fully-connected networks, and static as well as mobile mesh networks. Comparison with *Coupon Collector*, a state-of-the-art algorithm for neighborhood discovery [135] shows an almost similar convergence speed for neighborhood discovery on static networks. The main difference with that work, apart from targeting dynamic networks, is that our algorithm is designed to run continuously in time, removing assumptions regarding the start and termination of the algorithm. Additional simulations with mobile network topologies and evaluation on a topology constructed from real-world mobility traces, show that our algorithm is capable of discovering neighbors as efficiently as possible, from both a time and an energy perspective. As a last step, we also implemented *Nest* on a commercially available mote platform, in order to understand the ways in which real-world settings affect performance. Typically, the real-world environment presents certain channel-specific challenges, such as ambient noise and the capture-effect phenomenon, that adversely affect the accuracy of neighborhood size estimation. We designed and implemented *Nest* to work in spite of these channel effects, thereby making *Nest* a viable choice for practical implementation.

In summary, the main contributions of this chapter are:

- we introduce the *Nest* algorithm for estimating the size of the network neighborhood based on the distribution of packet inter-arrival times. The difference with similar, existing work [64] is that *Nest* is self-adaptive with respect to changes in local node densities, and operates in a completely decentralized manner on all nodes. *Nest* is shown to be capable of working regardless of the access strategy (slotted and pure Aloha and Carrier Sense), and duty-cycling strategy;

- we show through extensive simulations with both synthetic data and real-world mobility traces that the *Nest* algorithm is a practical solution for discovering neighbors under a range of realistic network conditions;

- we compare our work with the relevant state-of-the-art neighborhood discovery algorithm targeted at static networks and show that we obtain similar results, even if we address the significantly more difficult case of dynamic networks.

- we present results from a practical implementation on a real hardware platform, which shows that *Nest* is capable of estimating the neighborhood size with very good accuracy.

3.1 Background and Related Work

We address *the problem of estimating neighborhood size and discovering neighbors in a time-efficient manner for dynamic networks* characterized by node mobility and transient wireless links. These scenarios are typical of real-world applications such as vehicular traffic monitoring, crowd management, etc., in which not only the number of devices within a communication range varies over time, but also the membership of nodes as neighbors change constantly. The unavailability of global information significantly increases the difficulty of the problem. Most state-of-the-art algorithms are designed upon the unrealistic assumption of easy, accessible neighborhood information. This assumption is responsible for low quality of service and extensive infrastructure overhead, both in terms of hardware and software.

To support neighborhood discovery, wireless nodes typically broadcast short, so-called *beacon* messages, that serve as identification information. A node is said to have discovered its neighborhood after it has registered beacon messages from all nodes within its radio reception range. Much of the related work on message beaconing focuses specifically on either the beaconing policy, or mechanism, or a combination of both. Broadly speaking, we can classify the related work on neighborhood discovery and beaconing into the following three network scenarios:

3.1.1 Wireless Mesh Networks

Applications that run on wireless (and also possibly mobile) mesh networks typically employ some form of routing algorithm to forward data to a basestation. Neighborhood discovery assumes importance in the context of maintaining the most recent information regarding the best neighbor to forward data packets. Algorithms for neighborhood discovery include the following: *(i)* probabilistic beaconing mechanisms [9, 86, 135], *(ii)* beaconing mechanisms for *pair-wise* discovery between nodes [24, 54, 132, 150], *(iii)* group testing strategies [82]. Most of these works target static networks [15, 36], and fix either the beaconing or the radio-duty cycle parameters at deployment time, such that adaptivity under changing circumstances is not addressed. An exception is the work by Vasudevan et al. [135], that models neighborhood discovery as the classical Coupon Collector's problem. The proposed algorithm (which we call *Coupon Collector*) is adaptive, and converges in $\Theta(ne\ln(n))$ time on *Aloha* networks, n being the neighborhood size, and e is the base of the natural logarithm. However, the *Coupon Collector* imposes terminating conditions that apply only to static networks and requires that all nodes bootstrap at the same instant. Real-world networks however, are subject to intermittent node-to-node encounters, owing to node mobility, and also to different node bootstrapping times. These are relevant issues concerning neighborhood discovery in dense, ad-hoc wireless networks, which are not solved by the aforementioned algorithm. Our solution, *Nest*, has been designed with these constraints in mind and addresses them by employing adaptivity at runtime.

3.1.2 RFID Networks

Radio Frequency IDentification (*RFID*) networks are typically used for monitoring and surveillance activities. Such networks usually comprise a dense deployment of low-powered passive tags attached to objects, such as for example, inventory items in a warehouse. Monitoring the

objects entails estimating the number of passive tags in a deployment *via* the use of a set of *readers*. Algorithms for fast tag identification can be broadly classified as relying upon *(i) Tree-based querying methods* [69, 96], *(ii) Capture-recapture model* [125, 140], and *(iii) Probabilistic estimation* by observing communication statistics [37, 64]. The aforementioned works are inherently poll-based approaches, and also, the identification process requires coordination through the set of readers. Particularly, the use of probabilistic counting has received recent attention, with algorithms relying on a framed *Aloha* communication model to estimate the number of passive tags by counting either the number of idle slots, or collision slots, or position of the first idle slot in a frame, or a combination of these methods. We notice an overlap in these approaches with our *Nest* algorithm on the matter of counting neighbors using communication statistics. However, the use of dedicated readers for explicit coordination restricts the use of these algorithms to centralized, or at best, *quasi*-distributed systems. In contrast, *Nest* achieves neighborhood discovery on all participating nodes, rather than on a single node. Additionally, *Nest* works in a completely decentralized manner, by achieving an *all-to-all* coordination in a neighborhood of nodes. This is different from the *one-to-all* style of coordination implemented in RFID networks.

3.1.3 Vehicular Networks

Vehicular networks are considered as one of the emerging technologies for providing traffic safety and Intelligent Transportation Services (ITS) [133]. Critical challenges in vehicular networks include providing time-critical information with minimum delay [56], and adapting the beaconing parameter on every node [117], to account for topological changes. In contrast to conventional MAC approaches towards maximizing channel throughput, the beacon adaptation is expressed as an optimization problem that also takes into account the average information age [56], or message utility [121]. Nonetheless, the timely availability of neighborhood information, or at least, the estimated density helps the cause for efficient inter-vehicular messaging. In this context, our *Nest* algorithm could serve as an estimator of neighborhood size that can be applied to the current state-of-the-art algorithms in vehicular networks.

Our preliminary work [47], titled *NetDetect*, addresses the neighborhood discovery problem for dense and dynamic network topologies, and uses a probabilistic beaconing mechanism. In contrast to other probabilistic approaches [9, 86, 135], we resorted to an online estimation of the neighborhood size, based upon observed statistics of received messages, and used that information to adapt the beaconing probability. Note that *NetDetect* is somewhat similar to [40], which gathers statistics of idle periods on the channel in order to perform traffic-rate control. The *NetDetect* algorithm however, has certain drawbacks that we address in this chapter. First, instead of being limited only to *slotted Aloha* networks, *Nest* covers also a wider range of random access link layer protocols, such as pure Aloha and nonpersistent CSMA. Second, *Nest* proposes a significantly improved neighborhood size estimator by making better use of the available data. Finally, *Nest* operates with duty cycling radios, an important requirement for many practical situations where energy efficiency is a premium.

We observe a marginal, and yet noteworthy overlap in our work with that by Krohn et al. [66]. The proposed estimator, called SJDS, estimates the number of neighbors by observing statistics of channel busy times, and uses jamming signals for the purpose. However, the strict need

of time synchronization, explicit signaling of start and stop times, and the somewhat crude estimation approach restricts its applicability solely to static networks. In contrast, *Nest* is generally applicable to both static as well as dynamic networks, and precludes explicit control signaling.

3.2 Applicability to Systems

From an application perspective, approaches involving large-scale networks of wireless nodes relying on neighborhood information find use in surveillance and monitoring activities [92], vehicular traffic policing [74, 100] and human social networks [90]. Existing works on neighborhood discovery [24, 54] take a systems approach, and focus instead on achieving energy-efficiency. We observe that these protocols do not account for topological changes arising from node mobility, and therefore, do not adapt to network dynamics. Deviating from these works, we take an algorithmic route to the problem of discovering neighbors in a time-efficient manner. In our approach, we decouple the beaconing policy from explicit beaconing mechanisms currently in practice. Particularly, our *Nest* algorithm adapts only the beaconing parameter on every node, and therefore can be implemented atop existing beaconing protocols. The design of *Nest* makes no specific assumptions regarding the operating parameters of popular wireless technologies such as IEEE 802.15.4 and IEEE 802.11. However, it assumes that the participating devices have the same notion of a *message frame time*, and the ability to detect idle times on the communication channel. In general, the decoupling from implementation-specifics allows for the applicability of *Nest* across diverse hardware platforms, provided the nodes are able to communicate wirelessly in an ad-hoc manner.

3.3 The *Nest* Algorithm

We first describe the system model and the general design overview for *Nest*. We then detail the building blocks that constitute *Nest*, and also remark on extending *Nest* to work on other link-layers and low-power networks.

3.3.1 System Model

We represent time as a sequence of slots, each slot wide enough to transmit or receive a message. From a communication perspective, at any give time, a node transceiver can find itself in one of the three states: transmitting information, listening on the channel, or sleeping. We introduce the *Nest* algorithm ignoring the sleep state first - the influence of this state is discussed in Section 3.3.6. Regarding channel events, a *successful message* transmission implies that only one node in a given neighborhood was in the transmit state for the full length of the message. However, if two or more devices transmit at the same moment in time, a collision will occur and the neighboring nodes will not be able to distinguish any of the information being sent. We will revisit this assumption later in Section 3.5, in which we discuss the signal capture effect phenomenon. Additionally, we assume that the radio interface allows a node to perform carrier sensing, thus being able to distinguish if the channel is empty or a collision takes place. This allows us to extend our system model to CSMA-styled protocols.

3.3.2 Design Overview

As described previously, we approach the problem from the perspective of a periodic beaconing mechanism. Each node i has a local probability $p_i(t)$ with which it will transmit a packet at slot t. This probability will be dynamically adapted with time. The content of a successful packet consists of the *node identifier* (needed for identifying the neighborhood) and the *probability* $p_i(t)$ with which the node transmitted the packet.

It is well known that, for large neighborhood sizes, when the beaconing probabilities across nodes are independent and identical, the number of transmissions k in a given time slot can be approximated by a Poisson process, expressed as $P(k, \Lambda) = \frac{e^{-\Lambda}(\Lambda^k)}{k!}$. Here, Λ denotes the offered load on the channel, and is equal to np, where n is the neighborhood size and p is the beaconing probability on every node. As an example [86], for a clique of n nodes, for the slotted Aloha communication protocol, the maximum number of successful messages is attained for all nodes transmitting with the same probability $p_i = 1/n$. In this optimal case, the maximum throughput is e^{-1}, meaning that, on average in the best case scenario, a successful message occurs every 2.7 slots. For the case of pure Aloha, the maximum throughput is $(2e)^{-1}$, achieved for a probability $p_i = 1/2n$ (leading to a successful message every 5.4 slots).

These numbers represent also the goal for designing our algorithm. In other words, assume a network with unknown topology and an initial set of random beaconing probabilities for the nodes. The objective is to find an algorithm that will ensure that all nodes converge to the *locally* optimum beaconing probability. Particularly for a clique, this equals the reciprocal of their neighborhood size; so the algorithm will automatically estimate the local neighborhood size as well. The assumptions for *Nest* are kept at a minimum: apart from the beaconing mechanism described above, nothing else is assumed - no synchronization information, no additional messages being exchanged between the nodes, no central authority or special nodes in the network, and no information on average densities, topologies or mobility patterns.

Conceptually, *Nest* contains two mechanisms that work in parallel (in practice the mechanisms overlap, as detailed in the next sections). The first mechanism is a novel form of *distributed consensus*: nodes spatially close to each other will converge towards an average of each-others probabilities via a mechanism similar to gossiping [57]. The second mechanism will push these local values towards the reciprocal of the local density of nodes, using a *maximum likelihood estimator* based on observed statistics of the communication channel, achieving the desired goal.

3.3.3 Distributed Consensus

The distributed consensus ensures that nodes in a neighborhood transmit beacons with almost identical probabilities. Doing so constitutes the first step towards achieving the optimal beaconing probability on every node. To illustrate this, we consider a clique in which each of the n nodes initially holds a (random) probability p_i (i being the node index $i \in \{1, 2, \ldots, n\}$). Every beacon from node i carries the current value of p_i, i. e., $p_i(t)$. We denote the beaconing probability received by node i at time t as $p_i^r(t)$. As part of the *Nest* algorithm, node i updates its $p_i(t)$ based on the values heard from its neighbors in the past $p_i^r(t - t_0)$, $p_i^r(t - t_1)$, ... as a weighted

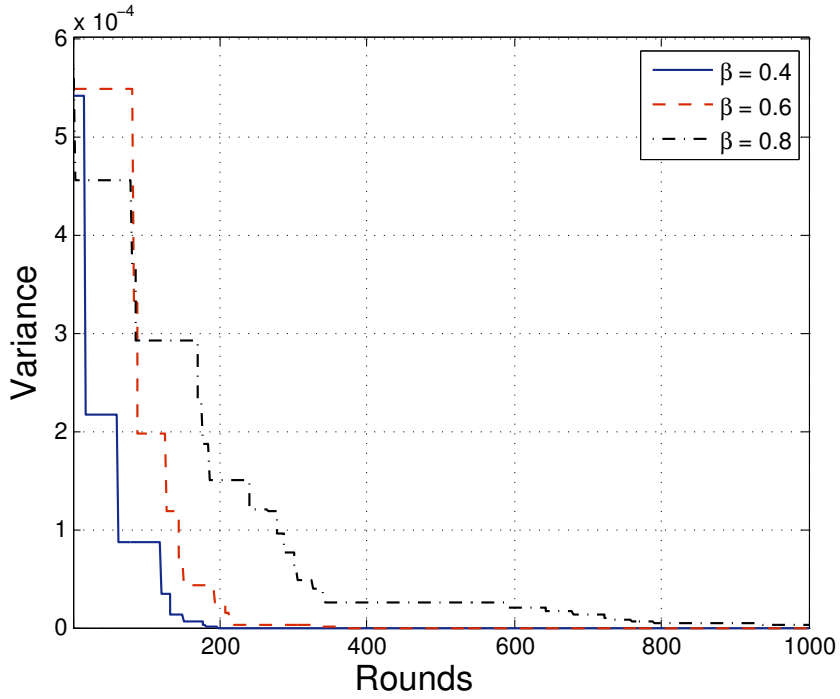


Figure 3.1: Distributed consensus in a clique of 100 nodes. The vertical axis shows the variance in beaconing probabilities, while the horizontal axis shows the evolution of time.

average. This mechanism maps onto a distributed gossip-based consensus, in which nodes exchange their beaconing probabilities, computing a new average on every beacon reception until they converge to a common value. Algorithm 1 presents the pseudocode for *Nest*, and line 9 in particular describes the consensus. The second term on the right, i. e., $\beta p_i(t) + (1 - \beta)p_i(t_{prev})$ represents a weighted average between a node's beaconing probability $p_i(t)$, and the probability $p_i^r(t_{prev})$ that it recently heard from its neighbor. The parameter β represents the consensus coefficient, and determines how quickly nodes converge to a common value.

The difference with the average aggregate presented in [57], is that the value to which the neighboring nodes converges cannot be predicted beforehand - it is a value situated between the minimum and maximum p_i, but the order in which the nodes advertise their values in the successful slots will influence it. Being from the same class as the gossiping algorithms, the consensus algorithm is assured to rapidly converge in fully-connected and mobile mesh networks [2]. Figure 3.1 shows the convergence of the distributed consensus component, for different values of coefficient β. The values are taken from [47], and show the exponential nature of the time taken for all nodes to converge to a common beaconing probability. We note that the smaller the value of β, the more pronounced is the mixing of beaconing probabilities, leading to a faster convergence. The convergence in static meshes will be slow at global level but fast for local neighborhoods - while in some algorithms this is a disadvantage, in our case it actually helps, being able to support networks in which the local densities are not constant. When evaluating *Nest*, in this chapter, we set β to 0.5.

Algorithm 1 *Nest*(P_i^m, D_i^m)

1: ▷ $p_i(t)$ is the *beaconing probability of node i at time t*
2: ▷ $P_i^m = \{p_i^r(t_1), p_i^r(t_2), \ldots, p_i^r(t_m)\}$, *the set of last m received probabilities*
3: ▷ $D_i^m = \{d_i(t_1), d_i(t_2), \ldots, d_i(t_m)\}$, *the set of m interarrivals*
4: ▷ α, β *are constants* $\in (0,1)$, $p_i^r(t_{prev}) = p_i^r(t_m)$ *is the previous probability*
5: ▷ *compute the current neighborhood size estimator*
6: $\hat{n}_i \leftarrow MLE(P_i^m, D_i^m)$
7: ▷ *compute current probability*
8: $\hat{p}_i \leftarrow 1/\hat{n}_i$
9: $p_i \leftarrow \alpha\hat{p}_i + (1-\alpha)\left(\beta p_i(t) + (1-\beta)p_i^r(t_{prev})\right)$
10: ▷ *return value*
11: p_i

3.3.4 Maximum Likelihood Estimator

In this section we introduce the derivation of the estimator for the local probabilities $\hat{p}_i$. The estimator works by taking into account the spacing in time between the successfully transmitted packets and the size of empty intervals on the communication channel.

To illustrate its underlying mechanism, we consider having the network in the steady state in which the probabilities on all nodes have stabilized to the same value. For the sake of simplicity, we assume the slotted Aloha communication model. If each of the n nodes transmits in the current slot with the probability $p_i = p$, it follows that the mean value of the Poisson process for having *only one node transmitting* in the current slot is $\lambda_{success} = np(1-p)^{n-1}$. By using the Poisson approximation to the binomial distribution we can describe the mean value for having an idle slot as $\lambda_{idle} = (1-p)^n$. Since the two Poisson processes describing successful transmissions and idle slots are independent, the process describing the case of a slot containing one message or none is also a Poisson process with a mean of $\lambda = \lambda_{success} + \lambda_{idle}$. Please note that as n can vary with time, λ is also a function of time $\lambda(t)$.

Let d_i be a random variable describing the time intervals between *relevant events* at node i (arrival of successful packets and length of empty slots). Based on the process from which the packets are generated, d_i is an exponential random variable, conditioned on the local probabilities $p_i = p$ via the parameter λ. Its punctual distribution function is thus $f_{d_i}(d_i|\lambda) = \lambda e^{-\lambda d_i} H(d_i)$ (where $H(d_i)$ represents the Heaviside step function).

When the network has stabilized (i. e., the probability on each node has converged to the same value), each node has access to a list of tuples $\{p_i^r(t_j), d_i(t_j)\}$ for a series of m consecutive relevant events. Here, $p_i^r(t_j)$ denotes the probability heard from the node that generated the last successful message, and $d_i(t_j)$ the distance, in slots, between the relevant events (j is an iterator over the last m relevant events, $j \in \{1, 2, ..., m\}$). Knowing the punctual distribution function for the $d_i(t_j)$ random variable, the question we ask is finding an estimator $\hat{n}_i$ for the number of nodes in the network that generated the outcomes $d_i(t_j)$. Once $\hat{n}_i$ has been determined, each node can adapt its probability of transmitting $p_i(t) = 1/\hat{n}_i$ (lines $6-9$ of Algorithm 1).

As a solution, we determine $\hat{n}_i$ using the maximum likelihood estimation (MLE) technique.

The likelihood function is:

$$L\left(d_i(t_j)|\lambda(t_j)\right) = \prod_{j=1}^{m} \lambda(t_j) e^{-\lambda(t_j) d_i(t_j)}. \tag{3.1}$$

The log-likelihood function is then given by:

$$\begin{aligned} \hat{l}\left(d_i(t_j)|n\right) &= \frac{1}{m} \log L\left(d_i(t_j)|\lambda(t_j)\right) \\ &= \frac{1}{m} \sum_{j=1}^{m} \left(\log \lambda(t_j) - \lambda(t_j) d_i(t_j)\right). \end{aligned} \tag{3.2}$$

The value $\hat{n}_i$ we are searching is the value that maximizes the $\hat{l}(d_i(t_j)|\lambda(t_j))$ expression - $\lambda(t_j)$ is a function of n thus $\hat{l}(d_i(t_j)|\lambda(t_j)) = \hat{l}(d_i(t_j)|n)$. $\hat{n}_i$ is to be found through the extremum values of the $\hat{l}(d_i(t_j)|\lambda(t_j))$ function, as a solution of $\frac{\partial \hat{l}(d_i(t_j)|n)}{\partial n} = 0$. A closed-form solution does not exist for $\hat{n}_i$, thus we need to employ a numerical estimation for the solution. $\hat{n}_i$ will be chosen as the solution that maximizes Equation 3.2. Correctly estimating the neighborhood size leads to the network converging to the maximum throughput rate, that maximizes the set of neighbors discovered.

It is proved that MLE estimators, when they exist and as the sample size grows large, achieve the Cramer-Rao bound (they are the unbiased estimators achieving the lowest possible standard deviation). Hence, this is the best approach one could choose from the class of unbiased estimators. The presented MLE is a one-shot estimator - in one step it offers a value with an accuracy that depends on the number of samples considered (the tuples $\{p_i^r(t_j), d_i(t_j)\}$). In the description of Algorithm 1, the notations P^m and D^m denote the buffers for these samples. As a minor observation, the constant α (line 9 in Algorithm 1) controls the trade-off between the rate of convergence for the consensus mechanism and the MLE estimator. It does not play a crucial role, and only moderately influences the speed of convergence. Therefore, we set α to 0.8 in our evaluation of *Nest* across all topologies and mobility models considered in this chapter. Another relevant observation is that our system model assumes that every node i starts off *Nest* with a random beaconing probability p_i. Since p_i could be drawn uniformly from the range $\{0, 1\}$, it is likely that the channel load $\Lambda = \sum_{i=1}^{N} p_i$ exceeds the operating limit for *Nest*, i. e., no channel statistics could be drawn. We fix this problem through the use of a *multiplicative decrease* policy, wherein a node i halves its beaconing probability if it has not received a message for e/p_i time slots. Every node repeats this step until the offered load on the channel permits the operation of *Nest*.

Figure 3.2 highlights the performance improvement of *Nest* over its previous counterpart, i. e., *NetDetect* [47] that uses received message interarrivals only. Specifically, we plot latency figures, i. e., the time taken by the algorithms to estimate the neighborhood size. We compare *Nest* against *(i) NetDetect* and *(ii)* a variant of *Nest* that uses only idle slot interarrivals, called $Nest_{idle}$. The reduced latency values for *Nest* is mainly attributed to the fact that it combines the distribution of idle and received slot interarrivals, and therefore, acquires more channel information per time unit. Furthermore, the inclusion of idle slots interarrivals also mitigates a form of estimation inaccuracy witnessed by *NetDetect*, that has been explained in [64].

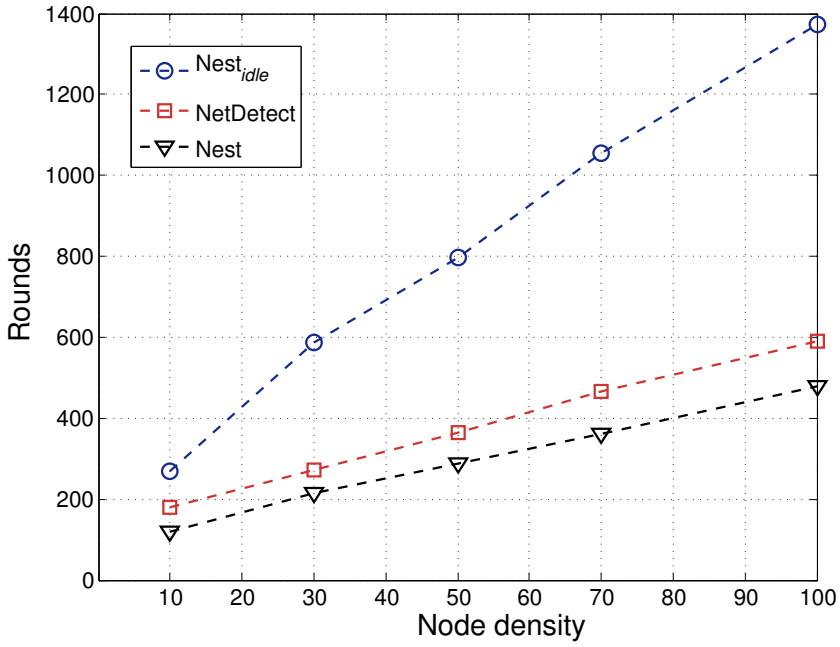


Figure 3.2: Comparing estimation latencies of different estimators ($Nest_{idle}$, *NetDetect*, and *Nest*), across varying neighborhood sizes.

3.3.5 Extensions to other Link Layers

We have so far, described *Nest* for the slotted Aloha protocol. In this subsection, we extend *Nest* to work on other random-access protocols, pure Aloha and non-persistent CSMA.

Pure Aloha A pure Aloha network does not discretize time into slots; instead nodes send messages as soon as they have data to transmit. The easiest way to model the throughput, is to consider time conceptually divided into slots of width τ, wide enough to accommodate one packet. Considering a neighborhood of n nodes, each broadcasting with probability p, let $T(t)$ and $\bar{T}(t)$ respectively be the event of a message transmission and idle period at time t. The mean value for a successful message transmission becomes:

$$\begin{aligned}\lambda^{PA}_{success} &= P\left(T(t) \cap \bar{T}(t-\tau)^{n-1} \cap \bar{T}(t+\tau)^{n-1}\right) \\ &= \binom{n}{1} \cdot p(1-p)^{2(n-1)}. \end{aligned} \tag{3.3}$$

The idle time on the channel is exponentially distributed with the rate $\lambda^{PA}_{idle} = (1-p)^n$. The combined event of getting either one message or none is a Poisson process with rate $\lambda^{PA} = \lambda^{PA}_{success} + \lambda^{PA}_{idle}$. The maximum likelihood estimation of neighborhood size $\hat{n}_i$ proceeds exactly as described in Section 3.3.4 for slotted Aloha, using the expression of λ^{PA} instead of λ.

Non-Persistent CSMA The non-persistent CSMA protocol works as follows: nodes that have a packet to transmit sense the channel to determine whether it is busy. If the channel is idle then the transmission occurs. If the channel is busy, then the node backs off for a time

interval that follows a random distribution. Along the line of reasoning in Kleinrock et al. [63], let Λ be the offered load, i. e., the average number of transmission attempts per time unit, and a be the ratio of message propagation delay to the transmission time. The probability of a successful transmission can be expressed as $\lambda_{success}^{CSMA} = \frac{\Lambda e^{-a\Lambda}}{\Lambda(1+2a)+e^{-a\Lambda}}$. Note that this relates to a case of an non-slotted non-persistent CSMA. In the case of a slotted non-persistent CSMA, the probability of a successful transmission is expressed as follows: $\lambda_{success}^{CSMASL} = \frac{a\Lambda e^{-a\Lambda}}{(1-e^{-a\Lambda})+a}$.

In our particular design of *Nest*, beacons may be sent as broadcast messages that do not necessarily require *per-packet* reliability. We believe that backoffs offer limited benefits to the cause of efficient beaconing, and also complicate the modeling of transmission probabilities that are vital for the design of our estimator. To this end, we design *Nest* in a way that nodes abort transmission once they sense the channel busy; and do not backoff. Therefore, the beaconing probabilities on every node retain their independence. If p is the beaconing probability on every node, then the traffic offered in a local neighborhood of n nodes can be expressed as $\Lambda = np$. We have:

$$\lambda_{success}^{CSMA} = \frac{npe^{-anp}}{np(1+2a)+e^{-anp}} \tag{3.4}$$

$$\lambda_{success}^{CSMASL} = \frac{anp \cdot e^{-anp}}{(1-e^{-anp})+a}. \tag{3.5}$$

The maximum likelihood estimator in Section 3.3.4 can now employ $\lambda_{success}^{CSMA}$ or $\lambda_{success}^{CSMASL}$ to determine the neighborhood size estimate $\hat{n}_i$.

3.3.6 Duty-Cycling in Low-Power Networks

While the *Nest* algorithm is applicable to a large class of wireless networks, the case of low-power, Wireless Sensor Networks (WSNs) needs special addressing. Research on energy-efficiency in WSNs focus specifically on the issue of radio duty-cycling, i. e., turning the transceiver on and off at regular intervals. In order to encompass the WSN domain in our design space of *Nest*, we introduce the notion of *asynchronous* duty-cycling, i. e., nodes in a neighborhood switch their radios between a *sleep* and *awake* mode at different time instances. In the following we provide the derivation for slotted Aloha only. For the other communication models, the reasoning is identical.

The radio duty-cycle is characterized as a probability p_{awake}, i. e., nodes communicate in $1/p_{awake}$ slots (chosen randomly) and find themselves in sleep mode in all the others. A node transmits with the absolute probability p_{tx} and listens on the channel with a probability of $p_{listen} = p_{awake} - p_{tx}$. With such a model, the probability of a successful transmission can be modeled as $p_s = n \cdot p_{tx} \cdot (1-p_{tx})^{n-1}$.

From the perspective of a node, however, the probability that a message is received is additionally subject to the condition that it is awake, and listening. This means that the probability of a message reception, which we denote as p_{rx} can be modeled as follows $p_{rx} = p_s \cdot (p_{awake} - p_{tx})$. Along the same line of reasoning, the probability of an idle slot, assuming no sleep state can be modeled as $p_{idle} = (1-p_{tx})^n$.

With the duty cycling probability p_{awake}, the observation of an idle slot is again, subject to the event that a node listens on the channel, such that the resulting p_{idle} can be re-written as follows: $p_{idle} = (1 - p_{tx})^n \cdot (p_{awake} - p_{tx})$.

The MLE part of the algorithm can be used as defined in Section 3.3.4 by simply modifying the probabilities according to the formulas above. For example, for the case of slotted Aloha, the expression $\lambda = \lambda_{rx} + \lambda_{idle}$ becomes $\lambda = p_{rx} + p_{idle}$.

3.4 Evaluation

We assess the performance of *Nest* through extensive simulations on Matlab. We use the slotted Aloha model for communication, and structure our evaluation as follows: Firstly, we characterize the neighborhood discovery latency for fully-connected and multihop static networks. For the case of mobile multihop networks, we showcase the instantaneous estimation accuracy of nodes, for different mobility models, and from a topology constructed using real-world traces. We then present numerical results that explain the trade-off between estimation accuracy and the resulting latency, for varying values of the sample size, and radio-duty cycle. Lastly, we present simulation results that show the performance of *Nest* across varying values of radio duty-cycle.

We perform our simulations on neighborhoods of varying sizes, under both fully-connected and multihop settings. Typically, we look at two metrics for accuracy, namely the *relative error* in estimated neighborhood size and the corresponding *variance*. Since our research focuses on timely discovery of neighbors, we also observed the latency incurred by our estimator, i. e., the time taken by *Nest* to estimate a neighborhood size that is within 25% of the true value.

3.4.1 Neighborhood Discovery

We evaluate *Nest* on fully-connected and multihop mesh networks, and look at the evolution of the neighborhood discovery over time. We compare our results with that of the *Coupon Collector* [135] algorithm. The *Coupon Collector* algorithm operates in phases, with a node decreasing its beaconing probability progressively over oncoming phases. It additionally features a terminating condition based upon the number of unique neighbors gathered in every phase. Such an approach is applicable to static neighborhoods of nodes alone. In a mobile network, however, both the neighborhood size and membership changes over time, such that a terminating condition loses relevance in this context. Therefore, the comparison between *Nest* and *Coupon Collector* makes sense only for the case of static mesh networks.

Static Networks Figure 3.3 shows the evolution of neighborhood discovery in fully-connected networks (left) and multihop mesh networks (right), for neighborhoods of different sizes. For each value of neighborhood size, we also plot the discovery evolution in the optimal case (OPT), i. e., in which the beaconing probability of every node is set to the reciprocal of its neighborhood size. For the case of a fully-connected network, we notice that while the *Nest* algorithm is 14.3% slower than *Coupon Collector* in a neighborhood of 30 nodes, it is able to discover neighbors in a 100-node setup about 5.3% quicker than *Coupon Collector*. For the case of multihop networks, we tested with average node degrees of 15 and 30, and found *Coupon Collector* to outperform *Nest* in terms of discovery performance. However, it must be noted that with

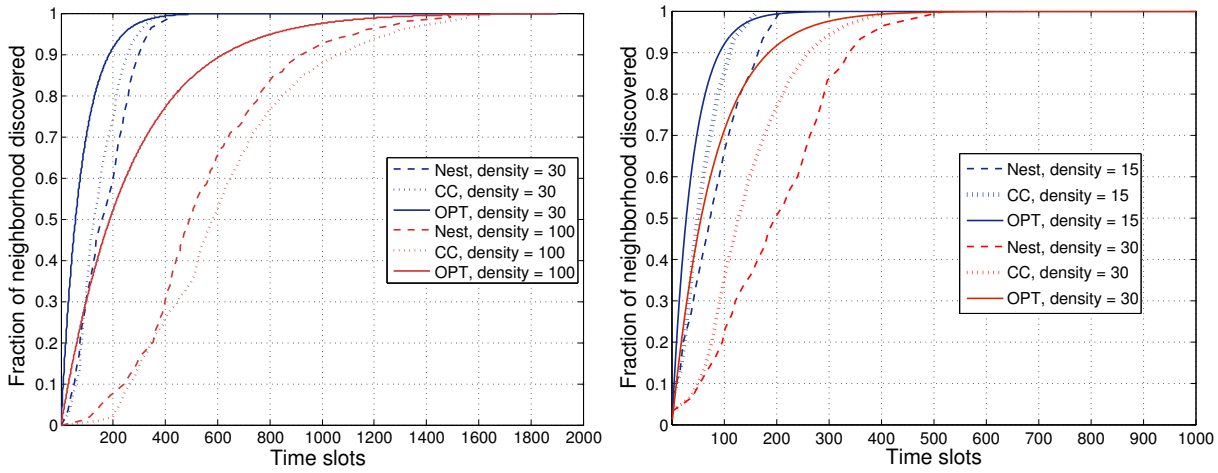
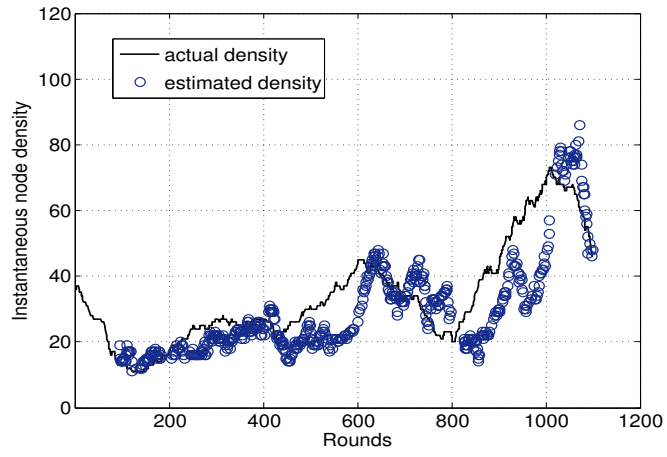


Figure 3.3: Evolution of Neighborhood Discovery for fully-connected (left) and multihop (right) mesh networks, for different values of local node density.

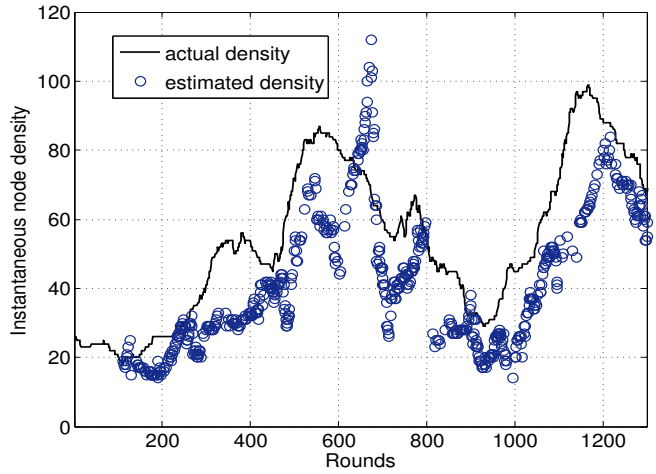
our simulations, *Nest* bears a greater initial overhead than *Coupon Collector* in bringing down the congestion level. Particularly, the *multiplicative decrease* policy of *Nest* operates over a larger time interval than *Coupon Collector*, thereby delaying the bootstrapping process for the estimator. Nevertheless, we notice that for different neighborhood sizes, both the *Nest* and *Coupon Collector* algorithms achieve neighborhood discovery in almost the same order of magnitude in latency as observed with an optimal beaconing scheme, i. e., $O(ne \ln n)$ as reported by [135].

Mobile Networks We simulated *Nest* on two different types of mobile networks, constructed using the *Random WayPoint* [11, 52] (RWPM) and the *SLAW* [73] mobility model. We chose these models in order to evaluate *Nest* on different types of mobile network topologies. In both the setups, we set the maximum velocity for nodes to 1.25 m/s (i. e., the average walking speed for humans), and the radio transmission range to 200m. In particular, the RWPM tends to generate a non-uniform distribution of node densities, i. e., nodes at the center of the topology have a greater neighborhood density than nodes at the edges. In contrast, the SLAW model is known to accurately capture the walking pattern observed in humans. Our objective is mainly to demonstrate that *Nest* can be applied to mobile networks, regardless of the type of mobility patterns observed.

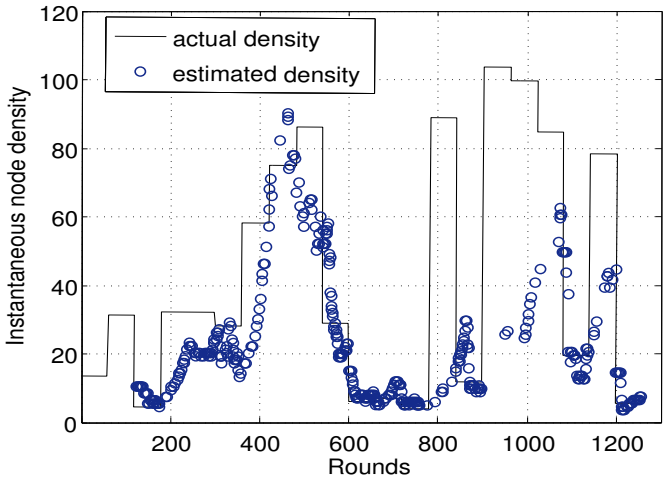
Figure 3.4 shows comparison of the true neighborhood size of a node, with the estimated neighborhood size, for the mobility models alluded above. We also performed a similar comparison on a mobile network topology constructed using real-world traces from the *Cabspotting* data set [109]. It can be observed that the estimated values reflect changes in the neighborhood size owing to node mobility. Although not shown here, we observed the relative error in estimation to be bounded to within 28% of the true neighborhood size, without significant deviation for different sample sizes for the estimator. We do, however, observe a phase lag between the predicted neighborhood size and the ground truth. This is attributed to the sample size used by *Nest*. In general, the larger the window size, the greater the observed phase lag. Nevertheless,



(a) RandomWayPoint



(b) Slaw



(c) Real-world traces

Figure 3.4: The graphs show the instantaneous neighborhood density estimation for a node, against the ground truth.

the results showcase the ability of *Nest* to adapt the beaconing probability on every node in response to changing neighborhoods.

3.4.2 Accuracy vs. Latency Trade-off

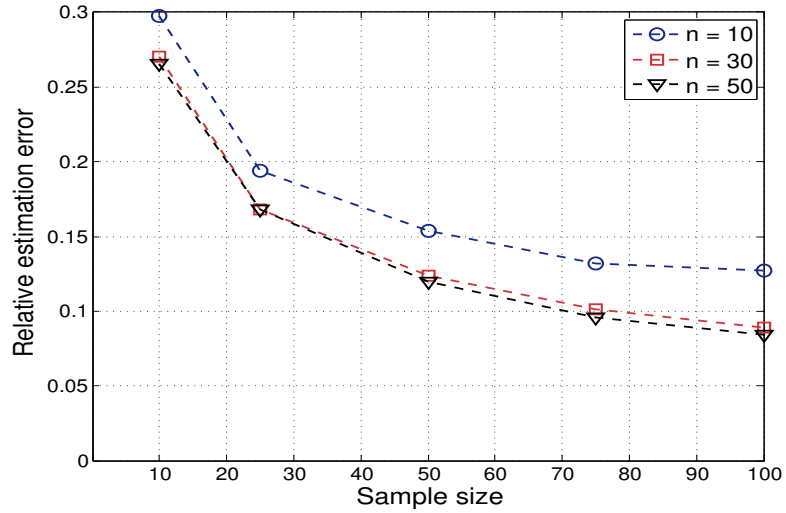
The performance of the *Nest* algorithm is characterized by a trade-off between two metrics, namely the estimation accuracy and the latency to converge to the estimated neighborhood size. These metrics are majorly influenced by the sample size of the interarrival distribution. A smaller sample size for *Nest* leads to faster estimation of neighborhood size, but doing so increases the error, and hence the variability of results. In contrast, a larger sample size improves estimation accuracy, but at the expense of an increased latency. We therefore, study this trade-off numerically in order to determine an appropriate sample size that achieves a certain upper bound on accuracy, without any significant compromise on delay.

We simulated *Nest* on cliques of sizes 10, 30 and 50, each time varying the sample size in the range 10, 25, 50, 75 and 100. Figure 3.5 shows both the relative error and the standard deviation (normalized to the mean) in the estimated neighborhood size. Initial observation reveals that the estimation error lies within 30% of the true value, even for neighborhood sizes as small as 10. Particularly, the relative error for a clique of size 10 is comparatively larger than for bigger cliques. We attribute this gap between the lines to the Poisson approximation of the interarrival distribution. In general, the variance in the estimation tends to decrease with an increase in sample size, and is nearly stable across different clique sizes. This is to be expected, as increasing the sample size leads the estimated values closer to the true neighborhood size. The latency increases linearly with the sample size, which is intuitive, given that the sampling process incurs a delay proportional to both the sample size and the offered traffic. Although not related to the discussion of trade-offs, Figure 3.5 shows that the latency increases also with the neighborhood size. This is attributed to the *multiplicative decrease* policy (cf. Section 3.3.4) that gets invoked more often for bigger neighborhoods, thereby contributing to an increased delay. As otherwise, and also claimed by [64], the sampling delay should not vary significantly for neighborhoods of different sizes, provided the offered load is kept relatively proportional to the node density.

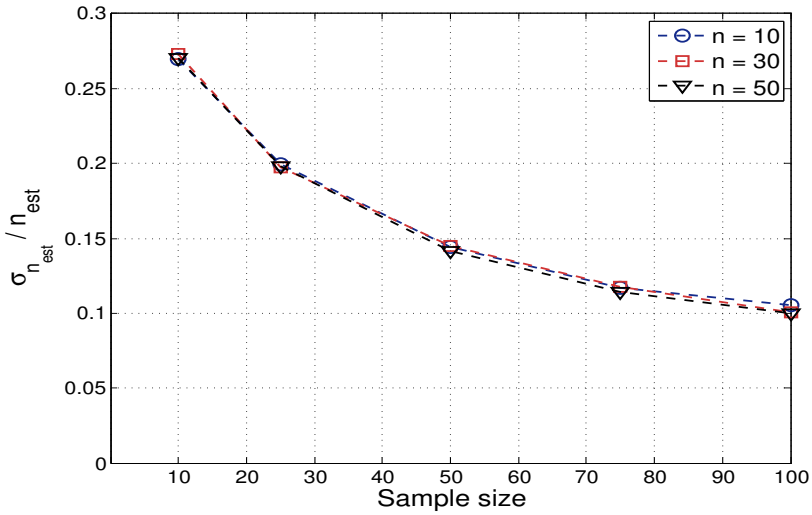
The values in Figure 3.5 suggest the use of a history sample size in the range of 25 to 50, such that the accuracy is bounded with 20% of the true value, without incurring a significantly long delay.

3.4.3 Duty-cycling Networks

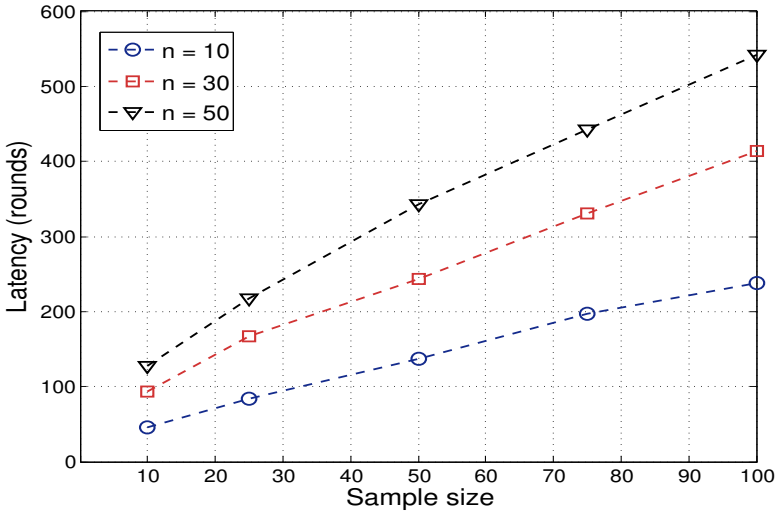
We simulated the *Nest* algorithm, with varying values for the radio-duty cycle parameter. As shown in Figure 3.6, both the estimation error and variance reduce with an increase of the radio duty cycle. We notice an exception to this trend for a neighborhood of size 10, i. e., the relative error and the corresponding variance increase when the duty cycle is varied from 0.9 to 1.0. A possible cause for this trend could be the Poisson approximation of the communication model. In general, the performance figures can be reasoned about as follows: at low values of duty-cycle (< 0.3), nodes tend to estimate their neighborhood sizes at different points of time. This follows from the asynchronous, independent nature of the sleep mode that we consider in our system model. The asynchronous estimation results in a temporary, non-homogeneous



(a) Relative Error

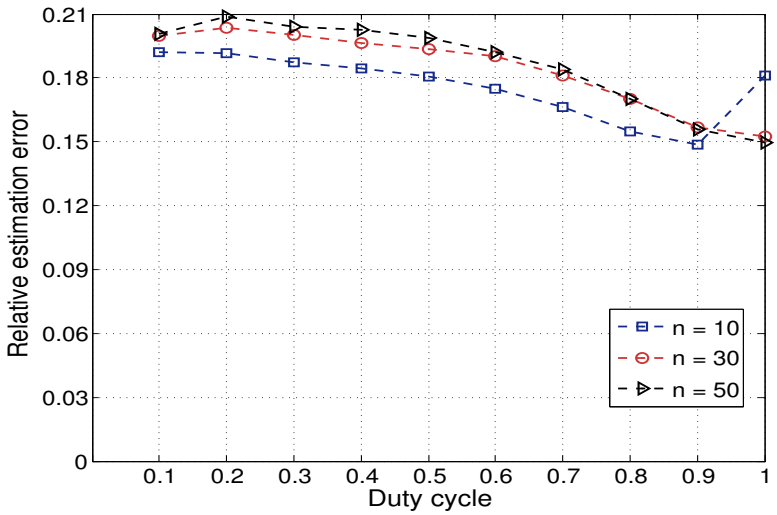


(b) Normalized Standard Deviation

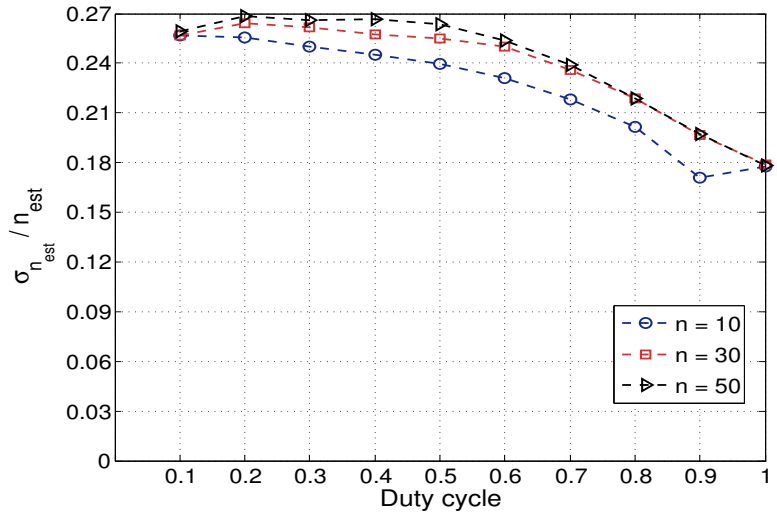


(c) Estimation Latency

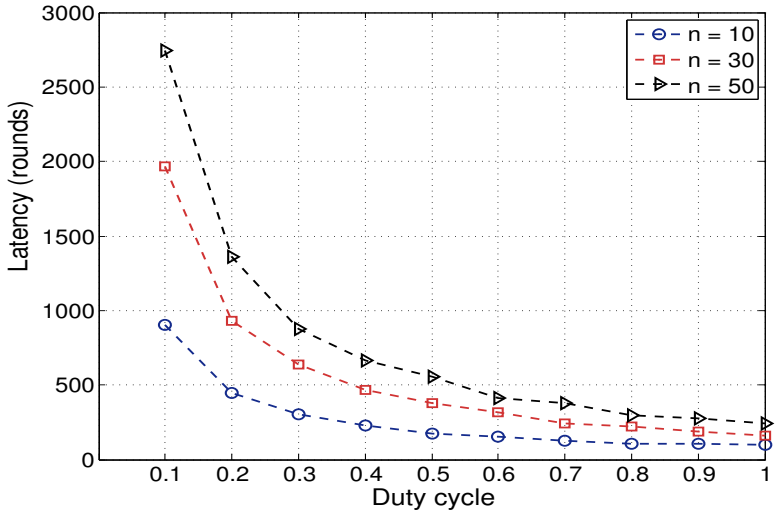
Figure 3.5: Estimation accuracy (top and middle) versus latency (bottom) for *Nest*, across varying sample sizes. Each point in the graphs was averaged over values collected from 50 simulation runs.



(a) Relative Error



(b) Deviation



(c) Latency

Figure 3.6: Relative error (top and middle), and Latency (bottom) for *Nest*, across varying values of radio-duty cycle, and neighborhood size. We set the sample size to 30. Each point in the graph was averaged over values collected from 20 runs.

distribution of beaconing probabilities. Although this is eventually smoothened out by the consensus component, the resulting average for the beaconing probability will bear an offset from its optimum value. We observe that this offset tends to decrease in magnitude as the duty cycle value is increased, as nodes tend to *mix* their probabilities quicker than at low-duty cycles, leading to a more consistent view of offered channel load and the resulting throughput. Correspondingly, the variance in the estimation decreases as nodes stay awake more often, mainly because the estimated neighborhood size is drawn from a more homogeneous distribution of beaconing probabilities. The latency for estimation decreases over increasing values of duty cycle, which is attributed to the observation that the longer the nodes stay awake, the quicker information regarding messages and idle periods is acquired.

To summarize, the results show that *Nest* works well, maintaining under 21% relative error, in scenarios where nodes duty cycle their transceivers to conserve energy.

3.5 Implementation on Hardware

In this section, we elaborate on specific implementation details and challenges for *Nest* on real hardware platforms. While simulation results shown in Section 3.4 reaffirm our expectations about *Nest*, there are certain factors that influence its performance in practical, real-world settings. In what follows, we describe three major factors that potentially influence the estimation accuracy, the beaconing convergence and the computation delay on the nodes. Firstly, note that while the maximum likelihood estimator is unbiased, the interarrival distributions that it relies upon are prone to errors, owing to spurious channel activity and the capture effect phenomenon [75]. We address this issue by showing that *Nest* works despite the occurrences of channel activity, when idle interarrivals are considered; the capture effect makes the use of received message interarrivals currently unfeasible. Secondly, the asynchronous operation of nodes introduces an issue related to convergence, which manifests under conditions of high channel traffic. However, this can be easily addressed with a sanity check on the nodes. Finally, the computation on the nodes needs to be carried out carefully in isolation from the communication process. This raises a concern of increased overall execution time, which we explain and address in this section.

The experimental results and findings are based upon an implementation on Tmote Sky motes [17], using TinyOS [76] as the software platform. For the purpose of inter-node communication, we used a *Pure* Aloha link layer. That is, nodes transmit randomly over a beacon interval, without carrier-sensing and backoffs. Our implementation of *Nest* adapts the beaconing interval, by estimating the neighborhood size concurrently on every node. The estimation runs asynchronously on all nodes; so no explicit coordination between neighbors is required. The neighborhood size estimation rules follow our reasoning presented in Section 3.3.5. We performed all our experiments on fully-connected topologies (cliques) of varying sizes, with the intent of showing an initial proof of concept for *Nest*.

3.5.1 Estimation Accuracy in Realistic Scenarios

The estimation accuracy for *Nest* depends on the correctness of the interarrival distributions that it relies upon. Here, the term *correctness* is used to describe how accurately the distribution

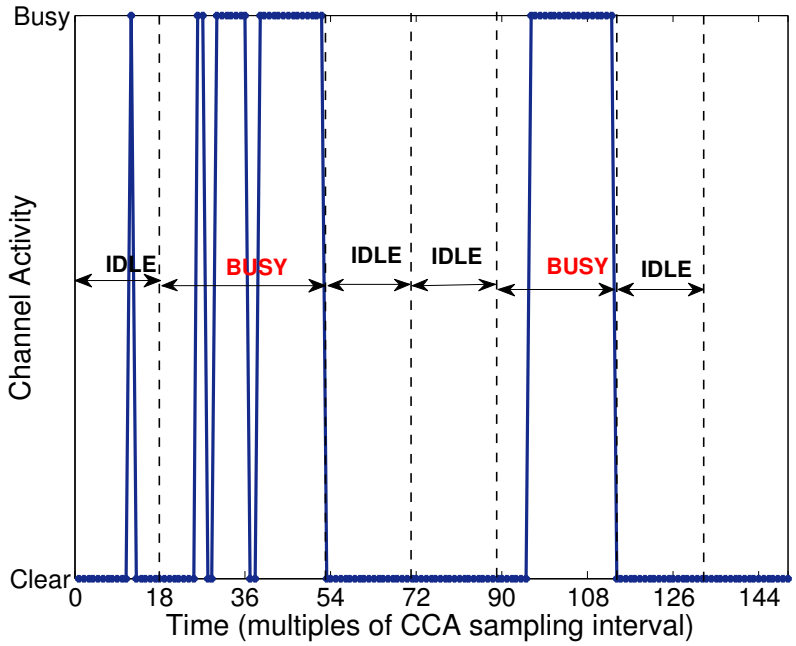


Figure 3.7: Determination of interarrivals of idle periods on the channel. The idle interarrivals is calculated by simply taking the sample difference between successive channel *IDLE* events.

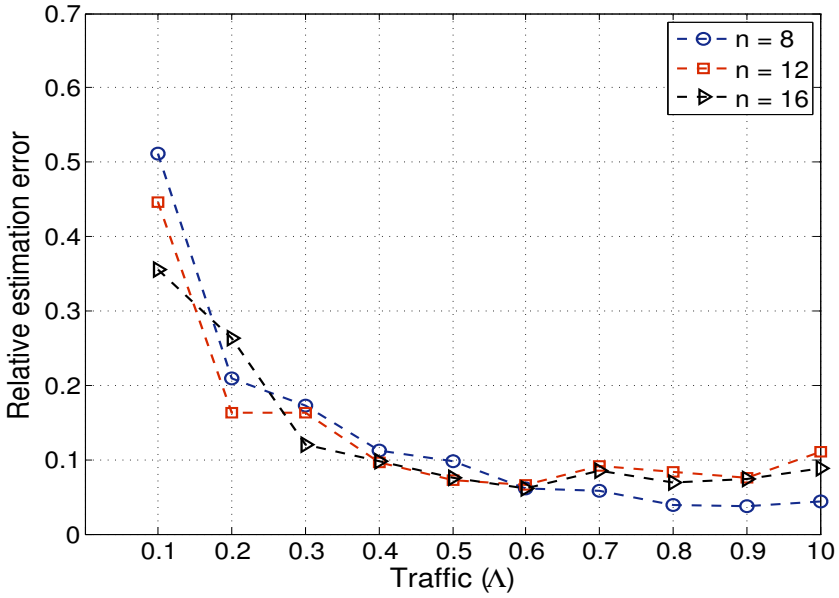


Figure 3.8: Relative error for $Nest_{idle}$ across varying values of offered traffic. Each line corresponds to a particular neighborhood size.

can be expressed as a function of the offered channel load Λ. In real-world settings, radio communication is influenced by phenomena such as interference and signal capture, which are not accounted for by our communication model. Therefore, the interarrival distributions that we observe from experiments are subject to irregularities. At present, we do not model the phenomena that induces these perturbations. Instead, we characterize these effects on the estimation accuracy for *Nest*.

Co-Channel Interference The communication channel at 2.4 GHz ISM frequency band is subject to ambient noise from WiFi access points and microwave ovens. This has a direct impact on the interarrival samples, i. e., they will be longer on account of the channel being frequently interfered. This would result in an overestimation of the neighborhood size. We conducted our experiments on zigbee channel 26, which is relatively interference-free compared to other zigbee channels, so that the effects of external interference on estimation were kept at a minimum. Nevertheless, we observed stray occurrences of channel activity from sources other than the nodes, which affected the estimation accuracy of *Nest*.

We address the problem of interference for the case of idle interarrivals alone. This is more a matter of convenience, as the effects of interference are mostly similar on both the received message and idle interarrival distributions. In order to be able to provide a distribution of the idle interarrivals, a continuous monitoring of channel activity becomes necessary. The Tmote sky platform features the CC2420 transceiver [45] that allows for a continuous sampling of the channel. By reading the CCA (clear channel assessment) pin of the transceiver at a rate of roughly 12 KHz, we can determine whether the channel was busy or clear for the duration of a packet transmission. In our implementation, nodes transmit messages with a fixed-size payload (45 bytes), that take approximately 1.44 ms to transmit over the channel. This translates to an observation window of roughly 18 CCA samples per channel event (i.e. message transmission, reception or collision) in order to infer either a *busy* or an *idle* slot.

Figure 3.7 shows a representation of channel activity using CCA samples. The values were taken from an experimental setup using 8 nodes in a fully-connected network topology. Each node beaconed randomly within a time interval of 80 ms, such that the collective traffic $\Lambda = 0.1$. A node samples the medium continuously, and maintains a counter for the number of consecutive channel *clear* samples that it observes. A so-called *IDLE* slot is inferred when the counter value exceeds a threshold. In our implementation, we set the threshold to 15 samples. Once an *IDLE* slot has been inferred, the corresponding interarrival is notified to *Nest*, and the node resets the counter for inferring the next idle slot. If the current CCA sample indicates channel *busy*, then the counter is reset, and incremented later when a channel *clear* sample is seen again.

In order to mitigate the *overestimation* problem, we allow the idle period inferring component to tolerate a stray *busy* sample (see Figure 3.7, between time 0 and 18). While such a recourse proves effective against few occurrences of spurious channel activity, it does not treat the problem completely. In fact, from our experiments, we have observed overestimated neighborhood sizes at low values of offered load, i. e., $0.1 \leq \Lambda < 0.5$. Therefore, as a more pragmatic alternative, we accept the problem as it is, and rely on the distributed controller to achieve convergence on the optimal beaconing probability. Figure 3.8 shows the relative estimation error across varying channel traffic Λ, considering only idle time interarrivals. We observe that the relative

error decreases with increasing values of offered load, i. e., it is less than 11% for values of $\Lambda > 0.5$. Although the relatively larger estimation errors at low traffic might seem problematic, it must be noted that the controller continuously adapts the beaconing probability, and leads it towards the optimal value. For example, the relative error at $\Lambda = 0.1$ is approximately 0.5; as in nodes overestimate their neighborhood size $\hat{n}$ as shown: $\hat{n} = 1.5n$, n being the true neighborhood size. Therefore, the entire neighborhood switches their operating traffic to $\Lambda = n \cdot (2\hat{n})^{-1} = 0.33$. In this manner, the decreasing trend of relative estimation error over Λ guarantees the convergence for the controller. Note that we tested *Nest* only over traffic in the range {0.1,1.0}. Kodialam et al. [64] provide a detailed analysis on the operating range of statistical estimators of neighborhood size. In contrast, we propose an alternative in the form of supportive policies such as the *multiplicative decrease* to get the nodes within the operating range for *Nest*.

Capture Effect and Estimation Accuracy The estimator for $Nest_{rx}$ (or equivalently, *NetDetect* [47]) uses interarrivals of received messages, and thus assumes a simplistic communication model, in which a packet collision always occurs when two or several message transmissions overlap in time. In reality, however, the message reception on a node is influenced by the interarrival time between overlapping packets, and their relative signal strengths. A signal capture is said to occur when two overlapping signals arriving at the receiver differ in their signal strength, such that the receiver locks onto the synchronization bits of the *stronger* signal and processes the data payload that follows. The capture effect is typically encountered in transceivers that use frequency modulation techniques [75]. The phenomenon of signal capture has been characterized a fair amount in the literature [71, 72, 144]; in fact, the research efforts by Lu and Whitehouse [81] show that the capture effect improves the latency of disseminating information in a wireless network. In the context of estimating the number of neighbors, the capture effect tends to perturb the interarrival statistics collected locally on a node. Particularly, our communication model misinterprets a message received by signal capture as an event in which the channel witnesses only a single transmission. This translates to an increased receiver throughput and therefore, shorter interarrivals on average. As a direct consequence, shorter interarrivals lead to an *underestimation* of the neighborhood size. However, the estimator for *Nest* that uses idle interarrivals works in spite of signal capture occurrences, as both a collision and message reception register as channel *busy* events. It is therefore, important to study the impact of capture effect on estimators such as $Nest_{rx}$ that uses only received message interarrivals.

As a starting hypothesis towards understanding how signal capture affects the accuracy of $Nest_{rx}$, we note the findings from [144], that show that the probability of signal capture peaks at high values of offered channel load, and decreases as the load is further increased. This suggests that the estimator relying on received message interarrivals would suffer a loss in accuracy at channel loads > 0.5. To verify our understanding, we carried out experiments with neighborhoods of sizes 12 and 16. Figure 3.9 shows the relative estimation error for $Nest_{rx}$, across varying values of channel traffic Λ. We notice a sudden rise in the estimation error, at values of Λ in the range {0.8,0.9}. The relative error decreases thereafter, when the the traffic load is increased.

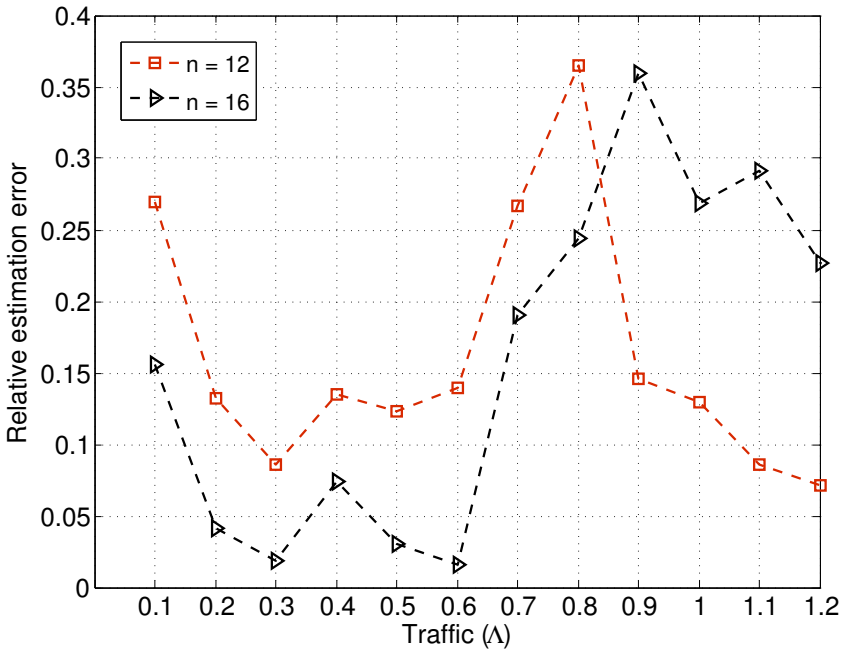


Figure 3.9: Relative Error for $Nest_{rx}$, for different values of offered load. Each line corresponds to a particular neighborhood size.

This has a direct relevance on the optimal beaconing convergence for $Nest_{rx}$. The distributed controller for the estimation will converge to the optimum beaconing value as long as the nodes offer a low channel load (< 0.5). However, at high contention, there is a definite possibility of an underestimation of neighborhood size, that forces the nodes to offer more traffic on the channel. As a result, the controller will exhibit a comparatively higher estimation error.

To summarize, the *Nest* algorithm in its current stage, is best implemented with idle interarrival distributions. The reason being that idle interarrivals prove effective even in the face of ambient noise and capture effect. Therefore, for the remainder of this section, we focus specifically on evaluating the estimator that used idle interarrivals, i. e., $Nest_{idle}$.

3.5.2 Asynchronous Computation and Consensus

Nest works in an continuous and iterative manner, by adapting the beaconing probabilities of the nodes each time, until they converge at a value for which both the throughput and discovery is maximized. The convergence to this optimal beaconing value is guaranteed as long as the nodes operate in synchronized time slots, and the components of *Nest*, i. e., the distributed consensus and the neighborhood size estimation execute synchronously on all nodes. However, in practice, nodes need not be always synchronized, and they usually estimate their neighborhood sizes at different instances in time. As a result, their computation and communication (consensus) phases do not perfectly overlap with one another.

The asynchronous in computation bears a significant influence on whether the nodes will converge to the optimal beaconing probability. In order to understand the problem in greater detail, we draw the reader's attention to the consensus component (cf. Section 3.3.3) of *Nest*, which causes neighboring nodes to converge to a common beaconing probability. Although the

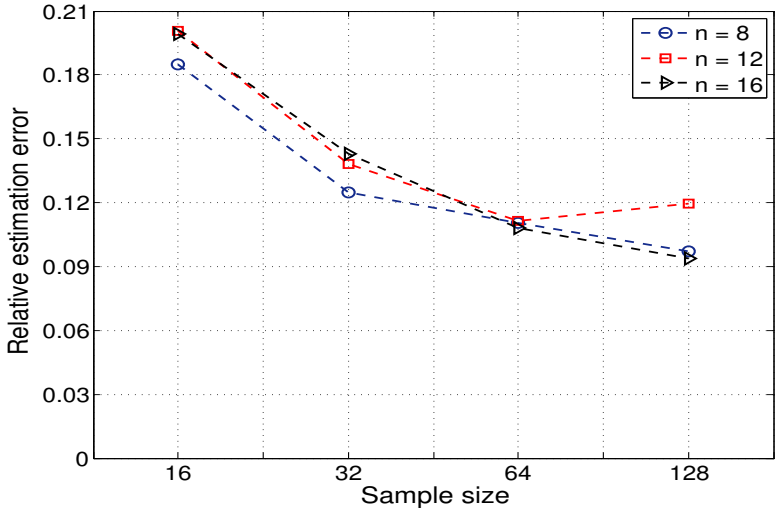
final value to which neighbors converge cannot be known beforehand, nodes that beacon more often are expected to influence the consensus more than their neighbors. This raises a relevant issue in a scenario of high channel contention, *i. e.*, $\Lambda > 0.5$. Typically, the estimator for a node causes a decrease in beaconing probability towards the optimal value corresponding to $\Lambda = 0.5$. However, given the asynchronous nature of computation, the neighbors of the node in question still beacon with higher probabilities. Therefore, the consensus component drags the node back to a state of high contention. Observe that this issue does not arise whenever nodes operate at low traffic condition, i. e., $\Lambda < 0.5$, the reason being that the estimator causes an increase in beaconing probability, that will subsequently pull the node's neighbors towards the optimal value.

The aforementioned problem can be mitigated by allowing nodes to refrain from participating in the consensus immediately following an estimation. In our implementation, nodes momentary disable the consensus, if the newly computed beaconing probability is smaller in absolute value than the previous probability by a certain difference. Since beaconing probabilities and intervals are inversely related, we can express the condition in terms of beaconing intervals. Let T_{old} be the beaconing interval for the nodes at high contention, and T_{new} be the newly computed beaconing interval for a node. The policy for disabling the consensus is activated when the following condition holds: $T_{old} - T_{new} > k$. In our implementation, we set k to 1, such that nodes do not participate in the consensus, if their newly computed beaconing interval is smaller than its previous interval by one millisecond. We evaluated $Nest_{idle}$ with this policy check in place, specifically under starting conditions of high traffic. In all the results, we observed that the nodes eventually converge to the optimal beaconing probability.

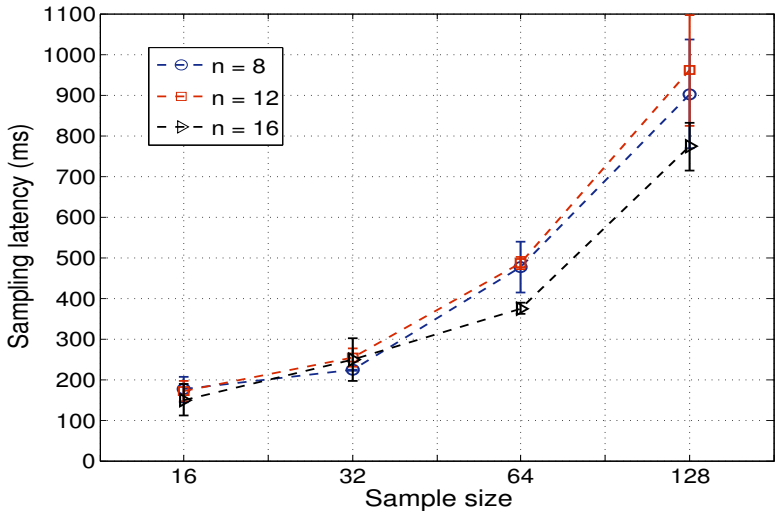
3.5.3 Accuracy *vs.* Computation Tradeoff

The performance of the *Nest* algorithm is characterized by a trade-off between estimation accuracy and latency, as described in Section 3.4. In this subsection, we address in particular, the *computation latency* on the motes. Note that this is different from the *sampling latency*, presented in Section 3.4. Intuition suggests that the computation time is a function of both the sample size and the search space of the neighborhood size. Therefore, the trade-off between estimation accuracy and computation time should follow a similar trend as covered under Section 3.4.2.

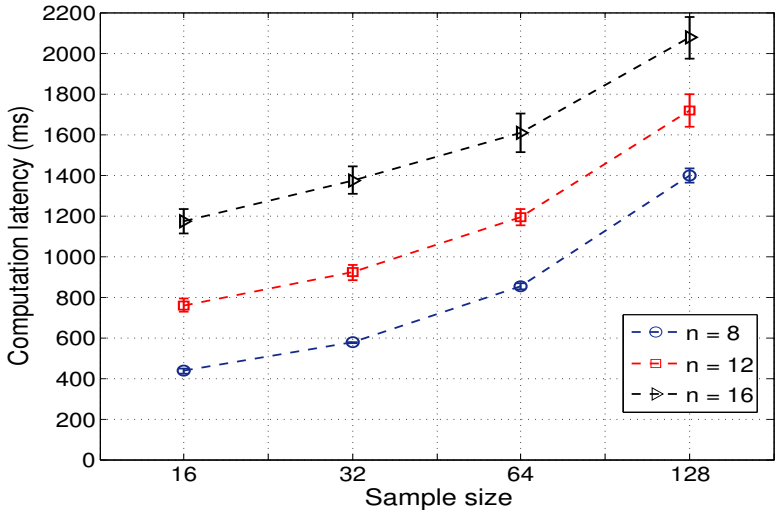
The maximum likelihood estimator for *Nest* involves operations on floating point numbers. For a microcontroller such as the *msp430* that is featured on Tmotes, this consumes a considerable amount of time, given that there is no hardware support for floating point arithmetic. We therefore resorted to fixed point representation of numbers and used lookup tables for mathematical functions wherever necessary. Additionally, we ensured that the computation and the beaconing process for *Nest* are carried out in isolation. This stemmed from the reasoning that if the beaconing process is repeatedly pre-empted by the computation, then the offered load on the channel does not reflect the beaconing values exchanged between neighbors. In fact, the offered load gets lowered, leading to an *underestimation* of neighborhood size. This presents a concern for the TinyOS platform that implements a default FIFO scheduling policy for concurrent events and tasks. Therefore, we implemented a simple scheduling policy that allows nodes to compute only when they are not transmitting packets. However, this introduces a



(a) Relative Error



(b) Sampling Latency



(c) Computation Latency

Figure 3.10: Estimation accuracy (top), sampling latency (middle) and computation latency (bottom) for $Nest_{Idle}$, across varying values of sample size. Each line in the figure corresponds to a particular neighborhood size.

waiting time for the computation, contributing to the overall estimation delay. In this regard, the computation latency shown in Figure 3.10 also includes the waiting time on account of the scheduling process.

Figures 3.10 shows the estimation accuracy, sampling and computation latencies respectively, across sample sizes 16, 32, 64 and 128. Note that we executed *Nest* with the control feedback loop in these experiments. The trend is similar to the one in Figure 3.5; increasing the sample size improves accuracy, but causes an increase in both sampling and computation latency. We notice that the sampling delay registers similar values across different neighborhood sizes, for a particular sample size. In contrast, the computation latency increases as both the sample size and the neighborhood size is increased. While the increase in latency over varying sample sizes is intuitive, the difference in latencies across varying neighborhood sizes warrants explanation. Our estimation technique uses a linear search in the space N of neighborhood size. This also involves computing the likelihood values $\forall n \in N$, such that the overall computation takes $\Theta(N)$ time. Additionally, our scheduling policy permits the search to operate in chunks of time that are spaced $2N$ milliseconds apart. The $2N$ milliseconds is derived from the optimal beaconing interval for a pure Aloha protocol. Therefore, the combined computation and scheduling delaying takes $\Theta(N^2)$ time. Nevertheless, we observe that the computation time is of the order of a few seconds, which is still viable for the case of mobile networks. It is however, possible to reduce the latency further by *(i)* packing more computation within one time slice, and *(ii)* replacing the linear search with numerical techniques such as bisection or the Newton-Raphson methods, as was done in [64].

Interestingly, the values in Figure 3.10 almost follow the trade-off characterization in Figure 3.5. In particular, the relative error is bounded between 10% and 15% for sample sizes in the range 32 to 64. This reaffirms our original choice of a sample size in a similar range, as mentioned in Section 3.4.

3.6 Conclusion

We addressed the problem of estimating neighborhood size in dynamic ad-hoc wireless networks. To this end, we designed and implemented *Nest*, a distributed algorithm that allows nodes to estimate their neighborhood size, and continuously adapt their beaconing probability in response to changing neighborhoods. Extensive simulations with *Nest* on diverse network topologies, and comparison with a state-of-the-art neighborhood discovery algorithm have shown that our algorithm not only tracks changes in neighborhood size owing to node mobility, but also discovers neighbors efficiently in time. We also validated our numerical results with a practical implementation of *Nest* on real hardware. In particular, we showed that the use of received message interarrivals for neighborhood size estimation is susceptible to the signal capture effect. In contrast, the use of idle periods on the channel was found to be more robust; we therefore recommend its use for practical implementations. The surprisingly simple nature of node-to-node interactions and the easy extensibility of the algorithm on duty-cycling networks, make *Nest* a viable and attractive choice for real-world networks. *Nest* is a generalized solution, can be applied to CSMA-style link layer protocols and also low-power, duty-cycling network of nodes

Our work on *Nest* presents a basic building block for networked embedded software, i. e., the knowledge of neighborhood density and the identities of neighboring nodes in real-time. *Nest* can therefore be used as a supporting service for other building blocks that require neighborhood context, such as routing and data aggregation.

We have so far, studied decentralized algorithms for estimation and adaptability at the level of a wireless neighborhood. In the following chapter, we reason beyond local neighborhoods, focusing instead on connectivity at the network level. Specifically, we present a decentralized algorithm that continuously estimates how long messages take to disseminate in a mobile network.

Chapter 4

Adaptive Online Estimation of Temporal Connectivity in Dynamic Wireless Networks

In previous chapters, we presented decentralized algorithms for adaptability and estimation in dynamic and changing wireless neighborhoods. We now extend our efforts to developing adaptive algorithms that reason beyond a wireless neighborhood. Particularly, we investigate the fundamental property of connectivity in dynamic wireless networks. This property is useful as knowing how reachable a node is from other nodes in a network provides a wealth of information regarding the delay in information propagation [34], the best route to send a message from a source node to a destination [87, 114], and to analyze the complex interactions among people in large crowds [101]. Traditionally, the connectivity between two nodes is represented as a spatial distance, and is expressed in terms of hop counts, i. e., the number of communication stages in between. The spatial representation of connectivity between nodes presumes the existence of a connecting path between them, at all times. However, in dynamic networks where links may fail intermittently, and in which nodes may move, the following challenges arise: *(i)* the connectivity between nodes changes continuously, *(ii)* the network is often not connected at any given point in time, and *(iii)* the temporal ordering of inter-node contacts strongly influences the connectivity between nodes. For these reasons, the conventional hop count-based metric for connectivity cannot be applied for the case of mobile networks.

In recent times, researchers have studied mobile networks [116, 128], generated real-world mobility traces [127] and proposed expressing connectivity in the temporal sense, i. e., how much time it takes for a message from node A to reach node B. Specifically, the shortest path between any two nodes in a network is expressed by a so-called *temporal distance*, which denotes the shortest actual time taken by the message to reach the destination from the source. The observed temporal distances in the network are used to compute aggregate metrics of temporal connectivity such as the average temporal distance, and the temporal efficiency [116]. In general, the knowledge of temporal network connectivity helps to track connectivity changes in real-time, and also the effects of node and link failures at distant parts of the network on overall connectivity. At present, however, the temporal connectivity measures are computed in

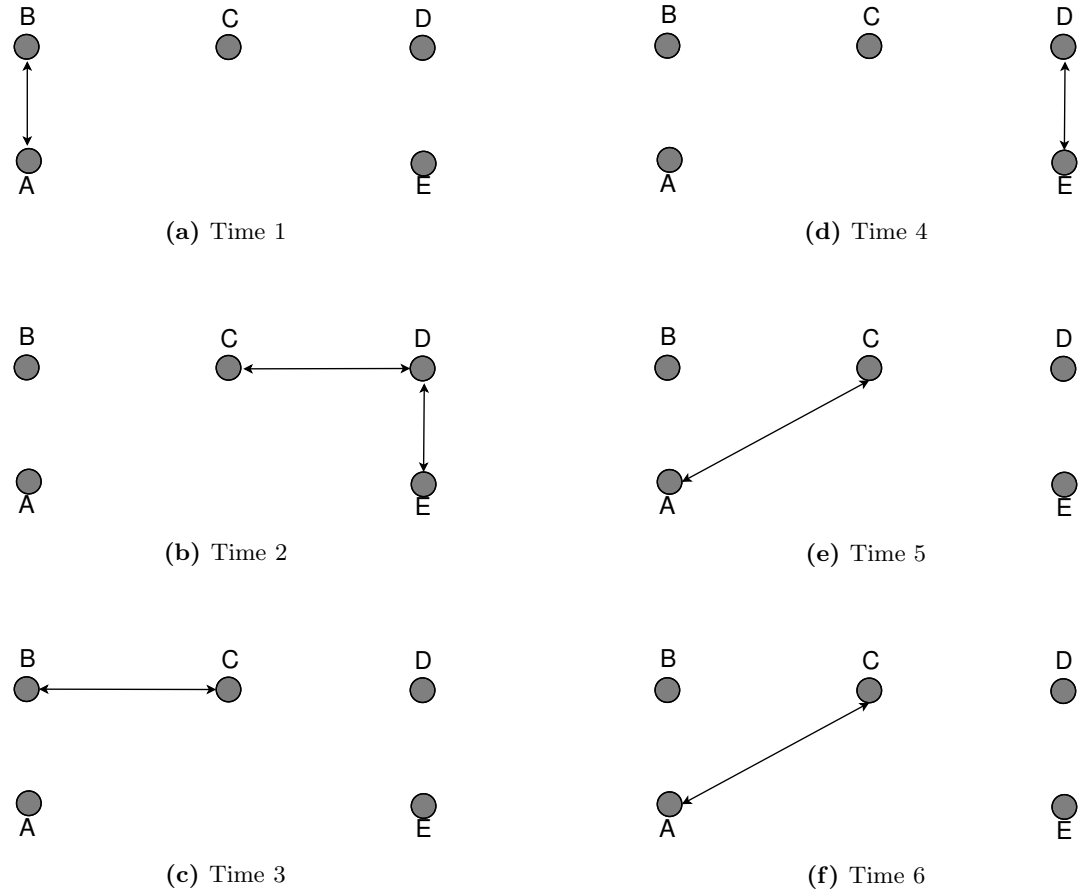
a centralized manner, using global knowledge about the network. While decentralized mechanisms exist such as in [124], they are restricted to observing distances to a reference node. It would be beneficial to have every node in the network reason about temporal connectivity in a decentralized manner, and in a *global* sense, relying only on local interaction with its neighbors.

This chapter focuses on the decentralized estimation of temporal connectivity on all nodes in a mobile wireless network. The main challenge here is to make the global knowledge of network connectivity available locally on every node. To the best of our knowledge, this is the first attempt to have every node reason about temporal connectivity over the *entire network*. Borrowing the metrics of temporal connectivity from earlier work [127, 128], we seek to determine the values at runtime. As a start, we collect distribution of temporal distances locally on all nodes in a mobile network. Since a global view on temporal connectivity is desired, it is necessary for nodes to communicate the locally collected distribution. Moreover, since a mobile topology changes with time, nodes have also to adapt and track changes in connectivity.

We present *PathDetect*, an online algorithm that estimates the shortest temporal paths in a dynamic wireless network. *PathDetect* works in a simple manner, and requires only local message interactions that can be achieved by a plain broadcast mechanism. Within the communication context of *PathDetect*, nodes act as message sources, while also forwarding messages from other nodes in the network. By annotating local timestamp information on every message transmission and reception, a distribution of shortest temporal path lengths can be determined locally on every node. Since a network-level metric for connectivity is desired, nodes exchange their values with one another, and construct a global view on temporal connectivity, with the help of distributed consensus [2, 57]. Using the global view, *PathDetect* estimates the so-called *temporal efficiency* that was introduced in [128] to characterize the average measure of connectivity between node-pairs.

Building a network-wide distribution of shortest distances incurs a considerable overhead in both local storage and communication. Therefore, we propose an alternative of approximating the observations by a Γ-distribution, and exchanging only the parameter values between nodes. In contrast, the simple approach of exchanging histogram bins is accurate, but vastly increases the overhead on storage and communication. In this chapter, we present cases in which the Γ-approximation works well, and mobility scenarios in which the histogram approach should be favored.

We evaluate *PathDetect* through simulations, and in particular, showcase its adaptivity to changes in network topology. Specifically, we compare the outcome of *PathDetect*, i. e., the decentralized characterization of temporal distances, to that of an offline computation. Simulations with varying node and link failures show that *PathDetect* approximates very closely (with a relative error of less than 10% in most cases) the globally computed value of temporal efficiency. Additionally, we investigate the more realistic case of disjoint networks, and show that *PathDetect* can track slow, gradual changes in network topology. More importantly, we address the issue of scalability, i. e., the problem of increased bookkeeping for very large networks. We show that for a large network, having fewer nodes as message sources vastly reduces the amount of information that is communicated, without any significant loss of accuracy (a relative error within 10% for most cases).



(a) Time 1 **(d)** Time 4

(b) Time 2 **(e)** Time 5

(c) Time 3 **(f)** Time 6

Figure 4.1: Link connectivity across 5 nodes over different time rounds. For simplicity, we assume a store-and-forward signalling scheme.

The rest of the chapter is organized as follows: Section 4.1 describes the system model and introduces the terms that define temporal connectivity. Section 4.2 describes *PathDetect* in detail. Section 4.3 presents the results of evaluating *PathDetect* under dynamic scenarios that include node failures, intermittent links, and mobility patterns. We also showcase the scalability of *PathDetect* in large networks. Section 4.4 deals with practical implementation challenges for *PathDetect*. Section 4.5 discusses related work, and Section 4.6 concludes the chapter.

4.1 System Model and Definitions

A mobile network can be modeled as a temporal graph *G(V,E,t)*, where V is the set of nodes, and E is the set of edges connecting the nodes at time t. We assume a discrete time model, i. e., the nodes move in rounds that represent the smallest unit of time. The temporal graph can therefore, be represented by an adjacency matrix that changes over time rounds. Figure 4.1 shows a sparse topology of 5 nodes, that changes connectivity over rounds. The temporal ordering of communication links ensures a path between at least a subset of node-pairs. For convenience, we assume that all nodes generate messages at the same round. Note that this is not a restrictive assumption for *PathDetect*, because *(i)* the temporal path lengths are recorded

relative to start of message generation, and *(ii)* we are interested in observing how quickly information could be disseminated, regardless of when nodes generate messages. Nodes within communication range at any given time round exchange broadcast messages with each other. We abstract away from the details of the underlying link layer, and instead assume that exchange of information between nodes is guaranteed. We revisit this point later on in Section 4.3, wherein we introduce node and link failures across the network topology.

A *temporal path* is defined as a sequence of tuples of the form $\langle t, n_i, n_j \rangle$, where a tuple is interpreted as follows: node n_i connects to n_j at time t. Note that the sequence of tuples increases successively in t, such that n_j of a given tuple equals n_i of the next tuple in the sequence. Moreover, n_i and n_j need not be connected at any point in time, so a temporal path could stall at a node until a link becomes available. For example, in Figure 4.1, although nodes A and C are disconnected, node A can reach node C through node B by the path $(\langle 1, A, B \rangle, \langle 2, B, B \rangle, \langle 3, B, C \rangle)$. The *temporal distance* between a source s and destination d is defined as the smallest number of rounds taken by a message originating at node s to reach d. Note that this is fundamentally different from the traditional definition of path length between any two nodes in a network. While the latter definition assumes a path existence between nodes at all times, the definition of a temporal distance relaxes this assumption. The temporal distance, denoted hereafter as $d_{i,j}$, refers to the earliest point in time that a node n_j receives a message from another node n_i. Messages are propagated from one node to another using a store-and-forward scheme. The storage time is one round, so information propagation is bound to one-hop distance in every round.

Every node observes the temporal distances in the network over a time window of τ rounds, as proposed by Scellato et al. [116]. Doing so restricts the maximum observable temporal distance to a value of τ. However, note that nodes may still not be connected by a temporal path within τ rounds; in such cases, we define the temporal distance to be ∞. In our evaluations, we set τ sufficiently high so that temporal connectivity between node pairs can be ensured, if they form a part of the same network.

Referring again to Figure 4.1, nodes A and B communicate in time round 1, so their temporal distance is one. Nodes C and D meet each other in time round 2, which is 2 rounds after they generated messages, so that the temporal distance between them is 2. In time round 3, when nodes C and B communicate, B could potentially receive a message from node D via node C. Therefore, the temporal distance from D to B is 3. Proceeding likewise for 5 time rounds, the temporal distance between the nodes can be summarized as shown in Table 4.1. Observe that the path matrix is not symmetric, e. g. , while the temporal distance from node A to node C is 3, it takes 5 rounds for a message to reach node A from node C.

The work by Scellato et al. [116] proposed a so-called Temporal Efficiency of a graph that characterizes how well connected a network is, in a temporal sense. The temporal efficiency is computed globally over an observation window τ, and is denoted by $E(\tau)$. It is computed as follows:

$$E(\tau) = \frac{1}{N(N-1)} \sum_{i,j} \frac{1}{d_{i,j}(\tau)}, \tag{4.1}$$

where N is the number of nodes in the network, and $d_{i,j}(\tau)$ is the temporal distance from n_i to n_j over a window τ. The Temporal Efficiency $E(\tau)$ ranges between 0 and 1. In the best-case scenario, i. e., when all the observed temporal distances are 1, $E(\tau)$ takes on a value of 1. In contrast, a value of 0 for $E(\tau)$ represents a completely disconnected network, in which the temporal distances between all node-pairs are ∞.

4.2 *PathDetect* Algorithm

We are interested in characterizing the distribution of temporal distances in a mobile network. While existing approaches [5, 116, 127] treat the problem offline so far, we propose a decentralized solution that tracks topological changes in real-time, and also the effects of node and link failures on overall connectivity. In what follows, we describe how a similar solution can be arrived at using a decentralized algorithm; in this case, *PathDetect*. To the best of our knowledge, this is the first attempt at characterizing the temporal connectivity of a network in a decentralized manner on all nodes. *PathDetect* relies mainly on broadcast message exchanges between neighbors, that convey temporal information concerning the shortest path lengths. As the distribution of shortest path lengths can differ from one node to another, we elaborate on the need of a distributed consensus component that allows the network to share a consistent global view over time. Finally, we describe how the temporal efficiency of a network can be computed in a completely decentralized manner on every node.

4.2.1 Gathering Temporal Distances

Allowing every node to know its temporal distance to other nodes incurs a considerable overhead on communication. A pragmatic alternative would be to locally record how reachable a node is from every other node in the network. This can be achieved using a simple broadcast communication mechanism that includes the necessary temporal information on message payload. Doing so sets the approach for *PathDetect*. Nodes running *PathDetect* periodically record the temporal distances from other nodes to themselves, and thereby build a local distribution of distances that varies over time. Tracking temporal distances requires some form of bookkeeping - in the case of *PathDetect*, every node n_i maintains a 3-tuple $< n_j, T_d^{j,i}, T_e^{j,i} >$ for every $n_j \in V$. Here $T_d^{j,i}$ represents the temporal distance recorded from node n_j to node n_i, and $T_e^{j,i}$ represents the time elapsed since node n_i updated its temporal distance from node n_j. Intially, $T_d^{j,i}$ is set to $\infty, \forall i \neq j, i, j \in V$. Specifically, when $i = j$, $T_d^{i,i} = 0$ and $T_e^{i,i}$ is the time elapsed since node n_i generated a message. The amount of bookkeeping required scales linearly with the number of nodes, which might prove unfeasible for large networks. Therefore, we advocate using only a subset of nodes as message originators. We revisit this point in detail in Section 4.3. Nodes include their bookkeeping information in broadcast messages, thereby communicating their temporal information to 1-hop neighbors. On receiving a message from node n_j, node n_i updates its temporal distance from n_j. Additionally, n_i may also update its temporal distance from a certain node n_k that reached n_j previously, as shown in Equation 4.2. Specifically, n_i compares its temporal distance from node n_k, i. e., $T_d^{k,i}$, with the sum of the temporal distance from n_k to

Table 4.1: Temporal distances between pairs of nodes that form the topology in Figure 4.1. Row and column labels correspond to node identifiers, such that the value captured at row i and column j represents the temporal distance $d_{i,j}$.

	A	B	C	D	E
A	0	1	3	∞	∞
B	1	0	3	∞	∞
C	5	3	0	2	4
D	5	3	2	0	2
E	∞	∞	∞	2	0

n_j and the time elapsed since n_k reached n_j. If the computed sum is smaller than the current value for $T_d^{k,i}$, both $T_d^{k,i}$ and $T_e^{k,i}$ are updated as follows:

$$\begin{aligned} T_d^{k,i} &\leftarrow \min(T_d^{k,i}, T_d^{k,j} + T_e^{k,j}) \\ T_e^{k,i} &\leftarrow 0. \end{aligned} \tag{4.2}$$

From Figure 4.1, the temporal distance from node D to node A can be reasoned about as follows: at time $t = 2$, node D communicates to node C, so that $T_d^{D,C} = 2$, and $T_e^{D,C} = 0$. Node C updates $T_e^{D,C}$ over time, so that at time $t = 5$, $T_e^{D,C} = 3$. Since node C can now communicate with node A, node A updates $T_d^{D,A} = \min(\infty, 2+3) = 5$. Proceeding likewise for all node pairs, every node obtains a list of temporal distances that correspond to their column vector shown in Table 4.1.

So far, the distributed algorithm allows nodes to have only a local view of the temporal distances in a network. In the following subsection, we explain how a global, and more coherent view of the temporal distances can be constructed using only local message exchanges between neighboring nodes. The distributed mechanism takes the form of a gossip-based consensus, as described in Algorithm 2.

4.2.2 Distributed Consensus

As explained in Section 4.2.1, every node has access to a local distribution of temporal distances. However, the distribution vary not only across nodes, but also change continuously over time. Therefore, it is necessary for *PathDetect* to not only build a so-called global distribution of temporal distances, but also to track changes in connectivity. For convenience, assume that every node maintains at time t, a state η_t that represents the distribution of temporal distances observed so far. We can express the global state of the network as a set of the individual states, i. e., $\eta_t = (\eta_t^1, \eta_t^2, \ldots, \eta_t^N)$. Our premise is that the constituent η_t^i terms are initially different, and that we would like nodes to achieve a consensus, such that eventually, the observed variance becomes negligibly small, i. e., $\sigma_{\eta_t}^2 \to 0$. In order to achieve consensus over η, each node n_i includes on its broadcast payload, its current state η_t^i. Additionally, node n_i updates its η_t^i as a weighted average over that received from its neighbor n_j, as shown:

$$\eta_t^i = \beta_{mix}\eta_t^i + (1 - \beta_{mix})\eta_t^j. \tag{4.3}$$

Algorithm 2 *PathDetect*(η_t)

1: ▷ η_t^i, *node i's local view of temporal connectivity at time t,*
2: ▷ $\eta_t = \{\eta_t^1, \eta_t^2, \ldots, \eta_t^N\}$, *distribution of temporal distances at time t,*
3: ▷ $N_i \rightarrow$ *Neighborhood set of node i*
4: ▷ $\alpha_{track}, \beta_{mix}$ *are constants* $\in (0, 1)$,
5: ▷ O_t^i, *the set of temporal distances observed by node i at time t*
6: ▷ *Exchange temporal connectivity with neighbors*
7: $\forall j \in N_i : \eta_t^i = \beta_{mix}\eta_t^i + (1 - \beta_{mix})\eta_t^j$
8: ▷ *Update temporal connectivity with local observations*
9: $\eta_t^i = \alpha_{track}\eta_{t-1}^i + (1 - \alpha_{track})O_t^i$
10: ▷ *return value*
11: η_t^i

Here, β_{mix} is the consensus coefficient ranging between 0 and 1. In general, the lower the value for β_{mix}, the more pronounced is the mixing of data between nodes, leading to a faster convergence. In particular, the consensus equation described above belongs to the class of gossiping algorithms as in [103], and is expected to converge rapidly.

Since the temporal distances can also vary over time, every node n_i also updates it state as a function of time as shown in Equation 4.4:

$$\eta_t^i = \alpha_{track}\eta_{t-1}^i + (1 - \alpha_{track})O_t^i. \tag{4.4}$$

Here, the term O_t^i denotes the temporal distances observed at time t. The term α_{track} is the weighted coefficient ranging between 0 and 1. Equations 4.3 and 4.4 correspond to lines 7 and 9 of Algorithm 2 respectively. Both the operations run concurrently on a node, such that their operations overlap with one another. The values for coefficients α_{track} and β_{mix} determine the accuracy to which nodes approximate the global network distribution of temporal shortest distances. The sensitivity of *PathDetect* to these parameters is elaborated upon in detail in Section 4.2.4.

4.2.3 Representing the Shortest Path Distribution

The term η_t introduced in Section 4.2.2 represents a distribution of temporal distances on every node. The easiest way to represent η_t would be to maintain a histogram of values. As mentioned earlier in Section 4.1, nodes record the shortest path lengths over an observation window τ. Therefore, a node n_i maintains a vector $\eta_i^t = (p_1, p_2, \ldots, p_\tau)$ of probabilities. Specifically, the value p_k indicates the probability that the temporal distance to node n_i is k. The probabilities are computed as weighted components over the complete histogram set. Having obtained the values for $(p_1, p_2, \ldots, p_\tau)$, the consensus (cf. Equation 4.3) and the time update (cf. Equation 4.4) operations are carried out independently on the vector elements. While the approach of separating temporal distances into histogram bins is very straightforward to implement, it presents an overhead on communication. This overhead can be improved by gossiping only parts of the histogram between nodes. For example, given a set of temporal distances such as $(0, 0, 0.1, 0.4, 0.3, 0.2, 0.0, 0.0, 0.0)$, it suffices to gossip only the values pointed by indexes 3 to 6.

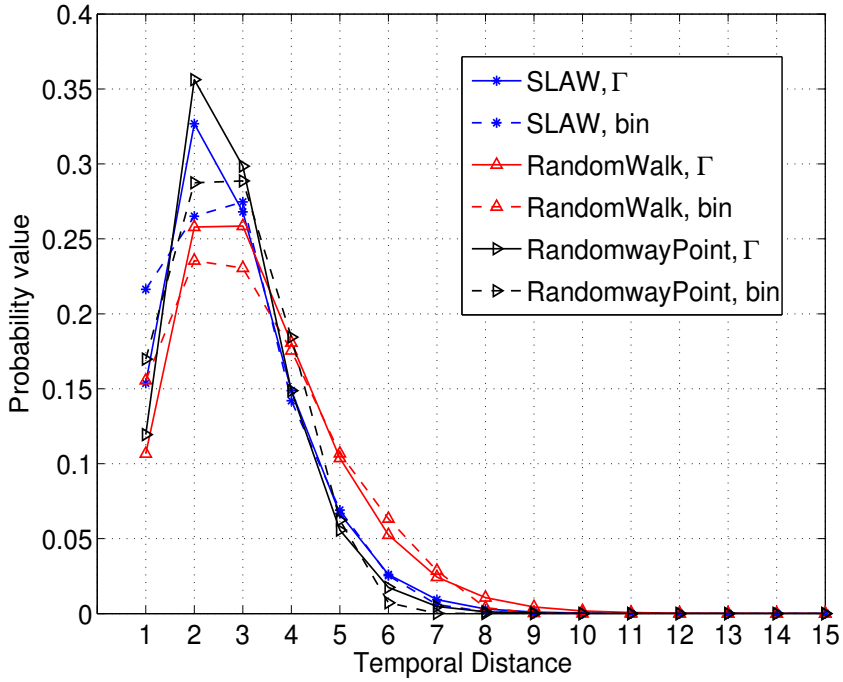
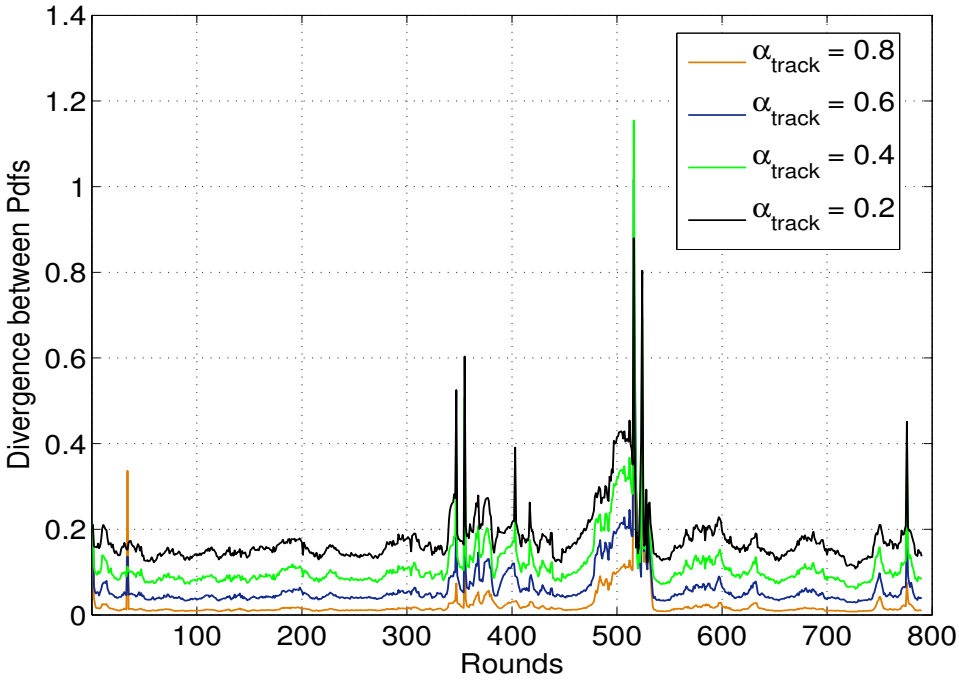


Figure 4.2: Distribution of Temporal shortest paths, for different mobility models. $\alpha_{track} = 0.6$, $\beta_{mix} = 0.8$.

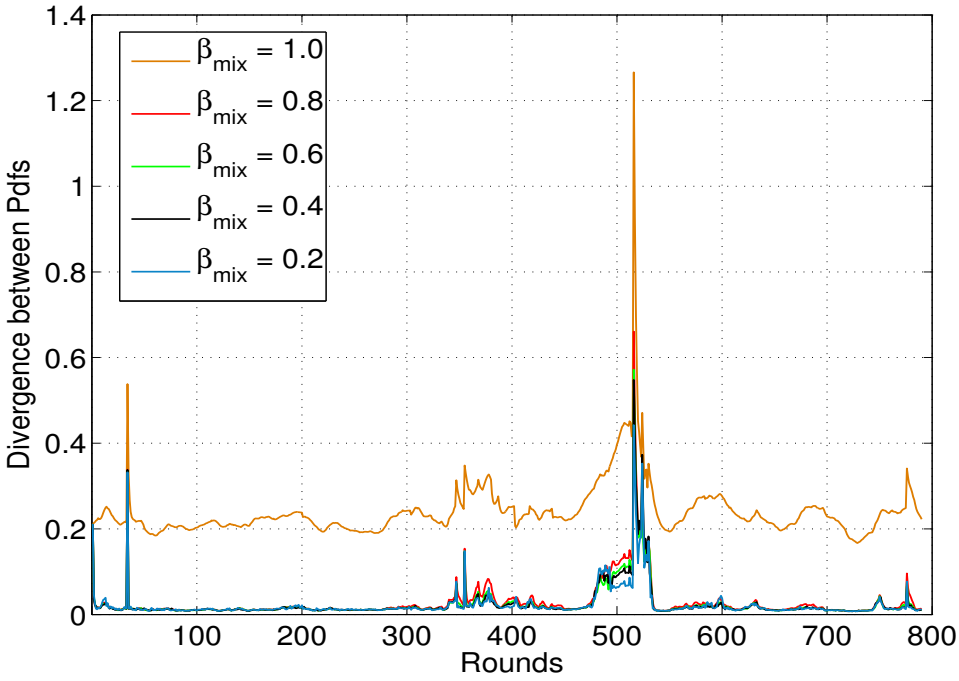
As an alternative, one could approximate the observed shortest temporal paths by a well-known distribution. We observe that, when a network of nodes is temporally connected over an observation window τ, the temporal distances can be approximated by a Γ distribution. We choose the Γ distribution mainly because it can be used to represent a family of continuous exponential probability distributions, and is traditionally used to model waiting-times until infinity. The Γ distribution takes on only two parameters, i. e., the shape k and the scale θ, that can be communicated between nodes. The parameters (k_i, θ_i) are estimated locally on a node n_i using a maximum likelihood estimator (MLE). The global view on temporal distances is then constructed by applying the consensus and time update equations independently over the k and θ values.

Throughout the chapter, our evaluation of *PathDetect* is based upon Matlab simulations. We choose three mobility models, namely the SLAW [73], RandomWalk and the Randomway-Point [11, 52]. Particularly, the SLAW [73] model describes the walking pattern observed in large crowds, and the RandomwayPoint [11, 52] model usually forms a topology that has a greater density of nodes at the center than at the edges. Every node moves at a speed that is uniformly chosen between 1.0 to 1.25 m/s. Each simulation lasts 8000 rounds. Each topology comprises 119 mobile nodes, and the average node density was set to 15. The observation window τ was set to 10 rounds.

Figure 4.2 characterizes the distribution of temporal shortest distances, for the three mobile network topologies. We show both, the distribution of shortest paths represented by the Γ approximation and the histogram bins. From the figure, we see that while the Γ-approximation exhibits a sharper mode than the histogram bins, the distributions are almost identical for most



(a) Varying α_{track}



(b) Varying β_{mix}

Figure 4.3: Network path divergence for different values for α_{track} and β_{mix}.

parts. However, the use of the Γ-approximation is motivated primarily by a need to cut down on communication overhead. Especially, it will work only in cases where the distribution of temporal shortest distances can be approximated by a family of exponential continuous probability distributions. In later sections, we show examples where the histogram bins are to be preferred over the Γ-approximation.

4.2.4 Parameter-Tuning for *PathDetect*

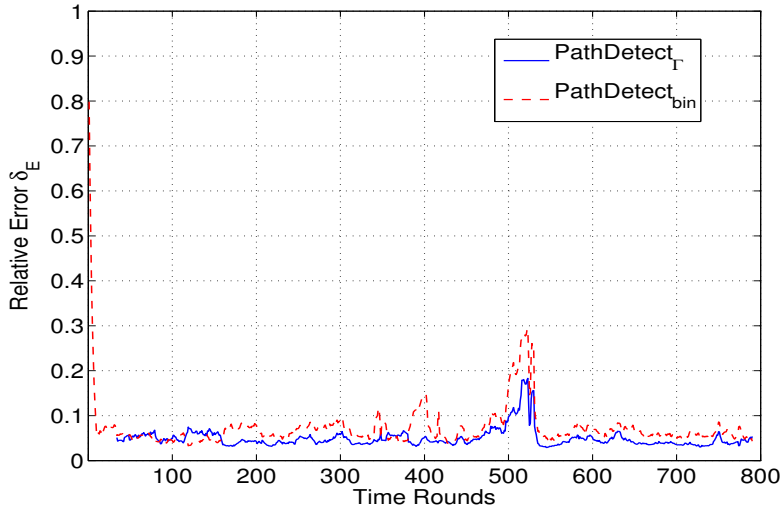
In order to characterize the similarity in the distribution observed across the network, we define a new metric, called the *network path divergence*. We denote by $\bar{D}_t$ the average network path divergence at time t. $\bar{D}_t$ is computed by considering all pairs of nodes, and computing the *Kullback-Liebler* divergence $\mathrm{Kld}(D_i, D_j)$ between their characteristic shortest path distributions. This can be expressed as: $\bar{D}_t = \frac{1}{N(N-1)} \sum_{i,j \in N} \mathrm{Kld}(D_i, D_j)$. Here, Kld denotes the *Kullback-Liebler* divergence between distributions D_i of node i, and D_j of node j, at time t. Note that the computation of divergence is not a part of the estimator, but rather an attempt to understand the degree of similarity nodes achieve in observing the temporal distances in a network.

The divergence of the estimator is influenced by the values for α_{track} and β_{mix}. Figure 4.3 shows the network path divergence for varying values of α_{track} and β_{mix}. In general, we find that the parameter α_{track} has a more pronounced effect on divergence than β_{mix}. Referring back to Equation 4.4, α_{track} weighs the partial result η^i_{t-1} obtained from consensus between neighbors, over its observation, i. e., $(1 - \alpha_{track})O_{t-1}$. Consequently, a greater value of α_{track} implies a greater confidence in the consensus value over locally observed distribution. This leads to not only a quicker convergence, but also a reduced divergence, as can be seen from Figure 4.3a. The parameter β_{mix} represents the consensus coefficient, and weighs the current estimate of a nodes distribution η^t_i over that of its neighbor. From Figure 4.3b, we see that as long as $\beta_{mix} < 1$, the divergence values are mostly similar for different values of β_{mix}. Therefore, we set β_{mix} in the range 0.2-0.8.

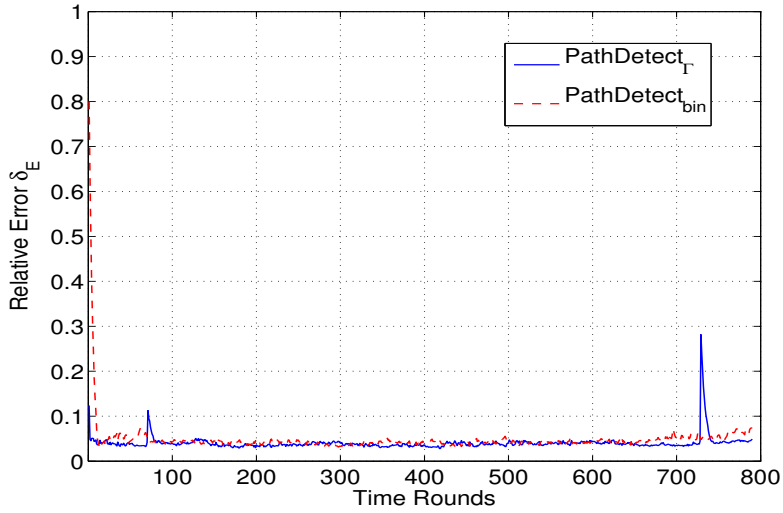
Figure 4.3 shows intermittent spikes in divergence. An inspection of the topology at these instances revealed that certain nodes were temporarily disconnected, and reconnected close to the end of the observation window τ. This leads in some sense, to a distribution that is skewed at values close to the diameter of the network. Such skewed distributions are responsible for the momentary increase in divergence. Nevertheless, the distributed consensus ensures that they get smoothened out over time. Hereafter, we set τ to a large value, i. e., 50 rounds.

4.2.5 Distributed Computation of Temporal Efficiency

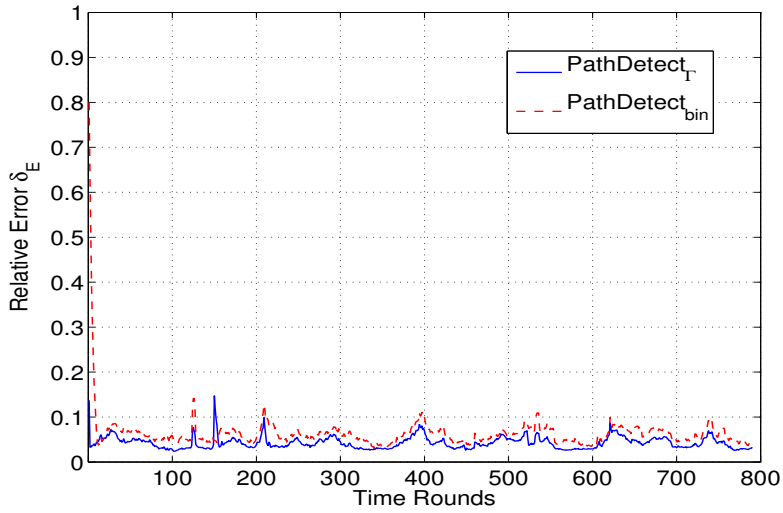
The temporal efficiency of a mobile network has been computed in [116, 127] by aggregating the temporal distances over all nodes, in a centralized manner. We are interested in computing this metric on all nodes, using a decentralized algorithm. Doing so would provide information regarding network connectivity on every node, in real-time. In what follows, we map Equation 4.1 (cf. Section 4.1) to a distributed computation process that can be carried out on the nodes. Note that $E(\tau)$ from Equation 4.1 can be expressed as an average of so-called *partial* temporal efficiencies recorded by individual nodes. For the sake of notation, let $\hat{E}_i(\tau)$ be the *partial* temporal efficiency computed by node i. The distributed average temporal efficiency, denoted



(a) SLAW



(b) RandomWalk



(c) RandomwayPoint

Figure 4.4: Relative error δ_E on different mobile network topologies.

by $\hat{E}(\tau)$ can therefore be expressed as: $\hat{E}(\tau) = \frac{1}{N}\sum_i^N \hat{E}_i(\tau)$. Here, $\hat{E}_i(\tau)$ is equivalent to and therefore replaces $\frac{1}{N-1}\sum_{i,j}\frac{1}{d_{i,j}(\tau)}$ from Equation 4.1. Since nodes already have the distribution of temporal distances, all that needs to be done is to compute the expected value of the temporal shortest path. This is equivalent to computing a weighted sum of the shortest path distribution. Next, given a set of values $\hat{E}_i(\tau)$, it is always possible to compute an average over these values using a distributed consensus algorithm. The consensus equation takes the following form:

$$\hat{E}_i(\tau) = \alpha_E \hat{E}_i(\tau) + (1 - \alpha_E)\hat{E}_j(\tau). \tag{4.5}$$

Where $\hat{E}_i(\tau)$ and $\hat{E}_j(\tau)$ are the *partial* temporal efficiencies computed at nodes i and j respectively. The parameter α_E represents the consensus co-efficient, which we set to 0.5. The distributed consensus is expected to converge very rapidly, such that eventually $\sigma_{\hat{E}(\tau)} < \epsilon$, where ϵ is a certain error threshold.

In order to characterize the accuracy of the distributed computation of temporal efficiency, we performed simulations with different mobile network topologies. In particular, we experimented with the SLAW, RandomWalk and RandomwayPoint mobility models. After each mobility round, we computed $E(\tau)$ globally, and also the relative error δ_E in temporal efficiency as shown: $\delta_E = \frac{1}{N}\sum_i^N \frac{|E(\tau)-\hat{E}_i(\tau)|}{E(\tau)}$. Figure 4.4 shows the relative error in temporal efficiency, computed for both a $\Gamma-$approximation and the method of histograms, named hereafter as $PathDetect_\Gamma$ and $PathDetect_{\text{bin}}$ respectively. In most cases, the relative error was below 10%.

4.2.6 *PathDetect*'s View On Temporal Metrics

PathDetect provides a bird's eye view of temporal connectivity across the connected component of a network. In some sense, this can be different from the same set of metrics observed over a global scale. Particularly, when the network is composed of several disconnected components operating in isolation, the globally computed temporal efficiency often provides a pessimistic representation of dissemination speed. This is mainly because the computation includes also the pairs of nodes that belong to disjoint network components. Specifically, if N' is the number of nodes in an isolated component of a network of N nodes, then the locally computed temporal efficiency can be expressed as: $\hat{E}'(\tau) = \frac{1}{N'(N'-1)}\sum_{i,j}\frac{1}{d_{i,j}(\tau)}$. In most cases, $\hat{E}(\tau) \leq \hat{E}'(\tau)$, i. e., the globally observed temporal efficiency is an underestimation of the actual temporal efficiency that characterizes a connected component. We argue from a practical standpoint that the locally estimated temporal efficiency is more relevant to the problem of determining the average dissemination delay in a network.

4.3 Evaluation

In this section, we present results of evaluating *PathDetect* under diverse topological scenarios that arise in mobile wireless networks. Specifically, we compare the estimates of $PathDetect_\Gamma$ and $PathDetect_{\text{bin}}$ against the ground truth that is computed centrally. The evaluation of *PathDetect* is divided into three subsections. Firstly, we study the robustness of temporal path connectivity that is influenced by random node and link failures. We show that our distributed approximation bears a good accuracy with respect to the globally computed temporal efficiency. Secondly, and

more importantly, we investigate the scalability of *PathDetect* concerning accuracy, with the number of nodes in the network. Specifically, we look at how *PathDetect* performs when only a subset of the nodes in the network serve as message sources, for varying values of network size and node density. Lastly, we look at how *PathDetect* adapts to gradual changes in topology.

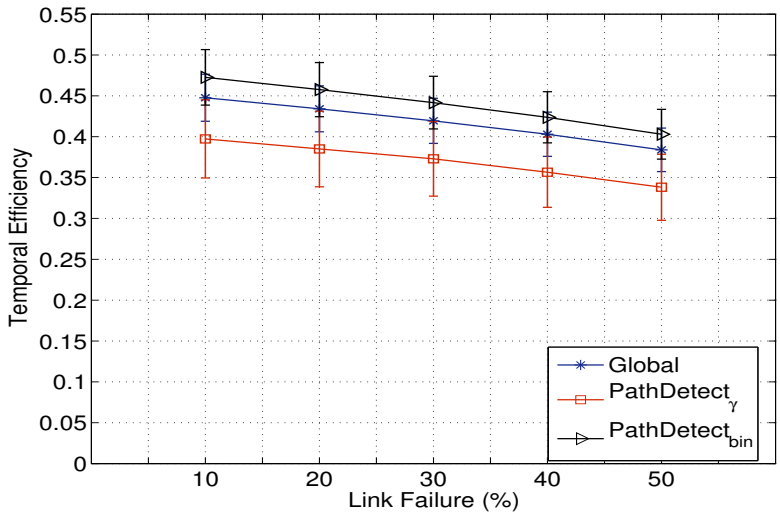
4.3.1 Robustness To Failures

We characterize information propagation speed under the influence of node and wireless link failure conditions, and also showcase the accuracy of *PathDetect* in recording the resulting degradation in temporal efficiency. In what follows, we consider link and node failures in isolation.

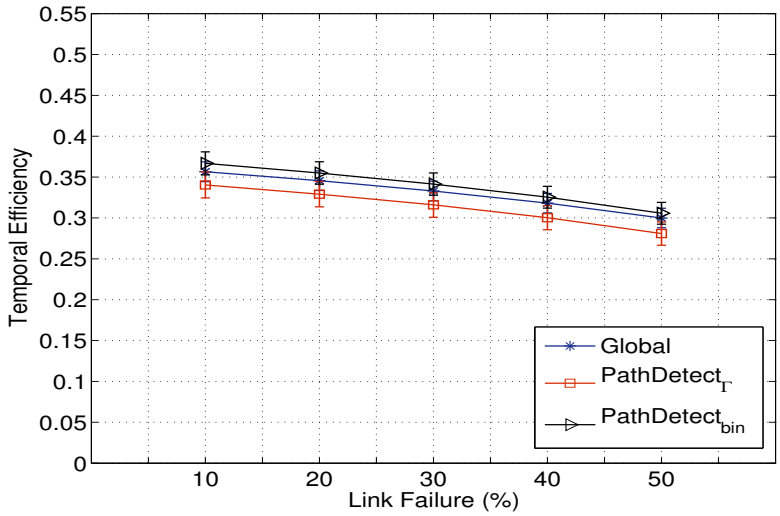
Link Failures We model link failures as a fixed probability p, i. e., nodes drop messages at random with probability p. The probability p is varied from 10% to 50%, and the resulting temporal metrics are observed. Figure 4.5 shows the averaged temporal efficiencies computed by $PathDetect_{\Gamma}$, $PathDetect_{\text{bin}}$, and compares them against the ground truth, which we denote as *Global*. In general, both the SLAW and the RandomwayPoint mobility temporal efficiency register a greater temporal efficiency than RandomWalk. This is mainly because nodes tend to form closely-connected clusters in the case of SLAW or RandomwayPoint, while they are usually distributed uniformly in the case of RandomWalk. The temporal efficiency degrades slowly (by only a 11% for *Global*) as the link failure rate is increased from 10% to 50%. Furthermore, both $PathDetect_{\Gamma}$ and $PathDetect_{\text{bin}}$ estimate the temporal efficiency with a very good accuracy with respect to the ground truth. In most observations, the relative estimation error for both $PathDetect_{\Gamma}$ and $PathDetect_{\text{bin}}$ is below 10%.

Node Failures We model node failures by disabling a subset of nodes at random in every simulation run. For the sake of comparing temporal efficiencies, we plot for $PathDetect_{\Gamma}$ and $PathDetect_{\text{bin}}$, the temporal efficiency over the entire set of network nodes. As explained in Section 4.2.6, the estimates recorded by *PathDetect* accounts only for nodes that are temporally connected. Therefore we scale down the results of $PathDetect_{\Gamma}$ and $PathDetect_{\text{bin}}$ to compare against the ground truth. Note that the scaling operation does not form a part of the algorithm itself. Figure 4.6 shows the averaged temporal efficiencies for $PathDetect_{\Gamma}$, $PathDetect_{\text{bin}}$ and compares them against the temporal efficiency computed centrally. The major difference observed in Figure 4.6 with respect to Figure 4.5 is that the degradation in temporal efficiency is more severe. Figure 4.6a shows a 73% decrease in temporal efficiency when the node failure percentage is increased from 10% to 50%. Nevertheless, we observe that both the results of $PathDetect_{\Gamma}$ and $PathDetect_{\text{bin}}$ are accurate to within 10% of the ground truth, in most cases.

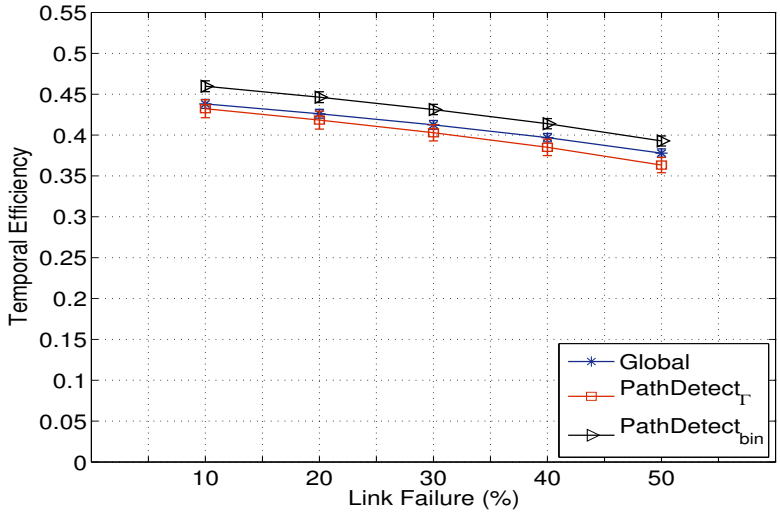
To summarize, we observe that *PathDetect* can follow in real-time, changes in temporal efficiency on account of random node and link failures.



(a) SLAW Model

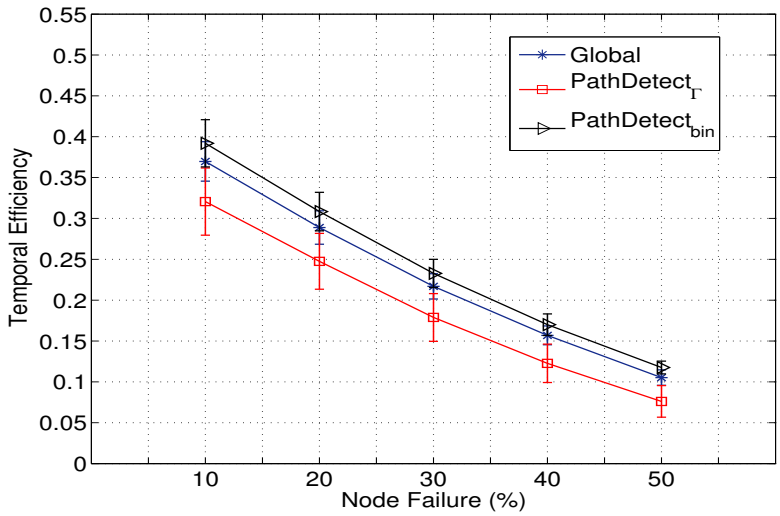


(b) RandomWalk Model

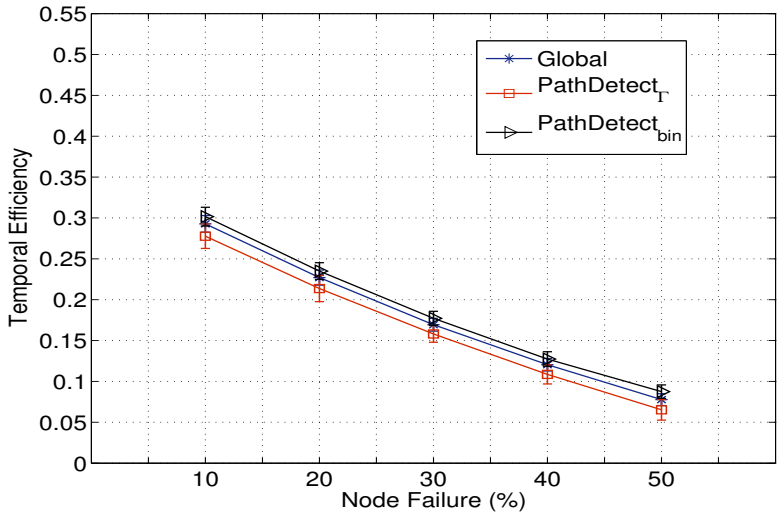


(c) RandomwayPoint Model

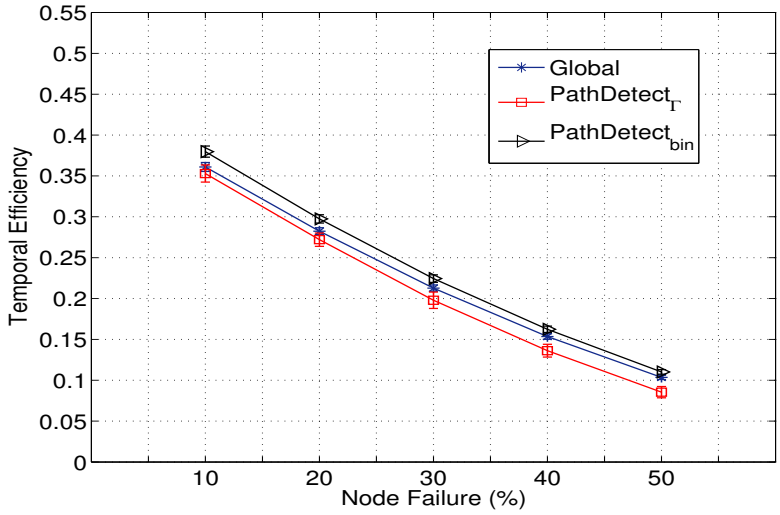
Figure 4.5: Degradation in temporal efficiency over increasing link failures. The results are averaged over 50 runs.



(a) SLAW Model

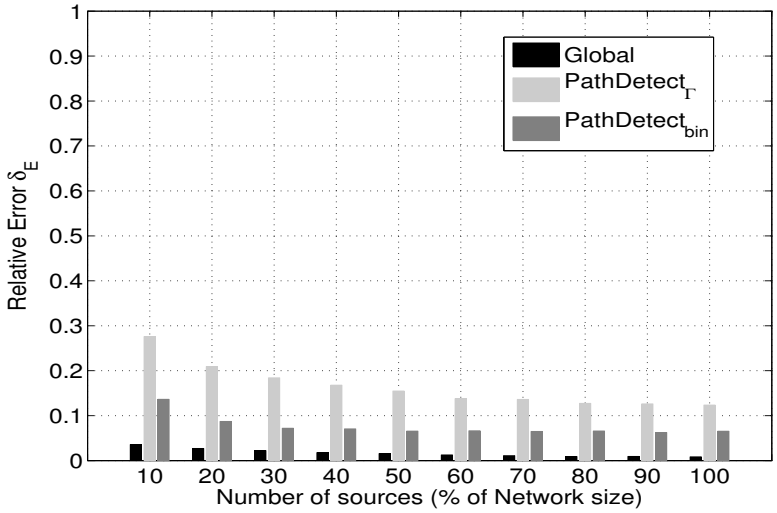


(b) RandomWalk Model

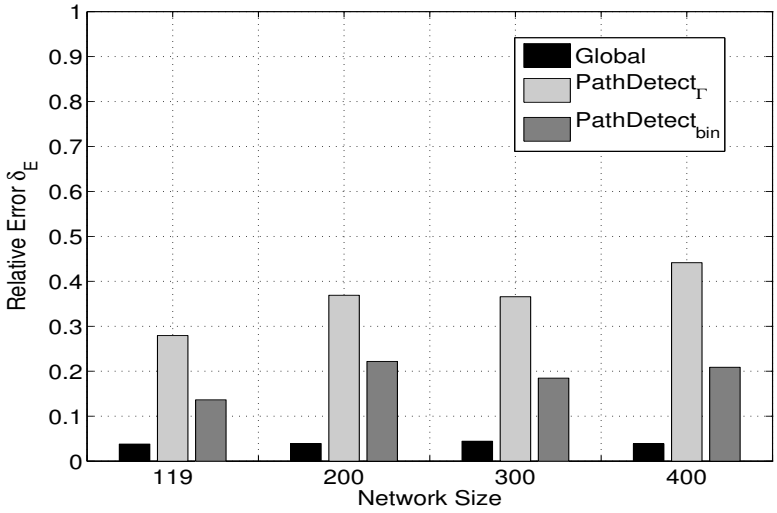


(c) RandomwayPoint Model

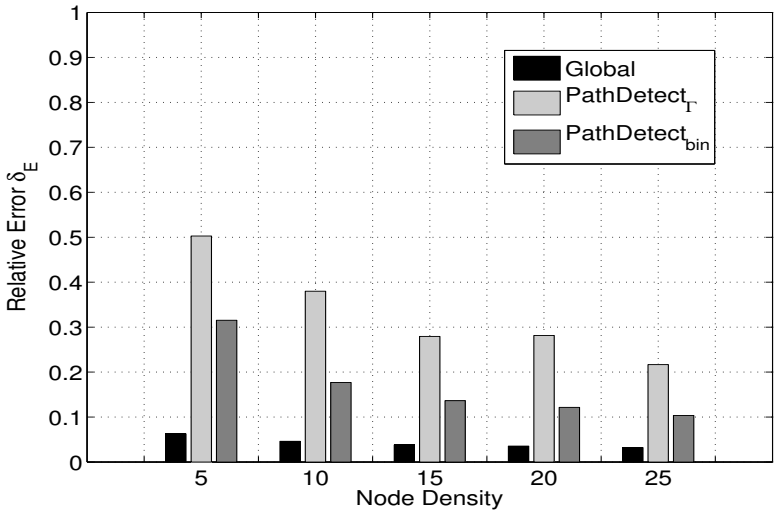
Figure 4.6: Degradation in temporal efficiency over increasing node failures. The results are averaged over 50 runs.



(a) Varying Sources, 119 nodes, density = 15



(b) Varying Network Size, 12 sources, density = 15



(c) Varying Node Density, 12 sources, 119 nodes

Figure 4.7: Overhead vs Accuracy Trade-off.

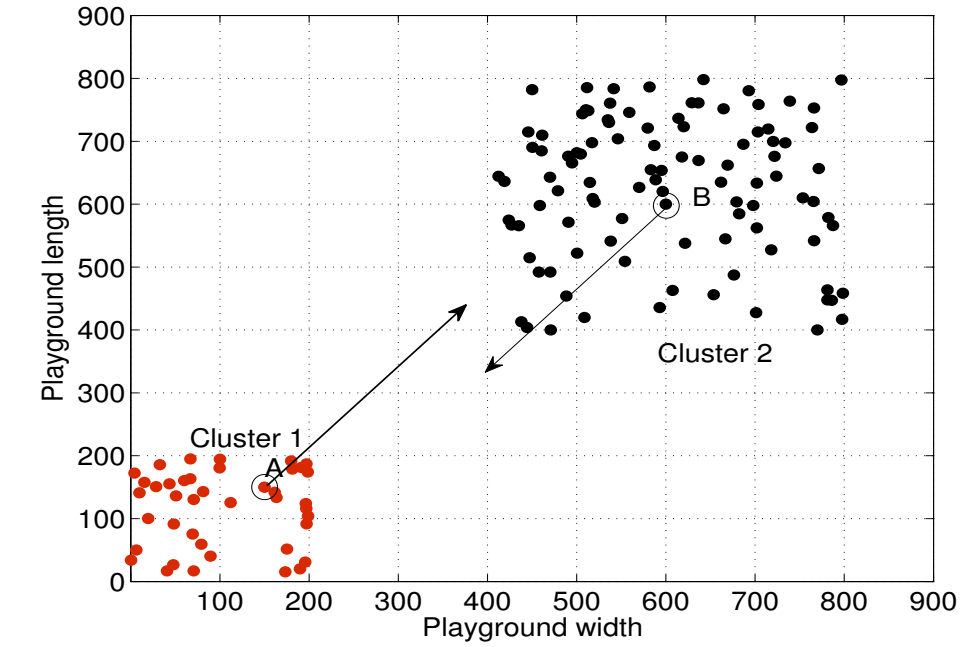
4.3.2 Overhead vs Accuracy Tradeoff

Having all nodes in the network periodically generate new messages raises the issue of scalability in very large networks, for example, > 100 nodes. We try to address this issue by limiting the number of source nodes within the network. We choose nodes at random to act as message sources. Nodes now propagate information concerning the shortest temporal paths only from the source nodes to every other node. However, the reduction in bookkeeping is accompanied by an increased approximation error of temporal efficiency with respect to the ground truth. We characterize the relative error in temporal efficiency, for both the central computation and *PathDetect*. Note that the offline computation also suffers from approximation errors, since we restrict our observation to a fixed number of source nodes.

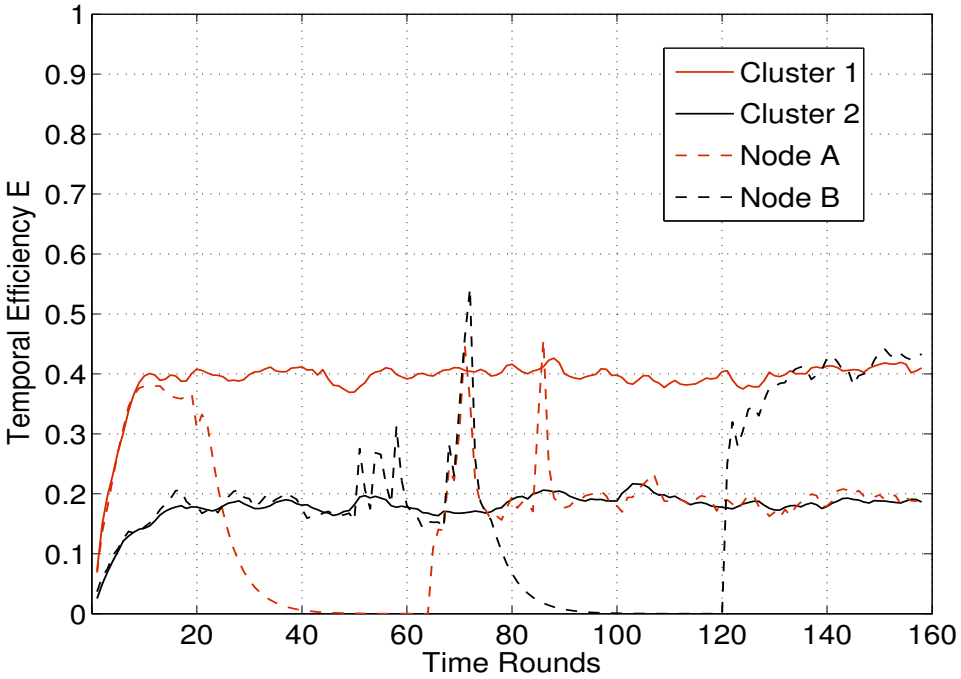
Figure 4.7 shows the relative error achieved by an offline computation (labelled as *Global*), and the two variants of *PathDetect*, for varying values of source nodes, network size and node density respectively. Figure 4.7a shows that even when the number of source nodes is fixed to 10% of the network size (i. e., 12 out of 119 nodes), the relative error for the offline computation is within 5% of the ground truth. The relative error for $PathDetect_{\text{bin}}$ is found to be within 10% of the true value, in most cases. However, the performance of $PathDetect_{\Gamma}$ degrades when the number of source nodes is decreased. The increased relative error is caused by a faulty estimation of the Γ-distribution parameters. Moreover, the distributed consensus ensures that a faulty estimate propagates rapidly through the entire network, thereby worsening the overall estimate. Such cases arise when the partially observed distribution of temporal distances do not resemble the family of distributions that the Γ-distribution can approximate. In contrast, the approach of exchanging discrete probability bins is simple, and yet effective in approximating the distribution of shortest temporal paths. More importantly, Figures 4.7b and 4.7c collectively show that the relative error of *PathDetect* increases for large and sparse networks. For example, with 10% of nodes as sources, $PathDetect_{\text{bin}}$ achieves a relative error of 32% in sparse topologies (average node degree of 5), and 20% in a network of 400 nodes. In such cases, having more source nodes helps to approximate better the temporal efficiency of the network.

4.3.3 Adaptivity to Topological Changes

While being able to compute the temporal efficiency on every node, it is also important for a node running *PathDetect* to track gradual changes in its temporal connectivity. Figure 4.8a shows a mobile network composed of two isolated clusters 1 and 2. Nodes within a cluster move according to the rules laid out by the SLAW [73] mobility model. Note that such a scenario is characteristic of groups of people moving along specific parts of a large city. The thick lines in Figure 4.8b refer to the temporal efficiency computed by the nodes in both clusters, as a function of time. Particularly, we track changes in temporal efficiencies for nodes A and B (shown by dotted lines in Figure 4.8b), as they move from one cluster to another. We observe from the graphs that the temporal efficiency computed on nodes A and B reflect the network component to which they are connected. Node B observes an almost similar temporal efficiency as the rest of the nodes in cluster 2, until time $t = 70$ when it leaves cluster 2. Moreover, it begins to track the temporal efficiency of cluster 1 from time $t = 130$. The results show that *PathDetect* works in an adaptive manner on all nodes, and tracks connectivity changes in real-time.



(a) Topology



(b) Temporal Efficiency

Figure 4.8: Temporal Efficiency recorded on two nodes A and B, as they move between cluster 1 to cluster 2 . Node mobility within a cluster is based upon the SLAW model. Zero values for efficiency reflect sojourn periods in which the nodes are completely disconnected.

4.4 Discussion

The design of *PathDetect* forms the first step towards a decentralized characterization of temporal connectivity in mobile wireless networks. In the following, we elaborate on important factors to consider from both a systems perspective and observed characteristics of real-world networks.

4.4.1 Applicability to Systems

From a systems perspective, we are particularly interested in understanding the computational issues and messaging overhead of *PathDetect*. An important parameter that affects the outcome of *PathDetect* is the observed granularity of temporal distance. In our design and evaluation, we assumed it to be of the same order as a mobility round. In general, there is a trade-off between the desired granularity of information and the computational overhead. A high level of granularity provides a more accurate representation of information propagation speed, but also incurs an increased computation. Additionally, a high granularity of observation is likely to introduce sharp discontinuities in the distribution of temporal distances. This somewhat limits the applicability of a statistical approximation approach such as in $PathDetect_{\Gamma}$. However, the $PathDetect_{\text{bin}}$ should approximate the distribution with very good accuracy, regardless of the observation granularity.

Concerning messaging overhead, the system model in Section 4.1 assumes perfect communication between nodes in one mobility round. We believe this is not a restrictive assumption. A mobility round may be composed of several communication rounds. Assuming a communication round of one second, and a single packet transmission time of 2 milliseconds, the channel allows enough bandwidth to include 500 message broadcasts between a pair of nodes. While this does not scale with a large number of nodes, it must be noted that *PathDetect* accounts for communication delays in its computation. As long as the information for *PathDetect* can be piggybacked on normal broadcast messages, *PathDetect* will not introduce any significant additional traffic. Concerning the payload for *PathDetect*, every broadcast message from a node must include the following: (1) the set of reachable nodes, along with relevant timestamps, (2) the distribution of temporal shortest paths, and (3) the estimate of temporal efficiency. Since it is not feasible to include all the information in one packet, we advocate splitting them over several messages. In the face of message losses, however, such an approach will bear an increased approximation error against the ground truth. As future work, it would be interesting to characterize the approximation error over a varying degree of message losses.

4.4.2 Handling Real-world Networks

The evaluation of *PathDetect* so far, considered topologies that are temporally connected, i. e., in most cases, every node has a temporal path to every other node. In reality, mobility traces collected from several sources [59, 148] reveal sparse topologies in which a temporal path between pairs of nodes is not always guaranteed. Furthermore, the data samples are collected over a period spanning several days. In order to operate *PathDetect* in such deployments, the observation window τ must be set to a large value (several hours) to have a meaningful interpretation of data. Additionally, since the topology is sparse, we expect sharp discontinuities in the distribution of shortest paths. In such scenarios, $PathDetect_{\text{bin}}$ is to be preferred over $PathDetect_{\Gamma}$.

4.5 Related Work

Our definition of a temporal network borrows primarily from the recent works [5, 128] that represent it as a set of adjacency matrices. Additionally, the works introduce the concepts of temporal distance and the shortest temporal path length, which we use in this chapter. Tang et al. [128] investigate the small-world properties in temporal networks, while the work by Basu et al. [5] studies the reachability properties between node pairs based on temporal distances. Our work specifically extends the concepts of shortest temporal path lengths presented in [116, 126, 127]. The work by Tang et al. [127] introduces the definition of the global temporal efficiency for a network. The works by Tang et al. [126, 127] characterize the temporal metrics globally for network topologies gathered from real-world mobility traces. In contrast, we formulate an algorithm (cf. Algorithm 2) for a decentralized characterization of the shortest path distribution and the temporal efficiency. A recent work by Scellato et al. [116] studies the degradation in observed global temporal efficiency in the presence of node and link failures. We extend this study in our work, by comparing the ground truth observation with the results of *PathDetect*.

There is a fair amount of related work that studies the delay for information dissemination in wireless networks [34, 35, 50, 62, 65, 130]. We observe that these works are either restricted to analysis, or they focus on specific scenarios of node topology. For most parts, they ignore the temporal nature of information delay that includes also the waiting times on individual nodes. Kong et al. [65] uses the concepts of percolation theory to analyze the delay in the so-called *subcritical* and *supercriticial* phases of information dissemination in a mobile wireless network. However, the authors consider only a spatial distribution of nodes in the network, and ignore the temporal aspects of delay that are equally significant. Groenevelt et al. [35] model the expected message delay in a mobile network as a function of both the network size (i. e., number of nodes) and the exponential distribution parameter of the interarrival encounter times between nodes. However, the estimation of message delay is linearly sensitive to the distribution parameter. Additionally, a distributed variant of their work requires a knowledge of the network size. Likewise, the work by Gopalan et al. [34] uses the knowledge of network diameter to analyze the spread of information across a network. Jaquet et al. [50] provides an upper bound on the information propagation speed for RandomwayPoint and in general, Brownian motion models. The analysis is restricted to a particular mobility model, and requires a knowledge of the maximum node velocity. Different from other works, Zyba et al. [152] stress the importance of *vagabond* nodes on minimizing the information dissemination delay in mobile networks. Typically, they identify a critical ratio of vagabond nodes versus social nodes, beyond which dissemination is vastly improved.

An overlap with our work can be found in [42, 87] where the authors model the network topology as a temporal graph, and look into the problem of determining the shortest temporal graph. Both works are of a theoretical nature. Specifically, the work by Huang et al. [42] formulates two greedy centralized algorithms that finds the shortest temporal paths between pairs of nodes. Likewise, Meregu et al. [87] propose a Floyd-Warshall like algorithm that computes the shortest temporal paths between all node pairs. There are however, two principal differences between these approaches and our work: *(i)* We characterize a distribution of shortest temporal

path lengths, and do not specifically focus on routing, and *(ii)* The aforementioned algorithms are centralized, while *PathDetect* works in a decentralized manner on all nodes.

4.6 Conclusion

In this chapter we addressed the problem of estimating the temporal connectivity in dynamic wireless networks. In some sense this is equivalent to understanding the delay of information dissemination in wireless networks. Conventional approaches in literature either determine the shortest temporal connectivity information offline or analyze bounds on dissemination delay. In contrast, we treated the temporal connectivity of a network as a frequency distribution, and determined the distribution at runtime. We designed and evaluated *PathDetect*, a decentralized algorithm that estimates the distribution of temporal distances between all pairs of nodes. *PathDetect* disseminates temporal distance information using a broadcast mechanism, and couples it with a distributed consensus algorithm to achieve a global view on connectivity. Evaluation of *PathDetect* under diverse conditions of node failures, link failures, and mobility patterns has shown that *PathDetect* can track changes in temporal connectivity with very good accuracy, i. e., less than 10% relative error with respect to the ground truth in most cases. These promising results warrant an exploration into the possibilities of a real-world implementation of *PathDetect*.

In this thesis, we have so far addressed the need for self-regulating mechanisms for optimizing node-to-node communication and estimating connectivity in dynamic network topologies. In the following chapter, we will take a step closer towards an implementation that runs on real hardware. Additionally, we address the concern of energy consumption, which is vital especially for low-power monitoring networks such as WSNs.

Chapter 5

A Multichannel Approach to Mitigate External Interference*

In previous chapters, we looked at decentralized algorithms for estimation and adaptability at both the neighborhood and the network level. Our approach so far involved algorithm design, preliminary evaluation based on simulations, and some controlled experimentation on hardware. Although we focused mainly on the implications of node mobility, it must be noted that even a static wireless network can be subject to changes in link quality over time, due to changes in environment such as temperature, humidity, and also channel utilization. Most applications running on static networks involve transferring messages to a data collecting point, i. e., the sink. It is therefore, necessary to ensure that the transfer of data is carried out as efficiently as possible, with minimal number of failures. Furthermore, since static networks for monitoring applications are expected to operate unattended over a long timescale (months, or even years), energy conservation is of paramount importance.

In this chapter, we focus specifically on the reliability of radio communication links in sensor networks. Since sensor node radios operate in unlicensed ISM bands, they often face the problem of having to share the communication medium with other devices in the neighborhood. In particular 802.15.4 radios, which operate in the 2.4 GHz range, interfere with prominent communication technologies like Wi-Fi (802.11) and Bluetooth. Therefore, the predominant class of data collection applications for WSNs tend to suffer in data reliability on account of frequent occurrences of external interference. Interestingly, sensor node transceivers offer communication on different non-overlapping frequency channels. This multichannel feature could be leveraged to ensure a seamless operation of sensornets in the face of severe external interference. Since energy conservation is necessary for prolonging the network lifetime, the multichannel feature must be exploited in an energy-efficient manner, without significantly interfering with the main application that runs on the node.

As a multichannel solution for mitigating external interference, we present the integration of

*A preliminary version of this work, titled 'Chamaeleon - Exploiting Multiple Channels to Mitigate Interference' was presented at *IEEE INSS'10*, Kassel. The work was authored by V.G. Iyer, M. Woehrle and K.G. Langendoen. A revised version of this work by the same authors, and titled 'Chrysso - A Multi-channel Approach to Mitigate External Interference' appeared in the proceedings of *IEEE SECON'11*, Salt Lake City, Utah.

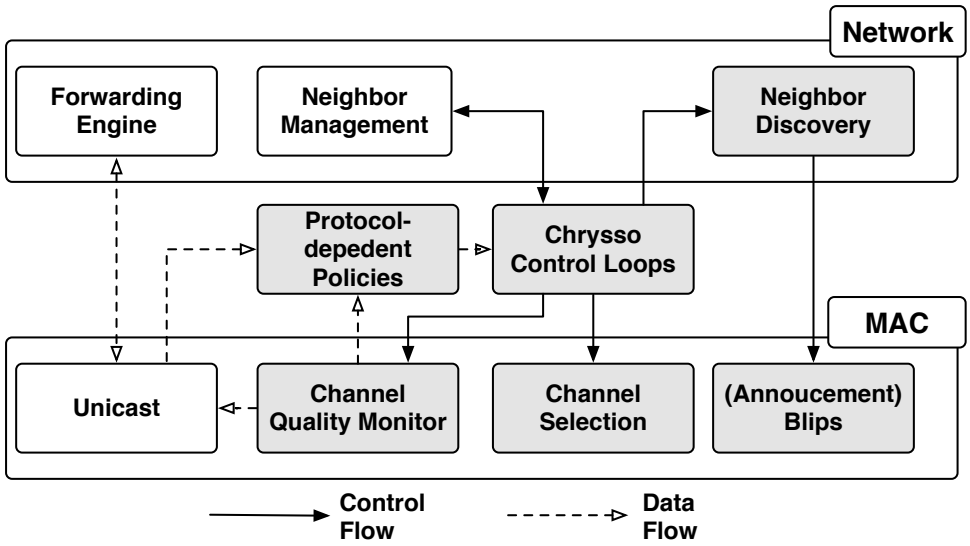


Figure 5.1: Integrated protocol stack with Chrysso-specific parts in grey

Chrysso*, a protocol extension, into a typical protocol stack of a convergecast routing protocol and a low-power MAC protocol as shown in Figure 5.1.

The design of Chrysso considers several crucial observations:

- External interference may only be local. Hence, coordination of channel assignments to nodes should be localized.
- External interference varies over time, so channel allocation cannot be performed offline.
- For large networks it is not feasible, if not impossible, to determine a single channel that is not affected by any source of external interference.

Accounting for these observations Chrysso performs energy-efficient, online monitoring of channel quality to detect external interference. Chrysso handles interference locally and hence does not require global synchronization nor scheduling. When interfered, Chrysso switches only the subset of affected nodes to a new channel effectively evading the interference source on the spot.

This chapter combines previous ideas and results [48, 49] with novel ideas and contributions. In summary, it includes the following contributions:

- The chapter presents our comprehensive work on Chrysso, a protocol extension for locally detecting and mitigating interference effects in data collection WSNs.
- It describes Chrysso's novel mechanism for dealing with neighbor discovery in a multi-channel environment.
- It details on the implementation of two generations of Chrysso, our multi-channel protocol extension, for a typical data collection stack in Contiki.

*The protocol is named after the spider Chrysso venusta, which has been observed to rapidly change its color when disturbed.

- It discusses the design of protocol-specific policies and benchmarks their performance with respect to their parametrization.

- It presents extensive evaluation of Chrysso in simulations and on three different testbeds. We show Chrysso's superior performance in normal indoor operation as well as in dedicated jamming tests, e. g. using JamLab [8].

The outline of the chapter is as follows: Section 5.1 describes the design goals for external interference mitigation, and explains the core-concepts of Chrysso. Section 5.2 covers in detail, an implementation of Chrysso on real-mote platforms. Specifically, it characterizes the sensitivity of certain controller parameters on performance, and also the resulting overhead. Section 5.3 presents an initial evaluation of a variant of Chrysso, called Nordica. The experiments feature both controlled and real-world interference scenarios, and is conducted on two real testbeds. Section 5.4 presents Venusta, an improved version over Nordica, that reduces protocol overhead and improves performance. Experiments with Venusta on a larger testbed, featuring realistic interference traces are covered in Section 5.5. Section 5.6 describes the related work on mitigation of external interference. Finally, Section 5.7 concludes the chapter.

5.1 Chrysso Design

Before presenting the design of Chrysso and its individual components, we introduce the general design goals. Additionally, we detail on Chrysso's neighborhood discovery mechanism as it is a vital component of any multi-channel protocol.

5.1.1 General Design Goals

The design of Chrysso is motivated by the following observations and ideas:

- Since external interference cannot be predicted, a continuous, yet energy-efficient monitoring of channel quality needs to be performed on each node. Hence, sensor nodes running Chrysso continuously monitor link qualities. The detection of external interference depends on the specific protocol stack. It is implemented on top of Chrysso with a set of protocol-specific policies. An example protocol policy for our Contiki-based prototype implementation, the so-called Nordica Chrysso variant, is detailed in Section 5.2. We improved Nordica with an adjusted protocol policy called Venusta, which we discuss in Section 5.4.

- External interference usually affects a group of nodes, rather than a single node. Therefore, interference detection requires a coordinated effort in a lean, energy-efficient manner. Chrysso exploits the inherent routing tree of data collection protocols to coordinate a parent with its children as shown in Figure 5.2. Parents and children collaboratively change their communication channel when detecting external interference. This collaboration of a parent with its children necessitates that the protocol stack provides a fairly stable routing tree.

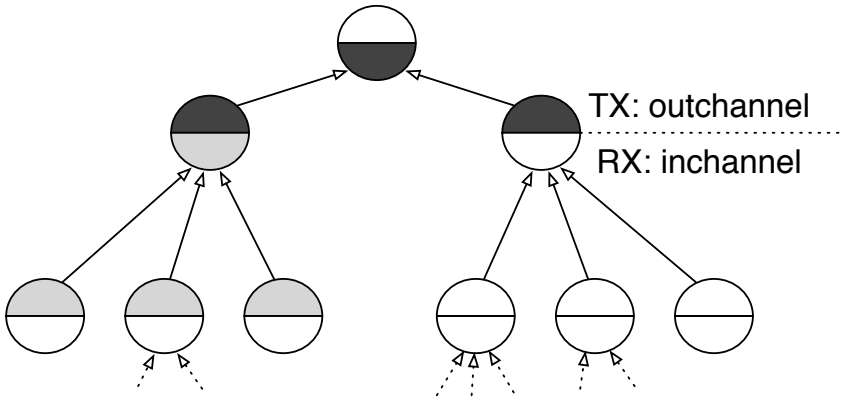


Figure 5.2: Parent-children groups each using a single frequency for communication. Each node listens on its inchannel and sends on its outchannel.

- We utilize *frequency diversity* for robustness against external interference. This frequency diversity has its price: neighboring nodes may operate on different channels, hampering neighbor detection. Chrysso includes an efficient neighborhood discovery mechanism for bootstrapping and joining of sensor nodes customized for multi-channel operation.

We specifically focus on the class of low-power listening protocols that use acknowledgements. Hence, asymmetric interference cannot be distinguished from symmetric interference as both result in retransmissions. Note that while some of its concepts also directly apply to *internal interference*, e. g. exploiting channel diversity during bootstrap, Chrysso specifically focuses on external interference. As such our work focuses on low data-rate data collection applications.

Figure 5.1 shows the design of Chrysso: It comprises several components integrated into an existing protocol stack, which will be explained below.

5.1.2 Channel Selection

The fundamental idea of Chrysso is to (logically) organize nodes of a data collection tree into parent-children groups as shown in Figure 5.2. Each parent-children group communicates on a selected channel. Individual parent-children groups may operate on different channels. Each node operates in two different roles: as a parent receiving packets from its children on its *inchannel* and as a child sending packets to its parent on its *outchannel*. Note that the inchannel and the outchannel of a node may be the same. The *channel selection* component controls the channel switching between in- and outchannel as determined by the *Chrysso control loops*. It maintains a pre-defined logical list of available channels, so that a parent and its children are consistent with their view on the next channel. It is advisable to space out (logically-adjacent) channels (frequency-wise) in a way that minimizes the possibility of consecutive channels being interfered by the same source. As an example, our implementation uses five 802.15.4 channels; 26, 14, 20, 11 and 22, in the specified order.

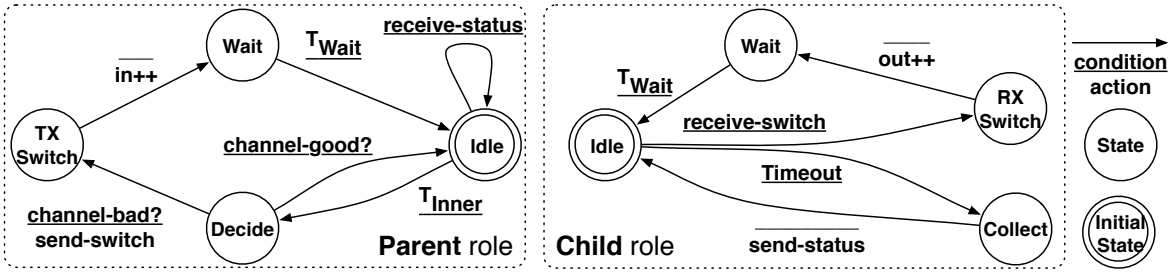


Figure 5.3: State machine for parent (left) and child role (right) of the *inner loop*. Transitions are labeled with conditions including timers and action labels (above the line, and below respectively).

5.1.3 Channel Quality Monitoring

On the lowest level, Chrysso adds a *channel quality monitor* that continuously monitors the links. For children, the monitor collects information about congestion backoffs and send successes. These are periodically piggybacked onto data packets to inform the parent as discussed below. The parent directly reads from the channel quality monitor the number of packets received. The information given by the monitor about the parent and its children is used by protocol-specific policies to determine if interference is present. The overhead of forwarding the information is negligible: for our implementation it is a mere 5 bytes.

5.1.4 Chrysso Control Loops

The core of Chrysso is a set of control loops that manages whether a parent-children pair should stay on the same channel or switch to a new channel. The *inner loop* is responsible for coordinated channel-switching between a parent and its children as soon as external interference is detected. When interference completely blocks any communication, a coordinated channel switch cannot be performed. For this case Chrysso uses the so-called *outer loop*, a watchdog mechanism that initiates an autonomous channel switch.

Inner loop Figure 5.3 shows the working of the inner loop on a single node. It differentiates among the behavior of a node in its role as a parent (on the left) and as a child (on the right). A child node periodically collects data from the channel quality monitor and piggybacks that onto data packets (`send-status`).

In periodic intervals of T_{inner}, the parent uses protocol-stack specific policies to determine whether or not the current measured channel quality indicates external interference (channel-bad? or channel-good?). In our implementation, the parent node computes an average over the backoff values and checks whether it exceeds a predetermined threshold. If this is the case, the parent initiates a coordinated switch by setting a channel switching indicator on subsequent ACK messages (send-switch). Having notified all its children, the parent node switches to the next (logical) inchannel (in++).

Outer loop The outer loop functions as a watchdog for cases of severe interference that disrupt the operation of the inner loop. Hence, the outer loop executes over a larger time interval than the inner loop. Here a node (in child and parent role) decides independently to switch channels

based on the protocol-specific policies. These decisions involve no explicit coordination between nodes; thus, channel switches triggered by the outer loop are instantaneous. However, in general both the parent and its children detect such interference and all perform the autonomous switch. In turn, the parent-children group meets on the consecutive channel.

For detecting severe interference, a child node monitors the number of failed transmissions (messages that were never sent) and switches its outchannel if the failure ratio exceeds a threshold. Likewise, the parent switches to the next inchannel when the ratio of received packets drops below a pre-set value. As these ratios depend on the specific protocol stack and the application, the corresponding thresholds are defined by the protocol-specific policies.

Wait time Chrysso incorporates a *Wait* state for both parent and child nodes, immediately after a re-connection on the new channel. Since transmissions failed before the switch, nodes are expected to have queued several packets causing a momentary increase in message transmissions after switching to the next channel. This leads to increased contention and could be mistaken for further external interference. Therefore Chrysso suspends channel switching decisions during such transient periods by implementing a timeout T_{wait}. Note that the wait time does not affect the reconnection latency; it only delays decisions about switching to yet another channel until statistics have stabilized again.

5.1.5 Neighborhood Discovery

Neighborhood discovery, i. e., finding a (new) parent, is vital in practical deployments. As an example, a routing switch may be necessary when the link quality to the current parent deteriorates and cannot be remedied by a channel switch. Because neighboring nodes operate on different channels, topology information available at a child node is only partial, and also subject to change, as neighbors may switch channels during network operation. Hence, Chrysso employs a special neighborhood discovery phase, the so-called *scan mode*, to find a new parent. Since discovering neighboring nodes in a multi-channel environment incurs additional overhead on processing and energy consumption, the scan mode is only triggered on demand.

The scan mode relies on beacons containing routing information. Routing beacons are sent on the *inchannel*. A node performing neighborhood discovery scans through the list of available channels to search for a new parent. A channel scan typically involves the node listening for beacons from all potential parents in the neighborhood. Having collected a set of parent candidates, the node chooses the best one and re-starts channel monitoring. In Section 5.2.2, we detail on our efficient implementation of the scan mode using a special kind of routing beacons, so-called announcement blips.

5.2 Implementation

To study the feasibility of our approach, we implemented Chrysso on the Telosb mote platform [17]. We chose the Contiki [21] operating system as the software platform. In the following we detail on generic implementation aspects, including the tuning of the timing parameters of Chrysso, and describe the protocol-specific policies for Contiki. Note that we refer to the specific protocol implementation of Chrysso as Nordica.

Table 5.1: Parameters of application and protocol stack

Stack	*Parameter*	*Notation*	*Value*
MAC	Wakeup Interval	T_w	250ms
	Retransmissions	N_{retx}^{MAC}	2
Collect (CT)	Retransmissions	N_{retx}^{CT}	4
Application	Sampling interval	T_{data}	32s

5.2.1 Protocol Stack

Our implementation on Contiki version 2.4 uses by default, a low-power listening (LPL) MAC. In some sense, it is similar to XMAC [12] that uses strobed preambles and duty-cycles the radio with a wakeup interval T_w. However, there are two major differences to X-MAC in the implementation on Contiki: (1) The given protocol does not use long broadcast messages with length T_w, but uses short strobe packets, so-called announcement blips, that are sent periodically with a random offset so that neighbors receive it with a given probability. (2) The MAC protocol also includes an optimization that uses time synchronization between a sender node and its receivers. In the following we use this optimization that closely resembles a synchronized LPL, in particular WiseMAC [27], wherein a sender node starts transmission just before the intended receiver wakes up.

We use the *Collect* routing protocol. The routing tree is formed and maintained by announcement blips [22] that are broadcast by every node. Each blip contains information about the node's address and its routing gradient to the sink. Based on announcements, each node builds a neighbor table that is used by the *Forwarding Engine* (cf. Figure 5.1) to select a parent. By default, the *Collect* protocol uses a naive policy for forwarding packets; for every packet, a node selects its neighbor with the best routing metric to the sink. Chrysso works best on a stable routing tree. To this end, we improve routing stability by leveraging a thresholding policy based on expected transmission count (ETX) for performing a routing switch as suggested in CTP [33]. Nodes switch their routing parent only if the newly discovered ETX is below the currently perceived best ETX by a factor of 1.5.

For testing Chrysso, we use a sense-and-send application that generates data every T_{data} on every node except the sink node. The relevant parameters of the implementation are shown in Table 5.1.

5.2.2 Chrysso Implementation

In the following we detail on the implementation of the Chrysso control loops and their coupling with the channel quality monitoring and neighbor discovery. Our implementation relies on a set of timing values that are related to protocol-specific parameters. Typically, the per-packet generation interval T_{data} and the wakeup interval T_w are important parameters that determine respectively the network traffic and available bandwidth. Our focus is on low data-rate applications, in which $T_{data} \gg T_w$. While in general Chrysso's timing parameters are dependent on T_w, in the considered low data rate applications the timing values in Chrysso are largely derived from T_{data}. Hence, they can be determined at compile time. Most importantly,

the temporal notion of the control loop evaluation is captured in the two main parameters T_{inner} and T_{outer}.

T_{inner} is the inner loop interval over which the parent evaluates the channel quality (cf. Figure 5.3). We prefer congestion backoffs as a measure of interference, chiefly because they abstract away from the underlying details of signal strength measurements as conducted in [95], and are freely available on most MAC implementations. For convergecast networks, nodes transmit packets over an interval that is equal to (for leaf nodes) or less than (for forwarding nodes) the data sampling interval. Therefore, the backoffs observed by several child nodes over a window of the data sampling interval T_{data} average to a value that is typically consistent with individual measurements. Hence, we set $T_{inner} := T_{data}$.

T_{outer} is the interval after which the outer loop decides on an autonomous channel switch. The outer loop operates over a much larger interval. This is because autonomous channel switches should only be performed when the inner loop fails to trigger a channel switch for an extended duration. We obtain good results with T_{outer} set to six times T_{inner}. With increasing values of T_{data}, the channel monitoring interval gets proportionally spaced out in time, which will increase the *absolute* latency in channel switching and reconnection. However, it must be noted that the traffic flowing to the sink reduces by a fixed proportion, such that the resulting performance would not be affected either by scaling up (or down) T_{data}.

Inner loop A parent node collects backoffs of its children measured on an average-per-packet basis. It computes the harmonic mean* over the child nodes' backoffs during an interval T_{inner}. If the computed backoff exceeds a threshold $r_{back,max}$, the parent triggers a co-ordinated channel switch by indicating so on subsequent ACK messages. The parent stays on the current inchannel for T_{data} before moving on to the next inchannel in order to notify all of its children and allow the children to follow the switching decision. Correspondingly, after receiving a switch indication, a child node waits for T_{data} before switching its outchannel.

Outer loop The outer loop is evaluated at the end of every interval T_{outer}, by both child and parent nodes. A child node monitors the number of successful and failed transmission attempts over T_{outer}. A transmission is only considered successful when a node manages to send the data packet after having received a strobe acknowledgement from its receiver. If the number of failed transmissions exceeds a certain fraction $r_{tx,max}$ of the total transmission attempts, then the child node switches its outchannel instantly and autonomously. Likewise, a parent node keeps a record of the number of packets received over T_{outer}. If the fraction of received over expected packets falls below a threshold $r_{rx,min}$, the parent switches its inchannel instantly and autonomously.

Switching mechanism On channel switches, a parent node resets its collected channel statistics. For a child node, both channel switches and routing switches necessitate a reset of its collected channel quality statistics. Additionally, the child node also clears its routing entries.

*The harmonic mean is typically used to express average of *rates*, and has been known to mitigate the impact of large outliers.

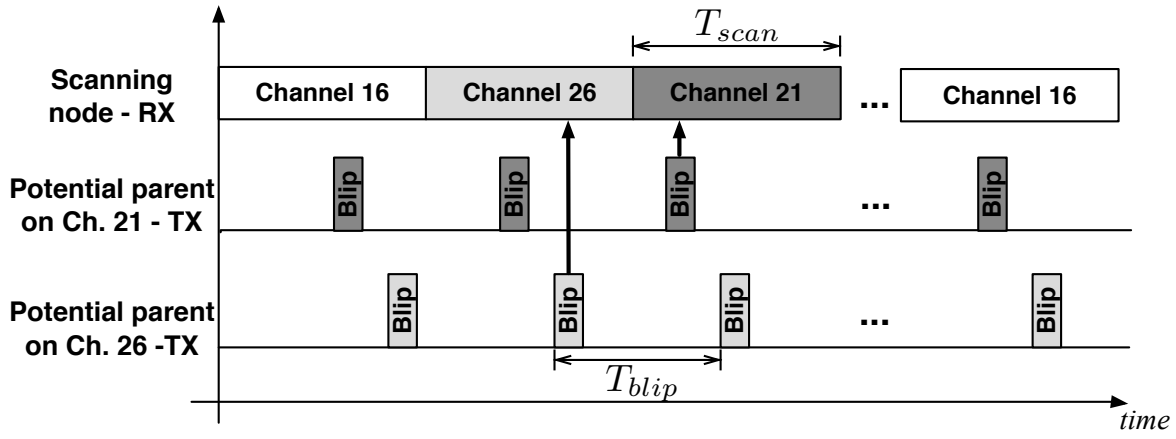


Figure 5.4: Channel scanning procedure: A scanning node sweeps through its logical list of channels (here $16, 26, 21, \ldots$), listening for blip transmissions of neighboring nodes

Immediately following a channel switch, a child node starts a watchdog timer. The watchdog initiates the scan mode (cf. Section 5.2.2) if the node does not find a parent available on the new channel. The watchdog is set to T_{outer}, thereby enabling the child node to detect a re-connection problem.

Once a connection is established, a wait interval T_{wait} is used before new channel quality evaluations are performed, as nodes experience transient effects of channel switching. In our implementation, we set T_{wait} to T_{outer}, i. e., the nodes skip one evaluation of both the inner and outer control loops.

Scan mode The *scan mode* is invoked at network bootstrap or whenever a sensor node loses connectivity to its parent and has no (other) entry in its neighbor table, e. g. following a channel switch by the outer loop. It replaces the neighbor discovery and management of the original stack.

During scan mode, a sensor node sweeps across the list of possible channels. The previously used outchannel is blacklisted for that period. A node listens for announcement blips for a duration of T_{scan} per channel, as shown in Figure 5.4. Having received one blip, the node continues to sweep across the channels one additional time, before deciding on a routing parent. This allows the scanning node to (potentially) take a better routing decision. We set T_{scan} to the same order of magnitude as T_{blip}, so that a scanning node is assured of a blip on a channel, if there is a node on it that can receive packets.

Bootstrap A node in scan mode does not send announcement blips. This is used for *network bootstrap*: All nodes are in scan mode and merely listen for blips. Only the sink sends blips to announce a routing gradient of 0. Sensor nodes recursively connect to the sink using the regular scan mode procedure.

In order to determine the bootstrap latency using our novel multi-channel scan mode, we performed experiments with and without Chrysso, i. e., using only a single channel. These tests were carried out on a four hop network comprising 26 Tmote sky nodes, including the

sink (cf. Testbed-Zurich in Section 5.3.1). Without Chrysso the median latency of the time to bootstrap for the sensor nodes is 130s and the 99%-percentile is 717s. With Chrysso the median latency is 164s and the 99%-percentile is 547s. The comparatively larger median latency of Chrysso is attributed to the *additional* scan time that a node spends before connecting to the network. Nevertheless, Chrysso produces fewer outliers on bootstrapping latency, thereby showing the effectiveness of our scan mode for multi-channel neighbor discovery.

Blips We prefer blips for neighbor discovery over long preamble transmissions, the original broadcast mechanism of X-MAC and other LPL protocols. This is because long preamble transmissions cause considerable channel contention due to their excessive strobing. In practice this results not only in an increased connection latency, but also in a sub-optimal routing tree. Moreover, blips allow for fast neighbor discovery in a multi-channel environment. Finally, we note that receive and send energy of typical 802.15.4 radios is comparable, e. g. for the the Tmote and the Iris sensor nodes. As a result, we advocate to replace LPL broadcasts with blips for multi-channel protocols on 802.15.4 radios.

Blips are only sent on the inchannel of a node. Since the inchannels and outchannels of a parent-child node pair are likely to be different, a fundamental problem of multi-channel protocols arises: *channel deafness*, i. e., not hearing a packet on a different channel. Therefore, in addition to announcing the routing metrics using blips, Chrysso also piggybacks routing metrics onto software acknowledgement messages. Note that the sequence of data message, followed by an ACK requires a parent-child pair to operate on the same channel. This ensures that routing updates are regularly received by a child node. In our implementation, the piggybacking feature involves minimal overhead on message payload, i. e., 2 bytes per ACK message.

5.2.3 Protocol-specific Policies

To identify external interference, protocol-specific policies need to be determined, i. e., few threshold parameters need to be adapted. While the threshold values vary from one protocol stack implementation to another, the process of determining these values remains the same. In the following, we describe the reasoning for determining the thresholds using our specific implementation, and support it with data from controlled tests on sensor nodes. In particular, to verify our reasoning, we performed a series of controlled jamming experiments with Tmote sky nodes. We use a setup of 1 parent and 16 child nodes. The jamming impact is varied from mild to extreme interference (as described in Section 5.3.2). We have different performance metrics for evaluating the policies:

- *False negatives* (FN), i. e., nodes failing to switch although they are interfered, should be low.
- *False positives* (FP), i. e., nodes switching channels although they are not interfered, should be low.
- The *latency* of a switching decision. The longer the latency, the larger the impact of the interference on the data collection process.

Table 5.2: Outer Loop Exploration for $r_{tx,max}$. FN and FP in percent. Latency (Lat.) in seconds. Nodes were subjected to extreme jamming to determine Latency and FN. FPs were determined without any jammer present

$r_{tx,max}$	0.6			0.7			0.75			0.8		
Metric	FN	FP	Lat.	FN	FP	Lat.	FN	FP	Lat.	FN	FP	Lat.
Result	0	2	$163s$	0	1	$187s$	0	0	$187s$	0	0	$379s$

False positives are measured using a non-interfered set of nodes using channel 26, a relatively clean 802.15.4 channel.

Inner loop policy The inner loop must detect channel interference within a short time, and co-ordinate a channel switch between a parent and its children. The inner loop should neither enforce unnecessary channel switches nor fail to react to external interference. The sensitivity of the inner loop is controlled by a backoff threshold $r_{back,max}$ that represents the maximum average backoff value per packet. $r_{back,max}$ mostly depends upon the MAC protocol implementation. With X-MAC, nodes back off, i. e., abort a transmission attempt, as soon as they receive a strobe or detect any interfering signal in their neighborhood. Therefore, a backoff threshold of 1 suffices to indicate whether the current channel is interfered. Our experiments show that in all cases neither false positives nor false negatives occurred. This demonstrates that Chrysso's inner-loop policy is robust, and effectively detects external interference.

Outer loop policy The outer loop serves as a fallback mechanism for nodes to switch channels in scenarios where the inner loop fails to trigger. Such cases typically occur when a child node misses the channel switching indication from the parent. They may also occur, although infrequently, when only a local set of child nodes experience interference, such that the parent notices nothing wrong with the channel. As with the inner loop, the thresholds for the outer loop need to be determined to minimize the occurrences of false positives and false negatives. Since the outer loop decisions are taken autonomously by both parent and children, the channel switch conditions are implemented independently for both roles.

A child node switches its outchannel if the following condition holds:

$$\frac{N_{TX\_failed}}{N_{TX\_failed} + N_{TX\_succeeded}} > r_{tx,max}, \tag{5.1}$$

where N_{TX_failed} and $N_{TX\_succeeded}$ are the number of failed and successful transmissions, respectively. As the outer loop is mainly targeted for handling severe cases of interference, we conducted short experiments with *Extreme Jamming* to evaluate different values for $r_{tx,max}$. The results shown in Table 5.2 suggest that $r_{tx,max} = 0.75$ results in a short channel switch latency with neither false positives nor false negatives. Hence, we adopt $r_{tx,max} = 0.75$ in the following.

A parent node keeps track of the number of messages received N_{RX} and switches its inchannel

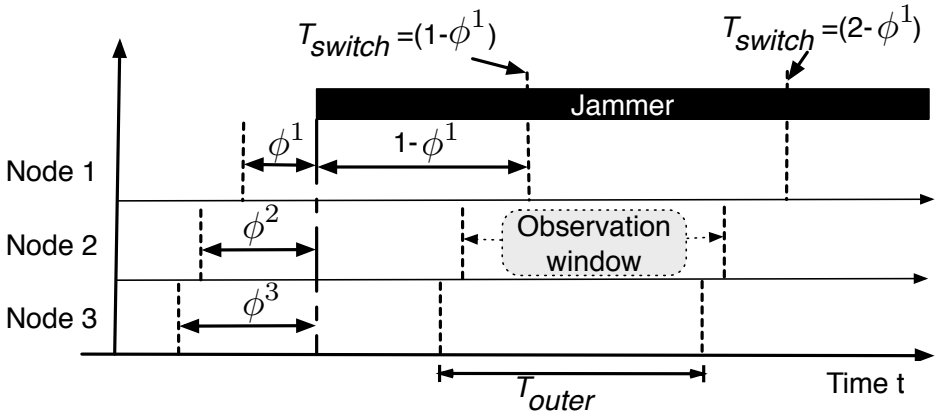


Figure 5.5: We denote the offset of the start of the T_{outer} observation window to the start of the complete jamming as ϕ^i. The outer loop is executed periodically every T_{outer}.

if the following condition holds:

$$N_{RX} < r_{rx,min} \cdot \frac{T_{outer}}{T_{data}}. \tag{5.2}$$

The ratio $\frac{T_{outer}}{T_{data}}$ gives the number of packets generated by a single leaf node. In the case of extreme interference on the parent's side the number of incoming packets would immediately drop close to zero. When only the children are affected, the incoming rate also goes to zero as an effect of them switching autonomously to another channel. In both cases, we set $r_{rx,min}$ to 0.5, so that the parent assumes the current channel noisy when it receives less than half of the packets that a single child would minimally generate. Such a policy is simple, yet effective in practice for detecting the impact of external interference on packet reception.

5.2.4 Channel Switching Latency

A major consideration for Chrysso is its responsiveness when being jammed. In particular, we must consider the channel switch latency, i. e., the time from being interfered to actually switching the channel in response. In the following we characterize the worst-case switching latency. Note that this worst-case occurs in case of complete jamming, i. e., when interference blocks any communication. This is because the resulting outer loop switch takes considerably longer than coordinated switching using the inner loop. Since the operation of the outer loop is based on observation windows of time T_{outer} as shown in Fig. 5.5, we normalize T_{switch}, the absolute worst-case channel switching time over T_{outer}, i. e., $\frac{T_{switch}}{T_{outer}}$. As the figure shows, the switching time is mainly dependent on the phase offset ϕ^i between the start of the observation window of a node i and the start of the jamming. In the following we describe how the worst-case latency is dependent on the phase offset for a child and a parent node. Note that for child nodes the latency concerns outchannel switches, and inchannel switches for parent nodes respectively.

Child Nodes Since outer loop channel switches are independently executed, we can analyze the switching latency for child nodes T^c_{switch} independently. Let ϕ^c be the phase offset of a child node c. This means that the child node can communicate successfully for $\phi^c * T_{outer}$ time

Figure 5.6: T^c_{switch} for different ranges of jammer phase ϕ^c.

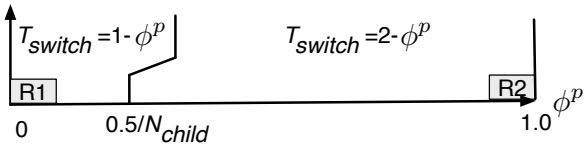


Figure 5.7: T^p_{switch} for a parent node, for different ranges of jammer phase ϕ^p.

units, and subsequently witnesses a complete breakdown in communication for the remaining $(1 - \phi^c) * T_{outer}$. Therefore, if $N_{TX\_succeeded}$ and N_{TX_failed} are respectively the number of successful and failed transmissions, then $N_{TX\_succeeded}/T_{outer} = \phi^c$, and $N_{TX\_failed}/T_{outer} = N_{retx} * (1 - \phi^c)$. Here N_{retx} represents the maximum number of retransmission attempts per packet, and is set to 8 in our implementation. Equation (5.1) and our setting $r_{tx,max}$ of 0.75 results in a switch decision when $\phi^c < 0.73$, i. e., when the phase is smaller than 0.73 a child node will switch. Figure 5.6 shows the channel switching latency T^c_{switch} over different values of ϕ^c. The two regions R_1 and R_2 denote the different ranges for ϕ^c, where in R_1 a child switches already at the end of the current outer loop window while in R_2 it does not. Hence, the switching delay for R_1 is therefore $1 - \phi^c$ and $2 - \phi^c$ for R_2 respectively.

Since ϕ^c is a uniform random variable in the range [0,1], T^c_{switch} is also a uniform random variable, whose expected value can be derived as follows.

$$E[T^c_{switch}] = \begin{cases} \frac{1}{2} \cdot ((1-0) + (1-0.73)) & \textit{if } 0 \leq \phi^c < 0.73 \\ \frac{1}{2} \cdot ((2-0.73) + (2-1)) & \textit{if } 0.73 \leq \phi^c \leq 1 \end{cases}$$

or similarly:

$$E[T^c_{switch}] = \begin{cases} 0.635 & \textit{if } 0 \leq \phi^c < 0.73 \\ 1.135 & \textit{if } 0.73 \leq \phi^c \leq 1 \end{cases}$$

The combined expected value for T^c_{switch}, i. e., how long it takes a node on average to respond to a worst-case external interference, is the weighted average of both cases: $E[T^c_{switch}] = 0.77$.

Parent Nodes Different to child nodes, the switching latency for a parent node T^p_{switch} is not only influenced by the phase offset ϕ^p between the outer loop and the interferer, but also by the number of child nodes N_{child}. A parent node switches its inchannel *via* the outer loop whenever the inequality in Equation (5.2) holds valid. As previously described, the inequality threshold $r_{rx,min} = 0.5$. Let us determine the worst-case analysis of switching times as before. If every one of the N_{child} child nodes sends packets at an interval of T_{data}, then the parent receives in an outer loop window on average $\frac{T_{outer} * N_{child}}{T_{data}}$ packets. Considering complete external interference, a parent node receives only a fraction of the packets proportional to ϕ^p, i. e., a parent node receives $(N_{child} \cdot \phi^p \cdot T_{outer})/T_{data}$ packets over an outer loop interval. Substituting this value for N_{RX} in Equation (5.2) yields $\phi^p \cdot N_{child} < 0.5$. If this inequality holds, the parent node switches its inchannel at the end of the current outer loop, resulting in a switching delay of $1 - \phi^p$ units. Correspondingly, If $\phi^p \cdot N_{child} \geq 0.5$, then $T^p_{switch} = 2 - \phi^p$.

Figure 5.7 depicts the switching latency T^p_{switch} dependent on ϕ^p. As we can see, there are two ranges R_1 and R_2. With range R_1, i. e., $\phi^p \in (0, 0.5/N_{child})$, the inequality in Equation (5.2)

holds, and the parent node switches its inchannel in the oncoming firing of the outer loop. This results in a switching delay of $1 - \phi^p$. For range R_2 (i. e., $\phi^p \in (0.5/N_{child}, 1)$), the parent node has collected enough packets to remain on the current inchannel. In such cases, the inchannel switch is performed at the next firing of the outer loop, resulting in a switching delay of $2 - \phi^p$. We can derive the expected values of the switching latency as shown below:

$$E[T^p_{switch}] = \begin{cases} \frac{1}{2}((1-0) + (1 - \frac{0.5}{N_{child}})) & if\ 0 \leq \phi^c < \frac{0.5}{N_{child}} \\ \frac{1}{2}((2 - \frac{0.5}{N_{child}}) + (2-1)) & if\ \frac{0.5}{N_{child}} \leq \phi^c \leq 1 \end{cases}$$

The combined expected value for T^p_{switch} can be expressed as a weighted average, which yields:

$$E[T^p_{switch}] = 1.5 - \frac{0.5}{N_{child}}. \tag{5.3}$$

Equation 5.3 shows that while the switching delay for the parent is upper-bounded by $1.5 \times T_{outer}$, it is sensitive to the value of N_{child}. Moreover, the upper bound is greater than the outer loop interval, i. e., T_{outer}. We revisit this issue later on in Section 5.4, where we improve on the protocol policy to ensure a faster channel switching.

5.2.5 Memory Footprint

Sensor nodes are typically resource constrained, for example, Tmote sky motes are equipped with a program flash memory of 48 KB and a data memory of 10 KB. Therefore applications need to be optimized for memory footprint. Table 5.3 shows the memory footprint of Chrysso, broken down to component level (control loops, MAC, and routing), as well as the size of their counterparts in Contiki's original, single-channel protocol stack. In total, Chrysso consumes around 4.7 KB of additional program memory and 192 extra bytes of RAM.

Table 5.3: Memory consumption (bytes) for Chrysso and the original Single-Channel (SC) protocol stack.

	Chrysso			SC	
	Ctrl loops	MAC	Routing	MAC	Routing
Program	3342	2956	2462	2254	1810
Data	148	122	398	118	358

5.3 Chrysso-Nordica evaluation

In the following we show results from several experiments across two testbeds and compare the Nordica variant of Chrysso against *(i)* the protocol stack with the channel control loops and scan mode disabled; that is, a single-channel solution denoted by *SC*, and *(ii)* a channel-hopping protocol.

5.3.1 Experimental Setup

For our test we use Tmote sky nodes that feature the CC2420 transceiver, a 802.15.4 radio. The tests were conducted on two real-world testbeds: Testbed-Delft at TU Delft comprises 17 nodes and is deployed over 14 rooms on an office floor. We set the transmission power levels such that all nodes are within communication range. Testbed-Zurich at ETH Zurich [25] features 25 sensor nodes that are also deployed on an office floor and usually create a four-hop network. Every node (except the sink) generates packets every T_{data}. For controlled jamming, we use the technique described in [7]: The jammer transmits an unmodulated signal on a channel, thereby interfering any concurrent communication within range.

We evaluate Nordica using two performance metrics: data yield and energy consumption. Since we are interested in ensuring robustness to interference, we mainly focus on the data yield at the sink: the percentage of uniquely generated packets that are received at the sink. As data collection WSNs typically rely on battery-operated sensor nodes, energy use is a prime concern. In our evaluation we focus on the radio duty cycle as the radio is the major consumer of energy on typical sensor node platforms. We use Contiki's energest module [23] and present the radio duty cycle computed at each sensor node. We present the results in four subsections. First, we test Nordica using a controlled jammer to verify the functionality of the control loops. Second, we induce Wi-Fi traffic in the proximity of the network, and observe how Nordica reacts to a sudden increase in channel activity. Both tests are performed on Testbed-Delft. These first tests show how Nordica is able to cope with interference compared to a single-channel solution. Third, we assess the stability and scalability of our channel switching protocol on a multi-hop network on Testbed-Zurich with and without controlled jamming. Finally, we compare Nordica with a frequency-hopping multi-channel protocol [10] on Testbed-Delft.

5.3.2 Controlled Jamming

First, we compare the data yield and duty cycle of Nordica against that of a single-channel solution using controlled jamming. Since we are interested in evaluating the operations of Nordica's control loops, we perform these tests by manually choosing 802.15.4 channel 26 as the channel on which nodes begin communication.* All experiments run for at least an hour on Testbed-Delft in the presence of a jammer, in our setup a jamming Tmote sky node. We turn on the jammer after the nodes had sufficient time to form a routing tree to the sink ($\approx 15min$). We vary the so-called *jamming impact*, defined as the percentage of the time that the jammer node is active over a given period. This period, a so-called *epoch*, is fixed in our experiments to two minutes. We chose the *epoch* considering that while typical Wi-Fi bursts last much shorter, downloading a large file takes over a few minutes. Note that we additionally performed these tests with much shorter jamming *epochs*, and found similar results. We distinguish four different categories of jamming: *mild*, *moderate*, *extreme* and *full*. For mild jamming, 17% of the epoch the jammer is turned on, for moderate jamming the ratio of jamming is 50% of the epoch, for extreme jamming it is 83%, and for full jamming 100%.

Figure 5.8 shows the boxplots results of the controlled tests, that are aggregated over 5 tests for each jamming impact value. For each plot, we also computed the sample average. We see

*Channel 26 is relatively cleaner than the rest and allows for experiments with controlled interference.

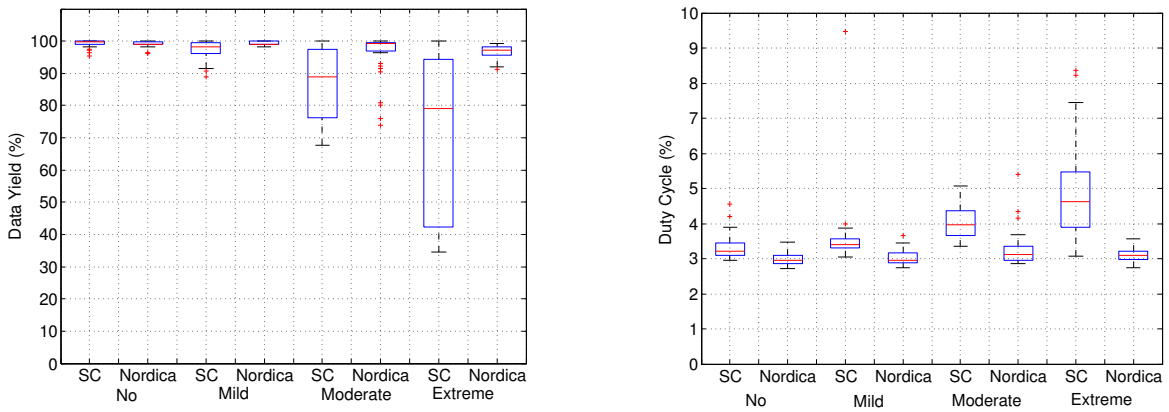


Figure 5.8: Data yield and duty cycle from controlled jamming on data yield (left) and duty cycle (right). The experiments include cases of *no*, *mild*, *moderate*, and *extreme* jamming. The horizontal axis shows these cases, which relate respectively to a jammer duty cycle of 0%, 17%, 50%, and 83%.

that when jamming increases, the data yield for the single-channel exhibits a greater sample variance, while Nordica sustains an almost similar distribution across varying jamming impacts. From our logs, we observed that the average data yield for the single channel stack drops as low as 70%. In contrast, Nordica sustains an average data yield of greater than 97%, with merely a momentary loss of packets that results from the channel switching co-ordination. We recorded that in 80% of the cases, the channel switch is initiated via the inner loop. For the rest, only a subset of child nodes were interfered by the jammer. In these cases, the scan mode was invoked after an outer loop switch on the affected child nodes. The outliers recorded for Nordica, particularly in the case of *Moderate* jamming, is thus largely attributed to the scan overhead. Note that while Nordica's yield is higher, its duty cycle is similar to or (for moderate and extreme jamming) even lower than for the single-channel solution. This is because communication on a noisy channel is very expensive mainly due to re-transmissions by the sender.

We performed detailed tests on the performance of the outer loop control for the case of *full* jamming on Testbed-Delft, with 8 child nodes and a sink, over 20 runs. We observed that both child nodes and the parent switched channels via the outer loop in all the runs, and reconnected on the next channel. The latency incurred in reconnecting parent nodes and their children on a new channel is influenced by jammer phase offsets ϕ^p and ϕ^c respectively, as detailed earlier in Section 5.2.4. Given the asynchronous operation of Nordica control loops on the nodes, it was decidedly non-trivial to fix the values for ϕ^p and ϕ^c at compile time. Nevertheless, we observed that in the worst case, both parent and child nodes switch channels at the next firing of the outer loop. The resulting latency to reconnect is bounded by $2 \times T_{outer}$, which places an upper bound on performance degradation in the case of *full* jamming.

5.3.3 Wi-Fi Interference

In the second set of tests, we compare the data yield and duty cycle of Nordica against a single-channel solution under real Wi-Fi interference. Testbed-Delft is located within the proximity of 3 Wi-Fi access points on the same floor. These access points occupy the Wi-Fi frequencies 1,

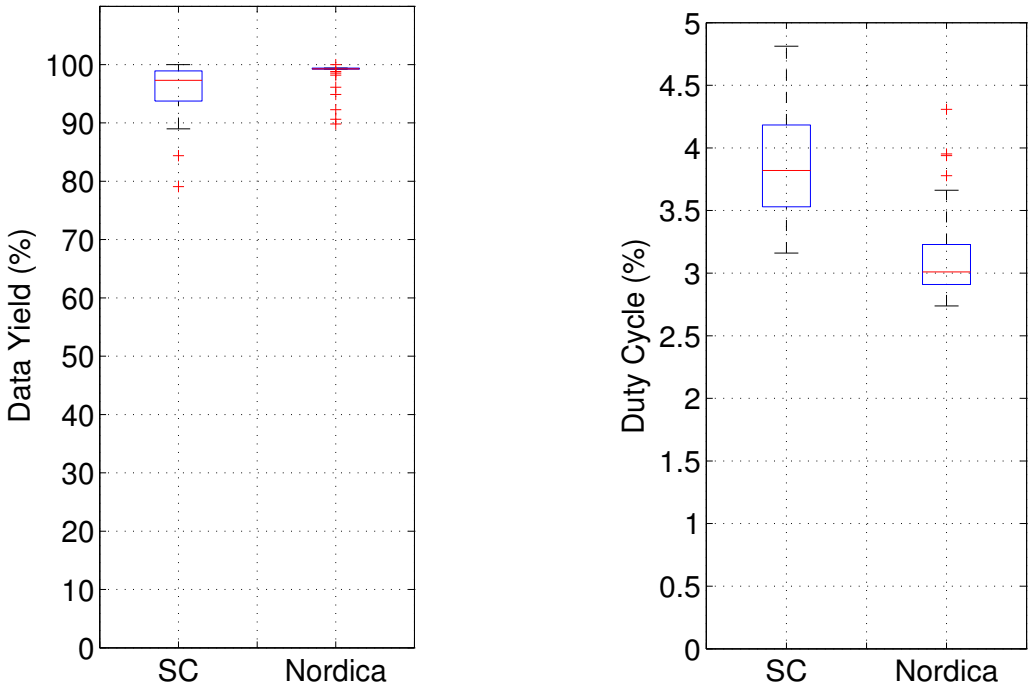


Figure 5.9: Results from moderate Wi-Fi interference on data yield (left) and duty cycle (right)

6 and 11, and exhibit intermittent bursts of activity throughout the day and hence provide a realistic, uncontrolled setting. These tests were done during day time when there was *moderate* amount of activity in the above mentioned Wi-Fi bands. Additionally, we wanted to test how Nordica would perform in comparison to the single-channel solution when there is a *high* activity on the Wi-Fi network. To this end, we initiated two file downloads of 693 MB each, while the data collection on the 802.15.4 channel was in progress. Depending upon the number of user requests over the Wi-Fi, such a download would last in the order of minutes. We choose an 802.15.4 channel that is close to the Wi-Fi hotspot to experiment with a realistic jamming impact. Note that this represents a worst-case scenario in that the external interference is on the same channel as the sensor nodes operate.

Figures 5.9 and 5.10 show the results from three experimental runs, each lasting for over an hour for both a moderate and a high amount of Wi-Fi activity. For moderate Wi-Fi activity (cf. Figure 5.9), Nordica achieves a marginally better data yield than the single-channel protocol, but consumes less energy. In this case, nodes running Nordica switched their outchannels at most once, showing a stable operation of the control loops.

Even under high Wi-Fi activity (cf. Figure 5.10), Nordica sustains an average data yield above 98%. Most of the nodes running Nordica switched their outchannels only once, via the inner loop. This indicates not only a stable operation of the control loops, but also the ability of Nordica to avoid interference effects on the spot. In contrast, the single-channel solution suffers from a poor average data yield (72%) and increased energy consumption under high Wi-Fi activity. For both moderate and high activity, our logs reveal that the energy overhead of the single channel stack is the result of the re-transmissions caused by the Wi-Fi interference.

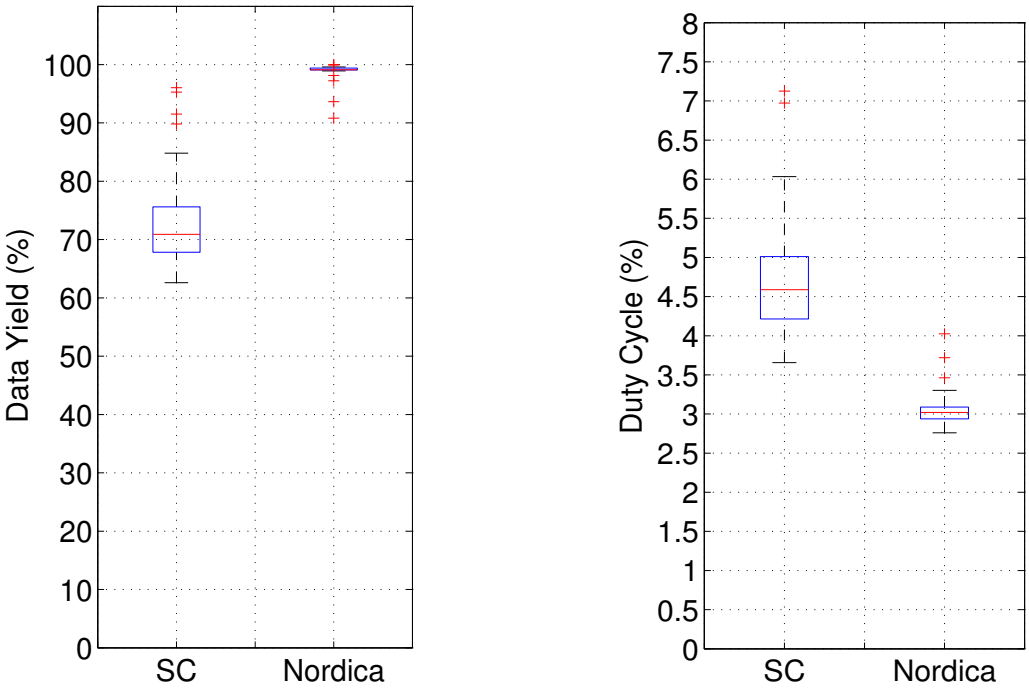


Figure 5.10: Results from high Wi-Fi interference on data yield (left) and duty cycle (right)

5.3.4 Multi-hop Setup

Additionally, we tested Nordica on Testbed-Zurich concerning a stable operation over a multi-hop network. These experiments were conducted during the night, so we attribute any performance degradation to unstable links rather than Wi-Fi activity. Nevertheless, it must be noted that Testbed-Zurich has challenging conditions due to contention caused by its dense deployment and multiple Wi-Fi access points in the building. Table 5.4 shows results from 3 runs, each lasting 10 hours. While the single-channel protocol stack achieves 86% data yield under normal operation, Nordica performs better with a data yield over 92%. Nordica also outperforms the single channel solution on the issue of routing stability; an average of 3 routing switches per node were observed, in contrast to 12 routing switches per node when the network operates on a single channel. This highlights the ability of Nordica to maintain a stable routing tree in the majority of cases, effectively only performing routing switches in the face of really unreliable communication.

To see how Nordica performs under the effect of external interference, we also selected a node near the sink as a jammer. We jammed the network (full jamming) for the duration of the experiment, after a bootstrap phase as described before. We performed 2 runs for each 10 hours. Table 5.4 shows that the data yield for the single-channel stack drops below 70%, while Nordica maintains a data yield greater than 90%. It must be noted that we ran the single channel implementation on channel 26, a relatively clean frequency. This shows that Nordica not only offers robustness against external interference, but also provides frequency diversity that guarantees a better performance than a single channel solution.

To assess the fidelity of the coordinated channel switching in a realistic network setup, we

Table 5.4: Evaluation on Testbed-Zurich. Data yield averaged over three experiments. Routing switches averaged per node, and over three experiments.

Jamming Type	*Metric*	*Single channel*	*Nordica*
No Jamming	Data Yield	86.3%	92.7%
	Routing switches	11.9	3.1
Full Jamming	Data Yield	65%	91%

Table 5.5: Inner loop performance from Testbed-Zurich. The values denote the percentage (%) of the total number of channel switches observed.

Inner loop	Outer loop	Channel scans	Routing change
85.7	8.6	4.7	1.0

analyzed the data from the experimental runs without jamming for incidences where a parent initiated a channel switch via the inner loop. For each incidence, we studied the number of child channel switches that resulted out of coordinated (inner loop) switching, autonomous switching (outer loop) and channel scans. We also looked at possible routing switches for a child node. Table 5.5 shows that 85.7% (349 switches out of 407 in total) of the channel switches for the child nodes are inner loop switches, coordinated by the parent node. In 8.6% (35 out of 407) of the cases, child nodes failed to receive the channel switch indication from their parent, and performed an outer loop switch. In contrast, only 4.7% of the channel switches resulted out of channel scans. This underlines the reason for Nordica being energy efficient: Channel switching is typically coordinated and thus has minimal energy overhead.

5.3.5 Comparison with a Frequency-Hopping Protocol

Finally, we performed a comparison with a frequency hopping protocol, MuChMAC [10]. We use the protocol implementation from the original authors with eight 802.15.4 channels and 100ms slots. Unfortunately MuChMAC experienced time synchronization, bootstrap, and neighbor management issues on Testbed-Delft. Note that these are typical issues for this class of protocols. In order to allow for a fair comparison, we used a simplified static topology of eight nodes, where MuChMAC performed flawlessly. We evaluated both MuChMAC and Nordica during daytime with Wi-Fi activity predominantly on Wi-Fi channels 1, 6 and 11 for the course of 12 hours.

While Nordica achieves a data yield of 98% by successfully evading interfered channels, MuChMAC only achieved 84% data yield. This is to be expected: a frequency hopping protocol is still affected when it hops to a channel that is interfered. In fact, when one out of 8 channels is blocked, we would expect under perfect conditions MuChMAC's data yield to be slightly higher at 87.5%. Nevertheless, this indicates that while frequency hopping ensures some form of robustness to external interference, it struggles with persistent channel interference. Hence, some form of blacklisting may be needed.

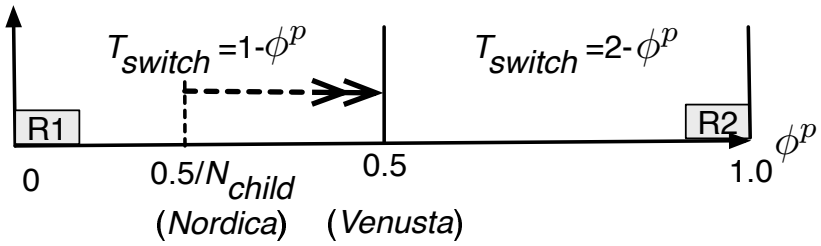


Figure 5.11: T^p_{switch} for different ranges of jammer phase ϕ_p.

5.4 Venusta: the second Chrysso generation

In this section, we present a refined protocol policy, and investigate the performance on this improved protocol variant called Venusta. In particular, we address the issue of a prolonged channel switching latency with Nordica as described in Section 5.2.4, i. e., we revise the outer loop policy for the parent node to reduce the channel switching latency. In the following, we describe in detail the rationale for the revised outer loop policy. Moreover, we provide simulation results that confirm Venusta's improvements on switching latency.

5.4.1 Outer loop switching time

Our analysis of Nordica in Section 5.2.4 identified a dependency of the switching latency of the outer loop in T^p_{switch} on the number of child nodes, i. e., N_{child}. Specifically, the greater the number of child nodes, the larger the value for $E[T^p_{switch}]$. We observe that the increased switching latency problem is due to the fact that Equation 5.2 does not consider the number of child nodes N_{child}. Hence, for Venusta we revise the inequality as follows:

$$N_{RX} < r_{rx,min} \times \frac{T_{outer}}{T_{data}} \times N_{child}. \tag{5.4}$$

Since $N_{RX} = (N_{child} \cdot \phi_p \cdot T_{outer})/T_{data}$, and $r_{rx,min} = 0.5$, the above inequality simplifies to: $\phi_p < 0.5$. Fig. 5.11 visualizes this change in policy: The region R_1 increases from Nordica to Venusta if a parent has more than a single child. This proportionally reduces the expected worst-case switching time of a parent. In particular we can calculate the expected value as:

$$E[T^p_{switch}] = \begin{cases} \frac{1}{2} \cdot ((1-0) + (1-0.5)) & \text{if } 0 \leq \phi_c < 0.5 \\ \frac{1}{2} \cdot ((2-0.5) + (2-1)) & \text{if } 0.5 \leq \phi_c \leq 1 \end{cases}$$

As before, the overall expected value for T^p_{switch} is the weighted average and simplifies to 1, i. e., it is one T_{outer} window. Hence, we a achieve a reduction in switching latency from Venusta to Nordica of $\frac{E[T^p_{switch}|Nordica]}{E[T^p_{switch}|Venusta]}$ of up to 1.5. Below we list the reduction of expected value for T^p_{switch}, across varying values of N_{child}.

Protocol	N_{child}				
	1	2	4	8	16
$\frac{E[T^p_{switch}\|Nordica]}{E[T^p_{switch}\|Venusta]}$	1.0	1.25	1.38	1.44	1.47

Note that this necessitates that each parent node monitors the number of its child nodes. As a consequence, Chrysso needs to consider the dynamics of parent switches in response to link quality fluctuations and their effect on N_{child}. To this end, Venusta performs a windowed observation of N_{child} at the parent nodes. Specifically, Venusta implements an exponentially weighted moving average (EWMA) for predicting the number of child nodes as shown below:

$$\hat{N}_{child}(t) = \alpha_N \times \hat{N}_{child}(t-1) + (1-\alpha_N) \times N_{child}(t-1). \tag{5.5}$$

$\hat{N}_{child}(t)$ denotes an estimate of the number of child nodes at time t, and is used in Equation (5.4). $N_{child}(t)$ denotes the *observed* number of child nodes at time t. The coefficient α_N is the EWMA filter parameter; we set $\alpha_N = 0.7$ in our implementation. We choose a time window of T_{outer} between consecutive updates, so that it involves minimal computational overhead, while still capturing changes in routing topology.

5.4.2 Policy validation

In order to test the effectiveness our of revised policy, we perform basic simulations using COOJA [104], on a simple topology comprising a parent node and several child nodes. We vary the number of child nodes in the range 4, 8 and 16. Note that these numbers represent topologies observed in real-world WSNs [105]. In each experiment, we choose an offset, such that $\phi^p \in (\frac{0.5}{N_{child}}, 0.5)$, since our policy update only affects this specific region. Specifically, we set the jammer to operate at a fixed phase offset $\phi^p = 0.27$. Figure 5.12 compares the results of Nordica and Venusta from the different simulation runs. Notably, we see that Venusta achieves a vast reduction in switching latency in this region, i. e., from approximately 325 seconds to 120 seconds – almost a factor of three reduction, which validates our claims of a improved switching latency for Venusta.

5.5 Venusta evaluation

In this section, we extend our evaluation using the revised protocol Venusta. We evaluate Venusta on a larger testbed, using current work on realistic jamming and demanding jamming scenarios.

5.5.1 Experimental setup

All experiments are performed on the *Twist* testbed [38], using a 54-node network. In all our experiments, the data collection tree is three hops deep, i. e., the maximum observed hop-distance from a node to the sink equals 3. We repeat each set of experiments twice. Experiments are performed during the night, and last a duration of 10 hours.

One node is randomly chosen as the sink, while the remaining nodes generate packets once every 3 minutes, such that $T_{data} = 180$ seconds. Therefore, in a 10 hour long experiment, every node generates 200 packets in total. We experiment with a low traffic rate primarily for two reasons: *(i)* low-power listening MAC protocols such as X-MAC [12] do not scale well with an increased volume of network traffic, especially for dense networks, and *(ii)* our experimental objective is to focus on Venusta's performance in mitigating external interference.

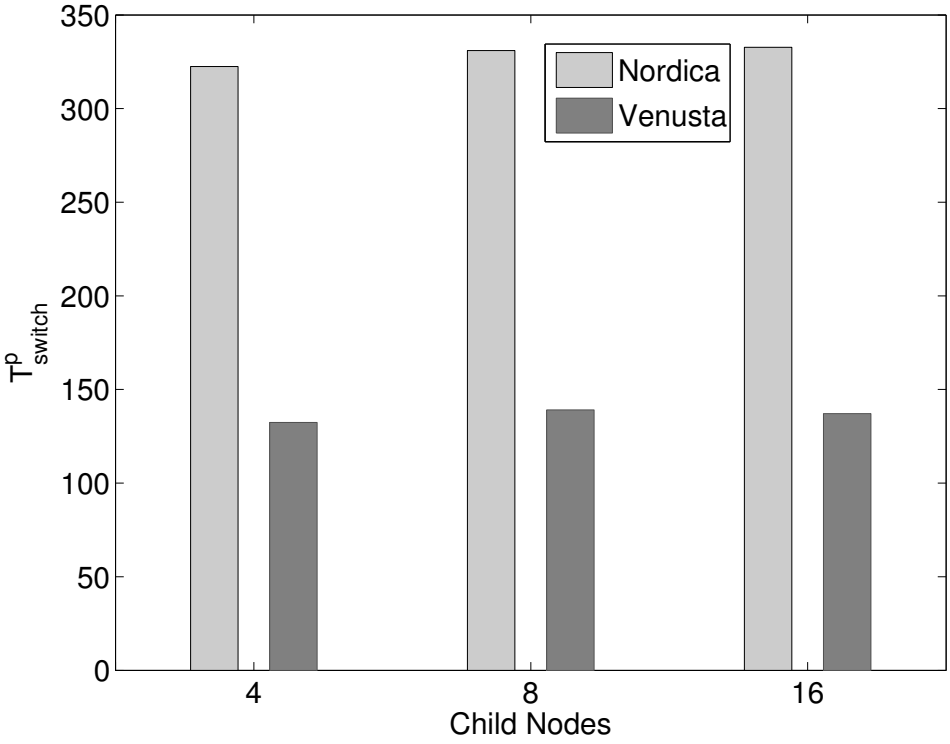


Figure 5.12: Switching latency (seconds) for a parent node, for different number of child nodes (horizontal axis). In each experiment, the jammer phase ϕ^p was set to 0.27. Smaller values for Venusta are a result of the revised outer loop policy.

We perform several experiments on Twist: *(i)* we record Venusta's performance under normal indoor operating conditions, *(ii)* we use JamLab [8] to investigate the effects of realistic WiFi interference, (*iii*) we test for *concurrent* jamming of multiple regions in the testbed and finally (*iv*) we evaluate Venusta in the face of jamming on *consecutive* channels. For all jamming test cases, we use realistic traces collected from a WiFi download [8] to emulate external interference. In our experiments, we set the jammer to start after $25 \cdot T_{data}$, i. e., approximately 75 minutes after network bootstrap. This ensures that the jammer begins to interfere after a stable routing topology has been setup. Similar to Section 5.3, we characterize the performance of both a single channel solution (labeled as *SC* for convenience) and Venusta based on data yield and radio-duty cycle. As with the experiments described earlier, a single channel protocol stack operates on channel 26, whereas with Venusta, the sink bootstraps on channel 26.

5.5.2 Large experiments on Twist

First we perform experiments without any injected interference to showcase Venusta's performance in typical indoor office environments. Figure 5.13 shows the distribution of data yield and duty cycle over the nodes, over different cases of external interference. We notice that the single channel solution exhibits greater variance in data yield, when compared to Venusta. This is attributed to the frequency diversity that Venusta provides by its multichannel bootstrap, thereby improving the average goodput. An offline inspection of the logs reveals that even in

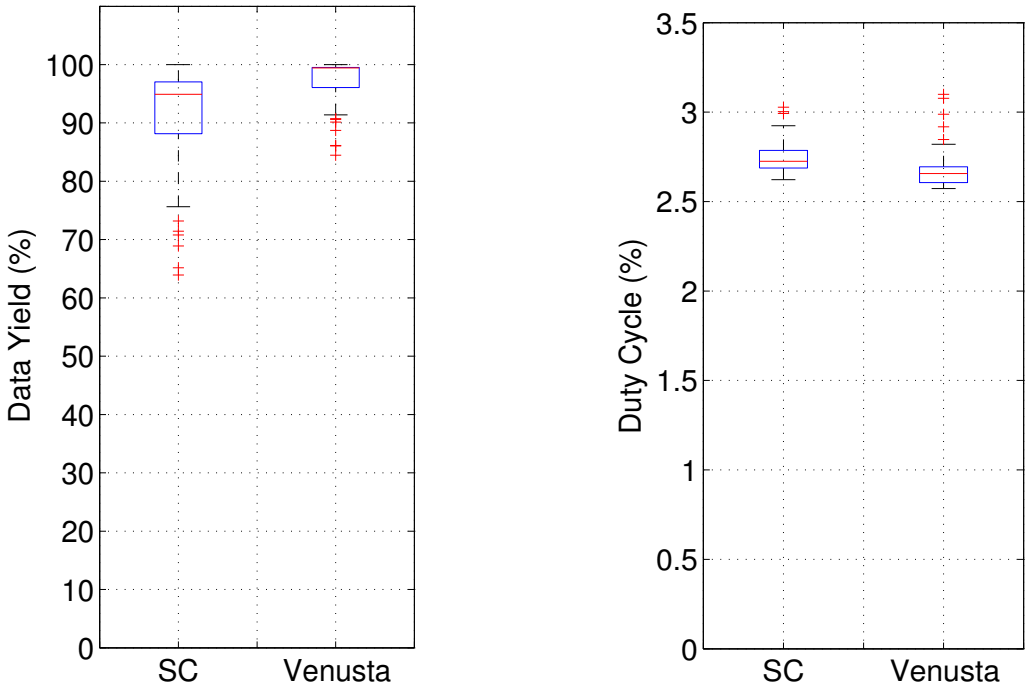


Figure 5.13: Data Yield (left) and Duty Cycle (right) achieved by 53 nodes, in the absence of external interference.

the absence of any induced external interference, the average data yield provided by Venusta is 97%, which is significantly higher than the 92% obtained by using a single channel. However, on the matter of energy consumption, we see that Venusta exhibits a few duty-cycle outliers, largely due to the overhead incurred by channel scanning. Nevertheless, the average (and median) duty cycle is slightly lower than what the single channel protocol achieves. This implies that Venusta provides a superior performance over the single channel protocol on a comparable energy expenditure.

5.5.3 Experiments with Jamlab

For the scenario of controlled jamming, we select a particular node close to the sink, that replays traces collected from a WiFi download using JamLab. The jammer turns on after network bootstrap, and lasts the entire duration of the experiment. Figure 5.14 shows the boxplot of data yield and duty cycle, for the single channel protocol and Venusta. We see that the single channel protocol shows an increased variance in both data yield and duty cycle as compared to Figure 5.13. In contrast, Venusta suffers minimally on account of external interference. Specifically, while the single channel protocol witnesses a significant drop in data yield, about 77% on average, Venusta sustains a stable average data yield of 96%, with only a marginal increase in variability. Furthermore, the duty cycle for Venusta does not change noticeably. This shows that any form of channel switching is carried out in an energy-efficient manner.

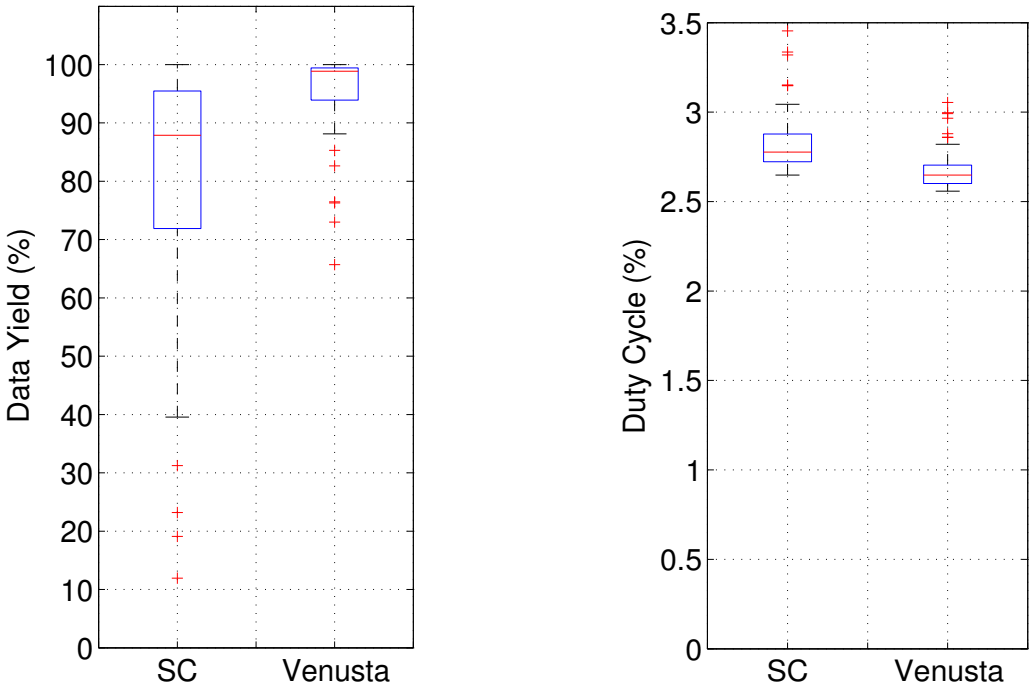


Figure 5.14: Data Yield (left) and Duty Cycle (right) achieved by 53 nodes, when the one-hop region around the sink is subject to external interference.

5.5.4 Concurrent Jamming

In the concurrent jamming scenario, we investigate the performance effects when two or more regions of a multi-hop network are simultaneously affected by external interference. Such a scenario may arise when a WiFi activity affects nodes stationed at different hop-distances from the sink. Our experimental objective is to mainly demonstrate that the data collection network formed by Venusta can be sustained, even when external interference disrupts communication at several regions. We emulate *concurrent* jamming by choosing two nodes in the data collection tree that are stationed one and two hops away from the sink.

Figure 5.15 shows the data yield and duty cycle achieved by both the single channel solution and Venusta, when the network is subjected to *concurrent* jamming. In general, the performance figures validate the ability of Venusta to perform channel switches concurrently at several regions in the network. Particularly, we observe that while the data yield for the single-channel solution reaches as low as 27%, Venusta produces only a few outliers with as low as 77% yield and mostly sustains a data yield close to 98%. Furthermore, the average duty cycle for Venusta is marginally lower than that achieved by a single-channel solution. This further confirms the duty cycle results presented in Figure 5.14. The increased energy consumption for the single channel solution occurs mainly because a greater fraction of the nodes are affected by concurrent jamming. In contrast, the channel switching policies for Venusta significantly reduce the number of nodes that are affected by the jammer for an extended duration. Note that Venusta co-ordinates channel switches between a parent node and its set of child nodes. It therefore handles the case of concurrent interferers *independently* at every affected parent-child pair, underlining Venusta's scalability.

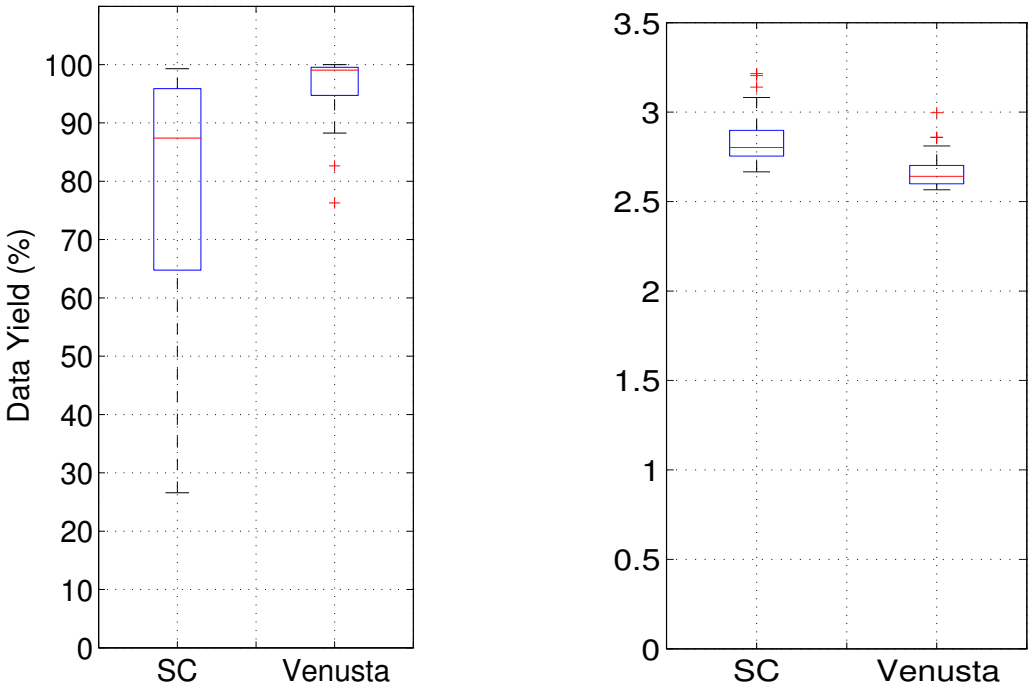


Figure 5.15: Data Yield (left) and Duty Cycle (right) achieved by 53 nodes, in the face of *concurrent* jamming.

5.5.5 Consecutive Jamming

Consecutive jamming refers to the test scenario of jamming of both the current channel and the *consecutive* channel that Venusta maintains on its channel list. Note that this is particularly challenging for Venusta as a parent-child pair that switches from the affected channel to the consecutive channel continues to be interfered, i. e., on the new channel the pair still cannot communicate. This scenario specifically tests the ability of Venusta to maintain a stable routing tree in the event of a prolonged channel switching process.

We evaluate Venusta for the case of consecutive jamming by deploying two interfering nodes near the sink. For simplifying the experiments, we constrain Chrysso's channel diversity by bootstrapping the sink node and its set of child nodes on channel 26. After the collection tree is set up, the jammer nodes emulate interference traces on channels 26 and 14, since channels 26 and 14 are consecutive channels in the implementation.

Table 5.6 details the percentile distribution of the resulting performance. Our results show that Venusta achieves on average, a data yield of 95.5% and duty cycle of 2.7%. In general, Venusta exhibits very few performance outliers. For example, the 25^{th} percentile of data yield is above 95%, and the lowest data yield is 64%. An inspection of the logs reveals that the 97.5% percentile for duty cycle represents an outlier that corresponds to a data yield of approximately 80%. This suggests that the outliers for duty cycle correspond to the outlier values for data yield. The observation is consistent with our reasoning that a reduced data yield for a node is accompanied by an increase of energy consumption, on account of increased retransmissions.

To conclude, our experiment validates the ability of Venusta to operate reliably in realistic indoor environments, in which interference may be spread over different, non-overlapping

Table 5.6: Table shows the performance achieved by Venusta in the face of *consecutive* jamming. The numbers represent percentiles.

	Percentile				
Metric	2.5	25	50	75	97.5
Data yield (%)	67.3	95.2	98	99.5	100
Duty cycle (%)	2.57	2.59	2.62	2.7	3.1

channels on the 2.4GHz spectrum.

5.6 Related Work

Multi-channel protocols are used for mitigating two different sources of interference: internal and external interference. In this work, we focus on related work addressing external interference. Protocols that mitigate internal interference are typically not designed to mitigate external interference, since: (1) They rely on offline approaches to channel allocation [43, 146] and hence do not help with external interference that is not predictable and varies over time. (2) They apply channel assignments to independent routing paths or trees [78, 138] and hence are not flexible enough to cope with the localized nature of external interference.

Previous work has proposed channel-hopping protocols [10, 61, 120, 139]; however, these protocols necessitate time synchronization between nodes. Chrysso does not require time synchronization and focuses on asynchronous, low-power listening MAC protocols. Nevertheless, we have shown a comparison to a channel-hopping protocol particularly designed for low-power sensor nodes [10] in Section 5.3. A recent work by Tang et al. [129] has incorporated the notion of channel-hopping on existing receiver-initiated MAC protocols for low-power wireless networks. As an added contribution, the authors design and implement an adaptive channel *blacklisting* feature that addresses the issue of external interference. While being elegant and efficient, *EM*-MAC is tailored to a receiver-initiated MAC protocol that implements channel hopping. Specifically, the channel *blacklisting* feature is carried out independently on nodes, which necessitates a frequency hopping mechanism in order to ensure a stable routing topology. In comparison, Chrysso allows a parent-child pair to communicate on a channel as long as it is not interfered. In this regard, Chrysso is suited for implementation over a broader range of MAC protocols that do not necessarily require either receiver-initiated data transfer or frequency hopping. All in all, *EM*-MAC and Chrysso can be perceived as complementary approaches for addressing external interference. In particular, *EM*-MAC can benefit from our work on protocol policies detailed in Section 5.2.

Our work is similar to the multi-channel protocol by Le et al. [70] that is specifically designed for data *aggregation* in WSNs. This is, however, subtly different from our focus on data *collection*, where packets are not processed in-network, but transmitted to the sink in their entirety. In particular, Le et al. propose aggregation-specific policies that cannot be applied to data collection. In contrast to Chrysso, the work does not focus on robustness to external interference, but rather on throughput maximization. As such, it focuses on switching complete

subtrees of an aggregation tree. This is a considerable overhead for deep trees and not necessary for localized external interference. Voigt et al. [136] developed a multi-channel feature that effects an adaptive channel switch when nodes find the current channel noisy. While the basic control mechanism is similar to Chrysso, the channel switching policy is very simplistic: a parent-children group switch channels when the parent has not received a data packet. Furthermore, it does not consider the case of localized interference, which may require a routing solution. The work by Kang et al. [55] is specifically targeted towards resolving external interference effects in ZigBee WPANs by the use of multiple radio channels. However, the authors assume a static routing topology, and do not account for dynamics at the network layer that could complicate channel allocation strategies. Every node within a cluster operates at the same channel, in contrast to our design that allows frequency diversity within a cluster of nodes. Furthermore, the multichannel protocol proposed in [55] has additional communication overhead such as *beacons* and *test frames* to detect external interference, which constitutes a significant overhead for duty-cycling radios.

Interestingly, Liang et al. [79] independently suggested a multi-channel protocol architecture that resembles the design of Chrysso. Although our work focuses on data collection WSNs, the *Channel Allocation* and *Channel Synchronization* components proposed by Liang et al. closely match Chrysso's *control loops* and *channel selection* in their functionality. While their proposed architecture was only conceptual at the time of publication, we actually have outlined the mechanisms for channel monitoring and control and also provide policies and how to derive them. In this regard, their work supports our approach to mitigate external interference and our research efforts for a more general use of Chrysso.

Finally, we note that most multi-channel solutions have been evaluated on a static routing topology. This fails to account for long-term link quality variations that necessitate routing changes and hence require a form of neighborhood discovery. An exception is Liang et al. [78], wherein nodes could connect to different routing trees on (possibly) different channels. However, their work concentrates on multiple data collection trees (using multiple sinks), each on an individual frequency, while Chrysso uses multiple channels within one data collection tree. We based our neighborhood discovery scheme upon this work.

5.7 Conclusions

This chapter addresses the issue of external interference hampering radio communications, such as Wi-Fi traffic affecting low-power 802.15.4 radios. We presented Chrysso, a multi-channel protocol extension, that leverages the channel diversity of typical sensor node radios. Chrysso addresses both the adaptability of inter-node communication in the face of external interference, and the discovery of neighbors on different channels. Chrysso mitigates the effects of external interference by switching (only) the affected nodes to a new set of channels. It is specifically tailored to data collection applications, and allocates channels for individual parent-children groups with the parent coordinating a channel switch upon detecting interference. For efficiency nodes normally operate on two channels only, one for incoming and one for outgoing traffic. We also present a novel scanning procedure probing all channels than can be used either at bootstrap or when losing contact with the parent.

As a proof of concept we integrated Chrysso in the Contiki protocol stack. Additionally, we developed Venusta, a variant of Chrysso that improves the channel switching latency of nodes when affected by external interference. We performed a detailed evaluation of Chrysso on three different testbeds, and on a large setup of nodes, under realistic conditions of external interference provided by Jamlab [8]. The results show the effectiveness of Chrysso in normal indoor operation, in the face of Wi-Fi interference and (controlled) jamming. A comparison with MuChMAC, a state-of-the-art channel hopping protocol, confirms that Chrysso outperforms such solutions for mitigating persistent external interference as it avoids the affected channels completely.

The results confirmed our expectations regarding data reliability and energy efficient neighborhood discovery of Chrysso with an increase of network scale. The localized co-ordination between nodes for mitigating interference, without the need for any additional messaging overhead, thus makes Chrysso a feasible choice for real-world WSN implementations.

Chapter 6

Conclusions and Future Work

In this thesis, we addressed the problem of decentralized adaptability and estimation in dynamic wireless networks characterized by node mobility and link changes. We studied typical problems in dynamic wireless networks, mostly related to the scheduling of communication and the estimation of network connectivity. Our contributions in this thesis can be summarized as follows: *(i)* we characterized the performance of desynchronization in mobile networks, and compared its performance against randomized mechanisms that are easier to implement, *(ii)* we designed and developed *Nest*, an adaptive algorithm for neighborhood discovery in mobile works, *(iii)* we designed *PathDetect*, an algorithm that estimates in real-time, and in a decentralized manner, the temporal connectivity in mobile networks, and *(iv)* we designed, developed and implemented on different testbeds, *Chrysso*, an adaptive multichannel protocol for data gathering WSNs that focuses specifically on mitigating external interference.

6.1 Conclusions

Concerning the issue of topological dynamics, adaptability and estimation in general, we have observed the following:

1. The use of explicit scheduling approaches such as desynchronization for inter-node communication works best on static and sparse topologies. However, at high values of node density and mobility, the use of a randomized *beaconing* yields a performance comparable to that achieved by desynchronization. As randomized mechanisms operate independently and involve no overhead, they are to be preferred for the case of dynamic networks. For static networks though, the problem of message collisions can be mitigated through explicit signalling mechanisms that do not require additional infrastructure such as time synchronization.

2. In dense and dynamic neighborhoods, estimating how many nodes there are in a neighborhood is quicker than discovering the identities of all neighboring nodes. Knowing the node density is fundamental and crucial, and helps to achieve efficient inter-node communication. Neighborhood density can be estimated by observing the statistics of the channel events, i. e., channel *idle* and channel *busy* times. This requires a continuous monitoring of the channel, and therefore, presents a processing overhead. In contrast, statistics of

received messages are relatively simpler to generate, but are prone to being biased by the *signal capture effect*. We therefore, advocate monitoring the channel for *busy* and *idle* events for the purpose of estimating neighborhood density.

3. Estimating the connectivity in mobile networks is hampered by the fact that the topology is almost never connected at any point in time. However, it is possible to reason about connectivity in a temporal sense, i. e., understand how long it takes for messages to disseminate throughout the network. We have shown that it is possible to design a decentralized algorithm that estimates locally on every node, the temporal connectivity in a mobile network. By combining local observations on temporal connectivity with distributed consensus, a global and network-wide view on temporal connectivity can be constructed on every node. Additionally, such a mechanism proved effective in the case of dynamics, by tracking changes in network connectivity in real-time.

4. Mitigation of external interference is a relevant problem for reliable and energy-efficient communication in low-power wireless sensor networks. Considering that external interference typically affects a subset of available frequency channels, nodes affected by it could change to a non-interfered channel to resume communication. We have shown through our design and implementation of the *Chrysso* multichannel protocol that it is possible for nodes to seamlessly adapt their operation in the face of external interference, with minimal overhead on energy consumption.

To summarize, we conclude that for the case of static networks, decentralized adaptability in communication is easily achieved through the use of explicit control signal notifications. However, in the more difficult case involving mobile networks, it is recommended to stay with randomized mechanisms for communication. Furthermore, estimating a quantity such as neighborhood density or network connectivity can be derived by distributed consensus, in which the value that the nodes converge to is continuously updated with changes in topology.

6.2 Future Work

This thesis focused on the design of decentralized algorithms for adaptability and estimation, and evaluated them with a monolithic application running on the nodes. However, with the increased usage of wireless devices, the concept of a single, monolithic application is expected to be replaced by one in which a node will execute multiple applications. The rise of multiple wireless applications leaves several implementation challenges to be addressed, such as isolating the adaptability component from the applications, and ensuring communication reliability for network protocols. We believe these challenges offer the following options for extending our work:

1. Preliminary experiments with the *Nest* neighborhood discovery algorithm have shown promising results on estimation accuracy. A natural extension would include applying *Nest* to more practical scenarios in which nodes duty cycle their radios, and in which neighborhood discovery runs alongside multiple applications, without interfering with them. The

objective is to implement a low-level discovery service that multiple applications running on a network could share.

2. With an increase in the number of applications that run on a network, it becomes even more necessary to compress and aggregate the large volume of generated data. Especially, the use of gossip-based aggregation protocols is most suited for mobile networks, as they do not rely on any infrastructure. However, gossip-based data aggregation is found to be sensitive to communication errors. Therefore, it is important to adapt aggregation-specific policies in the face of changing node neighborhoods and intermittent links. Particularly, being able to notify message failures (i. e., message collisions) offers valuable feedback that could be used to trigger corrective actions at the node-level.

Bibliography

[1] Alwen project. `www.alwen.nl`, 2012. 1

[2] T.C. Aysal, A.D. Sarwate, and A.G. Dimakis. Reaching consensus in wireless networks with probabilistic broadcast. In *Proc. of IEEE Allerton Conference, 2009*, pages 732–739. 39, 60

[3] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli. The hitchhiker's guide to successful wireless sensor network deployments. In *Proc. of ACM SenSys'08*, pages 43–56. 5

[4] YM Baryshnikov, EG Coffman, and KJ Kwak. High performance sleep-wake sensor systems based on cyclic cellular automata. In *Proc. of ACM IPSN'08*, pages 517–526. 7, 27

[5] P. Basu, A. Bar-Noy, R. Ramanathan, and M.P. Johnson. Modeling and analysis of time-varying graphs. *Arxiv preprint arXiv:1012.0260*, 2010. 63, 78

[6] J. Beutel, S. Gruber, A. Hasler, R. Lim, A. Meier, C. Plessl, I. Talzi, L. Thiele, C. Tschudin, and M. Woehrle. Permadaq: A scientific instrument for precision sensing and data recovery in environmental extremes. In *Proc. of ACM IPSN'09*, pages 265–276. 2

[7] C.A. Boano, K. Römer, Z. He, T. Voigt, M.A. Zúñiga, and A. Willig. Generation of controllable radio interference for protocol testing in wireless sensor networks. In *Proc. of ACM SenSys'09*, pages 301–302. 95

[8] C.A. Boano, T. Voigt, C. Noda, K. Romer, and M. Zúñiga. Jamlab: Augmenting sensornet testbeds with realistic and controlled interference generation. In *Proc. of ACM IPSN'11*, pages 175–186. 83, 102, 108

[9] S.A. Borbash, A. Ephremides, and M.J. McGlynn. An asynchronous neighbor discovery algorithm for wireless sensor networks. *Ad Hoc Networks*, 5(7):998–1016, 2007. 8, 35, 36

[10] J. Borms, K. Steenhaut, and B. Lemmens. Low-overhead dynamic multi-channel MAC for wireless sensor networks. In *Proc. of Springer EWSN'10*, pages 81–96. 95, 99, 106

[11] J. Broch, D.A. Maltz, D.B. Johnson, Y.C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proc. of ACM MobiCom'98*, pages 85–97. 45, 66

[12] M. Buettner, G. Yee, E. Anderson, and R. Han. X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In *Proc. of ACM SenSys'06*, pages 307–320. 87, 101

[13] Q. Cao, T. Abdelzaher, T. He, and J. Stankovic. Towards optimal sleep scheduling in sensor networks for rare-event detection. In *Proc. of ACM IPSN'05*, pages 1–8. 27

[14] M. Ceriotti. *Guaranteeing Communication Quality in Real World WSN Deployments*. PhD thesis, University of Trento, 2011. 7, 13

[15] R. Cohen and B. Kapchits. Continuous neighbor discovery in asynchronous sensor networks. *IEEE/ACM Transactions on Networking (TON)*, 19(1):69–79, 2011. 8, 35

[16] A. Cornejo and F. Kuhn. Deploying wireless networks with beeps. *Distributed Computing*, pages 148–162, 2010. 14

[17] Datasheet. Telosb. crossbow technologies. 50, 86

[18] J. Degesys and R. Nagpal. Towards desynchronization of multi-hop topologies. In *Proc. of IEEE SASO'08*, pages 129–138. 7, 13, 16, 31

[19] J. Degesys, I. Rose, A. Patel, and R. Nagpal. DESYNC: self-organizing desynchronization and TDMA on wireless sensor networks. In *Proc. of ACM IPSN'07*, pages 11–20. 7, 13, 15, 16, 31

[20] G. Dimitrakopoulos and P. Demestichas. Intelligent transportation systems. *Vehicular Technology Magazine, IEEE*, 5(1):77–84, 2010. 2

[21] A. Dunkels, B. Gronvall, and T. Voigt. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *Proc. of IEEE LCN'04*, volume 462, pages 455–462, 2004. 86

[22] A. Dunkels, L. Mottola, N. Tsiftes, F. Osterlind, J. Eriksson, and N. Finne. The Announcement Layer: Beacon Coordination for the Sensornet Stack. In *Proc. of Springer EWSN'11*, pages 211–226. Springer. 87

[23] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He. Software-based on-line energy estimation for sensor nodes. In *Proc of ACM SenSys'07*, pages 28–32. 95

[24] P. Dutta and D. Culler. Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In *Proc. of ACM SenSys'08*, pages 71–84. 8, 35, 37

[25] M. Dyer, J. Beutel, T. Kalt, P. Oehen, L. Thiele, K. Martin, and P. Blum. Deployment support network a toolkit for the development of wsns. In *Proc. of Springer EWSN'07*, EWSN'07, pages 195–211. 95

[26] N. Eagle and A. Pentland. Reality mining: sensing complex social systems. *Personal and Ubiquitous Computing*, 10(4):255–268, 2006. 2

[27] A. El-Hoiydi and J.D. Decotignie. WiseMAC: an ultra low power MAC protocol for the downlink of infrastructure wireless sensor networks. In *ISCC 2004*, pages 244–251, 2004. 87

[28] J.J. Enright and P.R. Wurman. Optimization and coordinated autonomy in mobile fulfillment systems. In *Proc. of AAAI'11*, volume 1, page 2. 2

[29] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In *Proc. of ACM IPSN'11*, pages 73–84, 2011. 5

[30] J. Gao, L. Guibas, N. Milosavljevic, and J. Hershberger. Sparse data aggregation in sensor networks. In *Proc. of ACM IPSN'07*, pages 430–439, 2007. 4

[31] JJ Garcia-Luna-Aceves and A.N. Masilamani. Nomad: Deterministic collision-free channel access with channel reuse in wireless networks. In *Proc. of IEEE SECON'11*, pages 1–9. 14

[32] D. Gavidia and M. Van Steen. A probabilistic replication and storage scheme for large wireless networks of small devices. In *Proc. of IEEE MASS'08*, pages 469–476, 2008. 3

[33] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proc. of ACM SenSys'09*, pages 1–14. 4, 5, 87

[34] A. Gopalan, S. Banerjee, A.K. Das, and S. Shakkottai. Random mobility and the spread of infection. In *Proc. of IEEE INFOCOM'11*, pages 999–1007. 8, 59, 78

[35] R. Groenevelt, P. Nain, and G. Koole. The message delay in mobile ad hoc networks. *Performance Evaluation*, 62(1):210–228, 2005. 8, 78

[36] S. Guna. *On Neighbors, Groups and Application Invariants in Mobile Wireless Sensor Networks*. PhD thesis, DISI - University of Trento, 2011. 35

[37] H. Han, B. Sheng, C.C. Tan, Q. Li, W. Mao, and S. Lu. Counting rfid tags efficiently and anonymously. In *Proc. of IEEE INFOCOM'10*, pages 1–9. 36

[38] V. Handziski, A. Köpke, A. Willig, and A. Wolisz. Twist: a scalable and reconfigurable testbed for wireless indoor experiments with sensor networks. In *Proc. of ACM REALMAN'06*, pages 63–70. 101

[39] M. Hefeeda and H. Ahmadi. A probabilistic coverage protocol for wireless sensor networks. In *Proc. of IEEE ICNP'07*, pages 41–50. 27

[40] M. Heusse, F. Rousseau, R. Guillier, and A. Duda. Idle sense: an optimal access method for high throughput and fairness in rate diverse wireless lans. In *Proc. of ACM SIGCOMM'05*, volume 35, pages 121–132. 36

[41] C. Hsin and M. Liu. Network coverage using low duty-cycled sensors: random & coordinated sleep algorithms. In *Proc. of ACM IPSN'04*, pages 433–442. 7, 27, 31

[42] M. Huang, S. Chen, Y. Zhu, B. Xu, and Y. Wang. Topology control for time-evolving and predictable delay-tolerant networks. In *Proc. of IEEE MASS'11*, pages 82–91, 2011. 78

[43] O.D. Incel, L. van Hoesel, P. Jansen, and P. Havinga. MC-LMAC: A multi-channel MAC protocol for wireless sensor networks. *Ad Hoc Networks*, pages 73–94, 2011. 106

[44] F. Ingelrest, N. Mitton, and D. Simplot-Ryl. A turnover based adaptive hello protocol for mobile ad hoc and sensor networks. In *Proc. of IEEE MASCOTS'07*, pages 9–14. 8

[45] Texas Instruments. CC2420 datasheet, 2007. 52

[46] V.G. Iyer, A. Loukas, S.O. Dulman, and K.G. Langendoen. Nest: A practical algorithm for neighborhood discovery in dynamic wireless networks using adaptive beaconing. Technical Report ES-2012-01, Delft University of Technology. 19

[47] V.G. Iyer, A. Pruteanu, and S.O. Dulman. Netdetect: Neighborhood discovery in wireless networks using adaptive beacons. In *Proc. of IEEE SASO'11*, pages 31–40. 19, 31, 36, 39, 41, 53

[48] V.G. Iyer, M. Woehrle, and K.G. Langendoen. Chamaeleon - exploiting multiple channels to mitigate interference. In *Proc. of IEEE INSS'10)*, pages 65–68. 82

[49] V.G. Iyer, M. Woehrle, and K.G. Langendoen. Chrysso - a multi-channel approach to mitigate external interference. In *Proc. of IEEE SECON'11*, pages 422–430, jun. 82

[50] P. Jacquet, B. Mans, and G. Rodolakis. Information propagation speed in mobile and delay tolerant networks. *IEEE Transactions on Information Theory*, 56(10):5001–5015, 2010. 78

[51] L. Jiang and J. Walrand. A distributed csma algorithm for throughput and utility maximization in wireless networks. *ACM Transactions on Networking (TON'10)*, 18(3):960–972. 7, 13, 27, 31

[52] D.B. Johnson and D.A. Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile computing*, pages 153–181, 1996. 45, 66

[53] D. Julian, M. Chiang, D. O Neill, and S. Boyd. QOS and fairness constrained convex optimization of resource allocation for wireless cellular and ad hoc networks. In *Proc. of IEEE INFOCOM'02*, volume 2, pages 477–486. 7, 13, 31

[54] A. Kandhalu, K. Lakshmanan, and R.R. Rajkumar. U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol. In *Proc. of ACM IPSN'10*, pages 350–361. 8, 35, 37

[55] M.S. Kang, J.W. Chong, H. Hyun, S.M. Kim, B.H. Jung, and D.K. Sung. Adaptive interference-aware multi-channel clustering algorithm in a Zigbee network in the presence of WLAN interference. In *2nd Int. Symposium on Wireless Pervasive Computing (ISWPC'07)*, 2007. 107

[56] S. Kaul, M. Gruteser, V. Rai, and J. Kenney. Minimizing age of information in vehicular networks. In *Proc. of IEEE SECON'11*, pages 350–358. 2, 8, 36

[57] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *FOCS 2003*, pages 482–491, 2003. 4, 38, 39, 60

[58] A. Keshavarzian, H. Lee, and L. Venkatraman. Wakeup scheduling in wireless sensor networks. In *Proc. of ACM MobiHoc'06*, pages 322–333. 7, 27

[59] M. Kim, D. Kotz, and S. Kim. Extracting a mobility model from real user traces. In *Proc. of IEEE INFOCOM'06*, volume 6. Citeseer. 77

[60] T. Kim, E. McFee, D.O. Olguin, B. Waber, and A. Pentland. Sociometric badges: Using sensor technology to capture new forms of collaboration. *Journal of Organizational Behavior*, pages 412–427, 2012. 2

[61] Y. Kim, H. Shin, and H. Cha. Y-MAC: An energy-efficient multi-channel MAC protocol for dense wireless sensor networks. In *IPSN '08*, pages 53–63, 2008. 106

[62] D.J. Klein, J. Hespanha, and U. Madhow. A reaction-diffusion model for epidemic routing in sparsely connected manets. In *Proc. of IEEE INFOCOM'10*, pages 1–9. 8, 78

[63] L. Kleinrock and F. Tobagi. Packet switching in radio channels: part i–carrier sense multiple-access modes and their throughput-delay characteristics. *IEEE Transactions on Communications*, 23(12):1400–1416, 1975. 43

[64] M. Kodialam and T. Nandagopal. Fast and reliable estimation schemes in RFID systems. In *Proc. of ACM MobiCom'06*, pages 322–333. 34, 36, 41, 47, 53, 57

[65] Z. Kong and E.M. Yeh. On the latency for information dissemination in mobile wireless networks. In *Proc. of ACM MobiHoc'08*, pages 139–148. 8, 78

[66] A. Krohn, M. Beigl, and S. Wendhack. Sdjs: Efficient statistics in wireless networks. In *Proc. of IEEE ICNP'04*, pages 262–270. 36

[67] Y. Kwon, Y. Fang, and H. Latchman. A novel mac protocol with fast collision resolution for wireless lans. In *Proc. of IEEE INFOCOM'03*, volume 2, pages 853–862. 7, 31

[68] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *14th Int. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, pages 8–15, 2006. 2, 5

[69] C. Law, K. Lee, and K.Y. Siu. Efficient memoryless protocol for tag identification. In *Proc. of the ACM DIAL-M'00*, pages 75–84. 36

[70] H. K. Le, D. Henriksson, and T. Abdelzaher. A control theory approach to throughput optimization in multi-channel collection sensor networks. In *Proc. of ACM IPSN '07*, pages 31–40. 106

[71] J. Lee, W. Kim, S.J. Lee, D. Jo, J. Ryu, T. Kwon, and Y. Choi. An experimental study on the capture effect in 802.11 a networks. In *Proc. of ACM WinTECH '07*, pages 19–26. 53

[72] J. Lee, J. Ryu, S.J. Lee, and T.T. Kwon. Improved modeling of ieee 802.11 a phy through fine-grained measurements. *Computer Networks*, 54(4):641–657, 2010. 53

[73] K. Lee, S. Hong, S.J. Kim, I. Rhee, and S. Chong. Slaw: A new mobility model for human walks. In *Proc. of IEEE INFOCOM'09*, pages 855–863. 45, 66, 75

[74] U. Lee, B. Zhou, M. Gerla, E. Magistretti, P. Bellavista, and A. Corradi. Mobeyes: smart mobs for urban monitoring with a vehicular sensor network. *Wireless Communications, IEEE*, 13(5):52–57, 2006. 37

[75] K. Leentvaar and J. Flint. The capture effect in FM receivers. *IEEE Transactions on Communications*, 24(5):531–539, 1976. 50, 53

[76] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al. Tinyos: An operating system for sensor networks. *Ambient intelligence*, 35, 2005. 50

[77] P.A. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proc. of USENIX NDSI'04*, pages 1–14. 3

[78] C. J. M. Liang, J. Liu, L. Luo, A. Terzis, and F. Zhao. RACNet: a high-fidelity data center sensing network. In *Proc. of ACM SenSys'09*, pages 15–28, NY, USA. 106, 107

[79] C.J.M. Liang and A. Terzis. Rethinking multi-channel protocols in wireless sensor networks. In *Proc. of ACM HotEmnets '10*, pages 1–5. 107

[80] A. Lindgren, A. Doria, and O. Schelén. Probabilistic routing in intermittently connected networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 7(3):19–20, 2003. 4

[81] J. Lu and K. Whitehouse. Flash flooding: Exploiting the capture effect for rapid flooding in wireless sensor networks. In *Proc. of IEEE INFOCOM'09*, pages 2491–2499. 5, 53

[82] J. Luo and D. Guo. Compressed neighbor discovery for wireless ad hoc networks: the Rayleigh fading case. In *IEEE Allerton Conf. on Communication, Control, and Computing.*, pages 308–313, 2009. 35

[83] S.R. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. Tinydb: An acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems (TODS)*, 30(1):122–173, 2005. 4

[84] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. The flooding time synchronization protocol. In *Proc. of ACM SenSys'04*, pages 39–49. 5

[85] F.J. Martinez, C.K. Toh, J.C. Cano, C.T. Calafate, and P. Manzoni. Emergency services in future intelligent transportation systems based on vehicular communication networks. *Intelligent Transportation Systems Magazine, IEEE*, 2(2):6–20, 2010. 2

[86] M.J. McGlynn and S.A. Borbash. Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks. In *Proc of ACM MobiHoc'01*, pages 137–145. 8, 35, 36, 38

[87] S. Merugu, M.H. Ammar, and E.W. Zegura. Routing in space and time in networks with predictable mobility. Technical Report GIT-CC-04-07, 2004. 8, 59, 78

[88] S. Moeller, A. Sridharan, B. Krishnamachari, and O. Gnawali. Routing without routes: The backpressure collection protocol. In *Proc. of ACM IPSN'10*, pages 279–290. 4

[89] E. Monton, JF Hernandez, JM Blasco, T. Herve, J. Micallef, I. Grech, A. Brincat, and V. Traver. Body area network for wireless patient monitoring. *Communications, IET*, 2(2):215–222, 2008. 2

[90] M. Motani, V. Srinivasan, and P.S. Nuggehalli. Peoplenet: engineering a wireless virtual social network. In *Proc. of ACM MobiCom'05*, pages 243–257. 37

[91] A. Motskin, T. Roughgarden, P. Skraba, and L. Guibas. Lightweight coloring and desynchronization for networks. In *Proc. of IEEE INFOCOM'09*, pages 2383–2391. 7, 14, 18, 19, 27, 31

[92] Luca Mottola, Gian Pietro Picco, Matteo Ceriotti, Stefan Guna, and Amy L. Murphy. Not all wireless sensor networks are created equal : A comparative study on tunnels. *ACM Transactions on Sensor Networks*, 7:15, 2010. 37

[93] S. Mueller, R. Tsang, and D. Ghosal. Multipath routing in mobile ad hoc networks: Issues and challenges. *Springer Performance Tools and Applications to Networked Systems*, pages 209–234, 2004. 8, 33

[94] C. Mühlberger and R. Kolla. Extended desynchronization for multi-hop topologies. Technical report, Tech. Rep. 460, Institut für Informatik, Universität Würzburg (Jul 2009), 2009. 16

[95] R. Musaloiu-E and A. Terzis. Minimising the effect of wifi interference in 802.15. 4 wireless sensor networks. *International Journal of Sensor Networks*, 3(1):43–54, 2008. 88

[96] J. Myung and W. Lee. Adaptive binary splitting: a rfid tag collision arbitration protocol for tag identification. *Mobile networks and applications*, 11(5):711–722, 2006. 2, 8, 33, 36

[97] M. Nabi, T. Basten, M. Geilen, M. Blagojevic, and T. Hendriks. A robust protocol stack for multi-hop wireless body area networks with transmit power adaptation. In *Proc. of ACM BodyNets'11*, volume 8, pages 77–83, 2010. 2

[98] T. Nandagopal, T.E. Kim, X. Gao, and V. Bharghavan. Achieving mac layer fairness in wireless packet networks. In *Proc. of ACM Mobicom'00*, pages 87–98. 31

[99] S. Nath, P.B. Gibbons, S. Seshan, and Z.R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of ACM SenSys'04*, pages 250–262. 4

[100] V. Navda, A.P. Subramanian, K. Dhanasekaran, A. Timm-Giel, and S. Das. MobiSteer: using steerable beam directional antenna for vehicular network access. In *Proc. of ACM MobiSys'07*, pages 192–205. 37

[101] N.P. Nguyen, T.N. Dinh, Y. Xuan, and M.T. Thai. Adaptive algorithms for detecting community structure in dynamic social networks. In *Proc. of IEEE INFOCOM'11*, pages 2282–2290. 8, 59

[102] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proc. of ACM MOBICOM'99*, pages 151–162, New York, NY, USA. 3

[103] R. Olfati-Saber, J.A. Fax, and R.M. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007. 65

[104] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *IEEE LCN' 06*, pages 641–648. Ieee. 101

[105] J. Paek and R. Govindan. Rcrt: rate-controlled reliable transport for wireless sensor networks. In *Proc. of ACM SenSys'07*, pages 305–319. 101

[106] W. Pan, W. Dong, M. Cebrian, T. Kim, JH Fowler, and AS Pentland. Modeling dynamical influence in human interaction: Using data to make better inferences about influence within social systems. *Signal Processing Magazine, IEEE*, 29(2):77–86, 2012. 2

[107] S.V. Pemmaraju and I.A. Pirwani. Energy conservation via domatic partitions. In *Proc. of ACM MobiHoc'06*, pages 143–154. 27, 31

[108] A. Pentland. Honest signals: how social networks shape human behavior. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 583–584. ACM, 2011. 2

[109] M. Piórkowski, N. Sarafijanovic-Djukic, and M. Grossglauser. A parsimonious model of mobile partitioned networks with clustering. In *In Proc. of IEEE COMSNETS'09*, pages 1–10. 45

[110] A.S. Pruteanu, V.G. Iyer, and S.O. Dulman. Churndetect: A gossip-based churn estimator for large-scale dynamic networks. In *Proc. of Springer Euro-Par'11*, pages 289–301. 4, 8, 33

[111] A.S. Pruteanu, V.G. Iyer, and S.O. Dulman. Faildetect: Gossip-based failure estimator for large-scale dynamic networks. In *Proc. of IEEE ICCCN'11*, pages 1–6. 4

[112] V. Rajendran, K. Obraczka, and J.J. Garcia-Luna-Aceves. Energy-efficient, collision-free medium access control for wireless sensor networks. *Wireless Networks*, 12(1):63–78, 2006. 31

[113] I. Rhee, A. Warrier, M. Aia, J. Min, and M.L. Sichitiu. Z-mac: a hybrid mac for wireless sensor networks. *IEEE/ACM Transactions on Networking (TON)*, 16(3):511–524, 2008. 5

[114] N. Sadagopan, F. Bai, B. Krishnamachari, and A. Helmy. Paths: analysis of path duration statistics and their impact on reactive manet routing protocols. In *Proc. of ACM MobiHoc'03*, pages 245–256, 2003. 8, 59

[115] Y.E. Sagduyu and A. Ephremides. Energy-efficient collision resolution in wireless ad-hoc networks. In *Proc. of IEEE INFOCOM'03*, volume 1, pages 492–502. 7, 13, 18

[116] S. Scellato, I. Leontiadis, C. Mascolo, P. Basu, and M. Zafer. Understanding robustness of mobile networks through temporal network measures. In *Proc. of IEEE INFOCOM'11*, pages 1–5, 2011. 8, 59, 62, 63, 68, 78

[117] R. Schmidt, T. Leinmuller, E. Schoch, F. Kargl, and G. Schafer. Exploration of adaptive beaconing for efficient intervehicle safety communication. *IEEE Network*, 24(1):14–19, 2010. 8, 33, 36

[118] V. Shnayder, B. Chen, K. Lorincz, T.R.F. Fulford-Jones, and M. Welsh. Sensor networks for medical care. In *Proc. of ACM SenSys'05*, pages 314–314. 2

[119] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: new aggregation techniques for sensor networks. In *Proc. of ACM SenSys'04*, pages 239–249. 4

[120] H.S. So, W. Walrand, and J.J. Mo. McMAC: A parallel rendezvous multi-channel MAC protocol. In *Proc. of IEEE WCNC'07*, pages 334–339. 106

[121] C. Sommer, O.K. Tonguz, and F. Dressler. Traffic information systems: efficient message dissemination via adaptive beaconing. *Communications Magazine, IEEE*, 49(5):173–179, 2011. 8, 36

[122] T. Spyropoulos, K. Psounis, and C.S. Raghavendra. Efficient routing in intermittently connected mobile networks: the multiple-copy case. *IEEE/ACM Transactions on Networking*, 16(1):77–90, 2008. 4

[123] A. Sridharan and B. Krishnamachari. Maximizing network utilization with max–min fairness in wireless sensor networks. *Wireless Networks*, 15(5):585–600, 2009. 31

[124] M. Steine, M. Geilen, and T. Basten. Distributed maintenance of minimum-cost path information in wireless sensor networks. In *Proc. of ACM PM2HW2N'11*, pages 25–32. 8, 60

[125] W.K. Sze and W.C. Lau. Rfid counting over unreliable radio channels-the capture-recapture approach. In *Proc. of IEEE ICC'10*, pages 1–6. 2, 36

[126] J. Tang, C. Mascolo, M Musolesi, and V Latora. Exploiting temporal complex network metrics in mobile malware containment. *CoRR*, abs/1012.0726, 2010. 78

[127] J. Tang, M. Musolesi, C. Mascolo, and V. Latora. Temporal distance metrics for social network analysis. In *Proc. of ACM WOSN'09*, pages 31–36. 8, 59, 60, 63, 68, 78

[128] J. Tang, S. Scellato, M. Musolesi, C. Mascolo, and V. Latora. Small-world behavior in time-varying graphs. *Physical Review E*, 81(5):055101, 2010. 8, 59, 60, 78

[129] L. Tang, Y. Sun, O. Gurewitz, and D. B. Johnson. Em-mac: a dynamic multichannel energy-efficient mac protocol for wireless sensor networks. In *Proc. of ACM MobiHoc '11*, pages 23:1–23:11. 106

[130] S. Tang, J. Yuan, X. Mao, X.Y. Li, W. Chen, and G. Dai. Relationship classification in large scale online social networks and its impact on information propagation. In *Proc. of IEEE INFOCOM'11*, pages 2291–2299. 78

[131] F. Tobagi and L. Kleinrock. Packet switching in radio channels: part ii–the hidden terminal problem in carrier sense multiple-access and the busy-tone solution. *IEEE Transactions on Communications*, 23(12):1417–1433, 1975. 18

[132] Y.C. Tseng, C.S. Hsu, and T.Y. Hsieh. Power-saving protocols for IEEE 802.11-based multi-hop ad hoc networks. *Computer Networks*, 43(3):317–337, 2003. 8, 35

[133] M. van Eenennaam, W.K. Wolterink, G. Karagiannis, and G. Heijenk. Exploring the solution space of beaconing in vanets. In *Proc. of IEEE VNC'09*, pages 1–8. 36

[134] L.F.W. van Hoesel and PJM Havinga. A lightweight medium access protocol (lmac) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches. 2004. 14, 18, 31

[135] S. Vasudevan, D. Towsley, D. Goeckel, and R. Khalili. Neighbor discovery in wireless networks and the coupon collector's problem. In *Proc. of ACM MobiCom'09*, pages 181–192. 34, 35, 36, 44, 45

[136] T. Voigt, F. Osterlind, and A. Dunkels. Improving sensor network robustness with multi-channel convergecast. In *2nd ERCIM Workshop on e-Mobility*, pages 15–24, 2008. 107

[137] M. Vojnovic and A. Proutiere. Hop limited flooding over dynamic networks. In *Proc. of IEEE INFOCOM'11*, pages 685–693. 3

[138] X. Wang, X. Wang, X. Fu, G. Xing, and N. Jha. Flow-based real-time communication in multi-channel wireless sensor networks. *Wireless Sensor Networks*, pages 33–52, 2009. 106

[139] T. Watteyne, A. Mehta, and K. Pister. Reliability through frequency diversity: why channel hopping makes sense. In *Proc. of ACM PE-WASUN'09*, pages 116–123. 106

[140] L. Weiss Ferreira Chaves, E. Buchmann, and K. Böhm. Tagmark: Reliable estimations of rfid tags for business processes. In *Proc. of ACM SIGKDD'08*, pages 999–1007. 4, 36

[141] M. Welsh, S. Moulton, T. Fulford-Jones, and D.J. Malan. Codeblue: An ad hoc sensor network infrastructure for emergency medical care. In *Proc. of BSN'04*. 2

[142] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing*, pages 18–25, 2006. 2, 5

[143] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal. Firefly-inspired sensor network synchronicity with realistic radio effects. In *Proc. of ACM SenSys'05*, pages 142–153. 5

[144] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler. Exploiting the capture effect for collision detection and recovery. In *Proc. of IEEE EmNetS-II'05*, pages 45–52. 53

[145] A. Wood, G. Virone, T. Doan, Q. Cao, L. Selavo, Y. Wu, L. Fang, Z. He, S. Lin, and J. Stankovic. Alarm-net: Wireless sensor networks for assisted-living and residential monitoring. *University of Virginia Computer Science Department Technical Report*, 2006. 2

[146] Y. Wu, M. Keally, G. Zhou, and W. Mao. Traffic-aware channel assignment in wireless sensor networks. *Wireless Algorithms, Systems, and Applications*, pages 479–488, 2009. 106

[147] R. Zhang, Y. Liu, Y. Zhang, and J. Sun. Fast identification of the missing tags in a large RFID system. In *Proc. of IEEE SECON'11*, pages 278–286. 2

[148] X. Zhang, J. Kurose, B.N. Levine, D. Towsley, and H. Zhang. Study of a bus-based disruption-tolerant network: mobility modeling and impact on routing. In *Proc. of ACM MobiCom'07*, pages 195–206. 77

[149] X. Zhang, G. Neglia, J. Kurose, and D. Towsley. Performance modeling of epidemic routing. *Computer Networks*, 51(10):2867–2891, 2007. 4

[150] R. Zheng, J.C. Hou, and L. Sha. Asynchronous wakeup for ad hoc networks. In *Proc. of ACM MobiHoc'03*, pages 35–45. 35

[151] G. Zussman, A. Brzezinski, and E. Modiano. Multihop local pooling for distributed throughput maximization in wireless networks. In *Proc. of IEEE INFOCOM'08*, pages 1139–1147. 7, 27, 31

[152] G. Zyba, G.M. Voelker, S. Ioannidis, and C. Diot. Dissemination in opportunistic mobile ad-hoc networks: The power of the crowd. In *Proc. of IEEE INFOCOM'11*, pages 1179–1187. 78

Summary

Adaptability in Dynamic Wireless Networks

Software for networked embedded systems faces several challenges when the deployed network is subject to changing circumstances during operation. Typically, inter-node communication and network connectivity are two crucial aspects that are directly affected by factors such as intermittent wireless links and node mobility. In this thesis, we investigated the possibility of using decentralized algorithms for run-time adaptability in dynamic wireless networks. To do so, we addressed research problems that touch upon concerns specific to communication and network connectivity in static networks with time-variant links, as well as dynamic (mobile) networks. We estimated quantities such as neighborhood density and temporal connectivity at run-time, based upon observations collected either locally or over the network. Additionally, we reconfigured communication parameters with the help of collaborative processing between neighboring nodes, wherever possible.

We started our research by investigating the benefits of desynchronizing node communication in a dynamic wireless network. Our findings indicated that desynchronization is best suited only for static and sparse networks. In dense and mobile network topologies, a plain randomized communication scheme achieves the same performance as desynchronization, and also precludes any overhead on collaborative processing. Notably, the parameter for randomized beaconing needs to be adapted with help of neighborhood context, in order to maximize communication efficiency. However, in dynamic wireless networks, the neighborhood of a node changes with time. Therefore, estimating the number of neighbors and acquiring their identities are non-trivial problems in mobile networks. We addressed these concerns by presenting *Nest*, an adaptive decentralized algorithm that estimates neighborhood density in both static and mobile networks, and therefore indirectly, accelerates the process of gathering neighborhood information. *Nest* combines distributed consensus with a statistical estimation of neighborhood size, and uses the estimate to quickly acquire neighborhood information. Our simulations and experimental evaluations showed that *Nest* estimates neighborhood sizes with good accuracy, and also tracks changes in node density in real-time.

The next issue we addressed was that of estimating connectivity in mobile networks, which is useful from the point of understanding the delay in disseminating information. Since nodes in a mobile network are most often not connected at any given time, traditional hop-count metrics for connectivity do not directly apply as solutions. Therefore, recent works have proposed so-called temporal distance metrics for quantifying how long it takes for a message to disseminate through a mobile network. However, these approaches characterize temporal connectivity with

the presumption of global knowledge, and involve centralized computation. We extended these research efforts by designing *PathDetect*, an adaptive decentralized algorithm that estimates the temporal connectivity locally on every node. *PathDetect* combines local observations with a distributed consensus to build a global, coherent view on temporal connectivity. Evaluation results show that *PathDetect* can not only estimate the temporal connectivity of a mobile network accurately, but also track connectivity changes in real-time.

To conclude our research, we addressed the design considerations and challenges of implementing decentralized algorithms for adaptability in real-world settings. We focused on the specific problem of external interference in low-power wireless sensor networks (WSNs). Sensor nodes that operate in the 2.4 GHz ISM band need to compete with other devices that include bluetooth peripherals, WiFi access points, and laptops. Since it is important for data gathering sensor networks to guarantee a seamless delivery of information to the sink, adaptive mechanisms need to be designed to counter the adverse effects of external interference. As a practical solution, we designed and developed *Chrysso*, a multichannel protocol wherein sensor nodes switch their frequency channels when affected by an external interferer. *Chrysso* comprises explicit channel switching policies for nodes, based on both collaborative and autonomous decision making. We evaluated *Chrysso* under a variety of interference cases, and compared its performance against *MuChMAC*, a state-of-the-art channel hopping protocol. Detailed experiments on different testbeds, and over long time spans showed that *Chrysso* mitigates the effects of external interference, and sustains data reliability with minimal energy consumption.

As an overall contribution, the research efforts in this thesis revealed common observations concerning adaptability in dynamic networks. Specifically, decentralized adaptability in communication for static networks can be realized through explicit control signalling, and through use of a tree structure. However, when the topology changes, adaptability and estimation can better be achieved through exchanging local observations via some form of distributed consensus. This offers not only a neighborhood or network-wide observation of relevant properties, but also seamlessly accommodates changing factors such as link quality and node mobility.

Samenvatting

Aanpassingsvermogen in Dynamische Draadloze Netwerken

Software voor networked embedded systemen staat voor diverse uitdagingen als het systeem onderhevig is aan veranderende omstandigheden tijdens het gebruik. Inter-node communicatie en netwerk connectiviteit zijn twee cruciale aspecten die direct worden beïnvloed door zaken als fluctuerende draadloze verbindingen en node mobiliteit. In dit proefschrift onderzochten we de mogelijkheid van het gebruik van gedecentraliseerde algoritmes voor run time aanpasbaarheid in dynamische draadloze netwerken. Om dit te bewerkstelligen, hebben we ons gericht op onderzoeksproblemen die raakvlakken hebben met zaken die specifiek betrekking hebben op communicatie en connectiviteit in statische netwerken met temporele verbindingen, alsook dynamische (mobiele) netwerken. We hebben grootheden als omgevingsdichtheid en temporele netwerkconnectiviteit geschat tijdens run time, op basis van observaties die lokaal of door het netwerk zijn verzameld. Daarnaast herconfigureerden we, wanneer mogelijk, de communicatieparameters met behulp van onderlinge samenwerking van buurnodes.

We begonnen ons onderzoek met het bekijken van de voordelen van gedesynchroniseerde communicatie tussen nodes in een dynamisch draadloos sensor netwerk. Onze bevindingen gaven aan dat desynchronisatie alleen geschikt is voor statische en dunbezaaide netwerken. In een dikbezaaid en mobiel netwerk presteert een eenvoudig willekeurig communicatieschema even goed als desynchronisatie, zonder de overhead van onderlinge samenwerking. Met name de parameter voor randomized beaconing moet worden aangepast met behulp van de buurnodes om zo de communicatie-efficiëntie te maximaliseren. In een dynamisch netwerk daarentegen, veranderen de buurnodes continu. Het schatten van het aantal buurnodes en het verkrijgen van hun identiteit is een niet triviaal probleem in mobiele netwerken. We hebben dit probleem aangepakt met het presenteren van Nest, een adaptief gedecentraliseerd algoritme, dat de grootte van de omgevingsdichtheid schat in statische en mobiele netwerken, alsook zo snel mogelijk de identiteit van buurnodes verzamelt. Nest combineert gedistribueerde consensus met een statistische schatting van de buurtgrootte en gebruikt deze schatting om snel informatie van de buurnodes te verzamelen. Onze simulaties en experimenten laten zien dat Nest de buurgrootte met een goede nauwkeurigheid alsook in real time veranderingen bijhoudt.

Het volgende vraagstuk was het schatten van connectiviteit in mobiele netwerken, wat belangrijk is voor het verkrijgen van inzichten over vertragingen bij het verspreiden van informatie. Omdat nodes in mobiele netwerken vaak niet op elk moment met elkaar zijn verbonden, zijn traditionele hop-counting metrics voor connectiviteit niet direct bruikbaar. Om deze reden heeft recent onderzoek de zogenaamd “temporal distance metric” voorgesteld, voor het kwantificeren

van de tijdsduur voor het verspreiden van een bericht door het netwerk. Echter, deze methode maakt gebruik van globale kennis, en gecentraliseerde berekeningen. We hebben dit onderzoek uitgebreid met PathDetect, een adaptief gedecentraliseerd algoritme, wat de temporele connectiviteit lokaal op elke node schat. PathDetect combineert de lokale observaties met een gedistribueerde consensus om tot een globaal en coherent overzicht op temporele connectiviteit te komen. Evaluaties laten zien dat PathDetect niet alleen de temporele connectiviteit in een mobiel netwerk goed schat, maar ook in real time de veranderingen in connectiviteit bijhoudt.

Om ons onderzoek af te sluiten, hebben we ons op de ontwerp afwegingen en uitdagingen bij het implementeren van een gedecentraliseerd algoritme voor aanpassingsvermogen in een real-world setting gericht. We hebben ons hierbij in het bijzonder gericht op het probleem van externe interferentie in laag-vermogen draadloze (Wireless) Sensor Networks (WSNs). Sensor nodes die opereren in de 2,4 GHz ISM band moeten concurreren met andere apparaten, zoals Bluetooth accessoires, WiFi access points en laptops. Omdat het belangrijk is voor sensor netwerken om informatie ononderbroken af te leveren bij de sink, zijn dynamische mechanismen nodig voor het tegengaan van nadelige effecten van externe interferentie. Chrysso bestaat uit een expliciet kanaalverspringingsbeleid voor nodes, op basis van zowel collaboratieve als autonome beslissingen. We evalueerden Chrysso met een verscheidenheid aan interferentie scenario's en vergeleken de prestaties met MuChMAC, een state-of-the-art kanaalverspringingsprotocol. Uitvoerige experimenten op diverse testbeds en over een lange tijdsperiode, lieten zien dat Chrysso de effecten van externe interferentie vermindert en een betrouwbare datastroom behoudt met een minimaal energieverbruik.

Als algehele bijdrage, liet dit proefschrift gemeenschappelijke observaties zien met betrekking tot aanpasbaarheid in dynamische netwerken. En meer specifiek, gedecentraliseerde aanpasbaarheid in communicatie voor statische netwerken kan worden gerealiseerd via expliciete controle signalen en door gebruik te maken van een boomstructuur. Daarentegen, als de topologie verandert, kunnen aanpasbaarheid en het schatten van metrics eenvoudig worden bereikt door middel van het uitwisselen van lokale observaties, via een vorm van gedistribueerde consensus. Dit biedt niet alleen een omgevings- of netwerk-brede observatie van de relevante eigenschappen, maar past zich ook naadloos aan aan veranderende factoren als link kwaliteit en node mobiliteit.

Biography

Venkatraman Iyer was born in Chennai, India, on June 6, 1981. He spent his childhood and early professional life in Mumbai, India. He received his Bachelor's degree in Computer Engineering from the University of Mumbai in 2003. In 2006, he decided to pursue a Masters degree in Computer Science at the Delft University of Technology, the Netherlands. Following his graduation in 2008, he decided to continue as a PhD candidate in the Embedded Software Group at Delft University of Technology.
Venkatraman, or *Venkat* as he is called, intends to pursue his career as a researcher.

Employment

PhD Researcher at Delft University of Technology The Netherlands	2008-2012
Programmer Analyst at Syntel India Ltd. Mumbai, India	2003-2006

Publications

1. V.G. Iyer, A. Pruteanu, and S.O. Dulman. Netdetect: Neighborhood discovery in wireless networks using adaptive beacons. In *Proc. of IEEE SASO'11*, pages 31-40.

2. V.G. Iyer, M. Woehrle, and K.G. Langendoen. Chrysso - a multi-channel approach to mitigate external interference. In *Proc. of IEEE SECON'11*, pages 422-430.

3. A. Pruteanu, V.G. Iyer and S.O. Dulman. FailDetect: Gossip-based Failure Estimator for Large-Scale Dynamic Networks. In *Proc. of IEEE ICCCN'11*, pages 1-6.

4. A. Pruteanu, V.G. Iyer and S.O. Dulman. ChurnDetect: A Gossip-based Churn Estimator for Large-Scale Dynamic Networks, In *Proc. of Springer Euro-Par'11*, pages 289-301.

5. V.G. Iyer, M. Woehrle, and K.G. Langendoen. Chamaeleon - exploiting multiple channels to mitigate interference. In *Proc. of IEEE INSS'10*, pages 65-68.

Teaching Experience

Teaching Assistant at *Delft University of Technology*, The Netherlands

Course	*Responsibility*
Compiler Construction	Lab Assistance, Grading
Wireless Sensor Networks seminar	Grading