Dissertation

Time-Slotted Schedule-based Channel Sensing in 802.11 Ad-hoc Networks

Thesis approved by the Department of Computer Science
of the University of Kaiserslautern (TU Kaiserslautern)
for the award of the Doctoral Degree

Doctor of Engineering (Dr. -Ing.)

to

M.Sc. Paulo Fernando Aragao Alves Junior

Date of Defense:	September 26, 2022
Dean:	Prof. Dr.-Ing. Jens Schmitt
PhD Committee	
Chair:	Prof. Dr.-Ing. Didier Stricker
Reviewers:	Prof. Dr.-Ing. Reinhard Gotzhein
	Prof. Dr. Klaus Schneider

D 386

Abstract

The wireless spectrum is already a scarce good, shared by multiple competing technologies such as Bluetooth, ZigBee and Wi-Fi, and the hunger for traffic is only increasing. Due to the heterogeneity of the existing wireless technologies and the real threat that interference poses to network performance, sophisticated techniques must be developed to ensure acceptable levels of quality of service.

In this thesis, we present a *passive* channel sensing scheme based on both energy and signal detection, that primarily considers the spectrum occupation of foreign traffic while allowing for additional complementary information such as the signal-to-noise ratio. The resulting channel quality metric is first corrected for the spectrum occupation of internal transmissions and later aggregated with help of a moving average followed by an exponential weighted moving average. This aggregation keeps the metric both sufficiently stable and adaptive to significant changes in channel usage. Moreover, the channel quality metric is made volatility-aware by penalizing qualities proportionally to their *downward* volatility. This yields a conservative metric and allows to differentiate channels with similar aggregated qualities but different volatility behavior.

Our second main contribution is in the form of a schedule-based channel sensing protocol, in which nodes possess two network interfaces, one for communication and one for channel sensing. Channel sensing schedules are derived from communication schedules, i.e. channel hopping sequences used for communication, with help of a stochastic local search-based heuristic that attempts to minimize channel sensing bias, the channel overlap between both schedules and to maximize overlap fairness. This minimizes the effect of internal transmissions in the resulting channel quality metric, allowing nodes to derive channel quality primarily based on foreign traffic in an unbiased manner.

Finally, we propose and implement a stabilization protocol for keeping nodes in an ad-hoc network tick-synchronized and schedule-consistent w.r.t. a communication schedule. This stabilization protocol makes use of special messages, namely tick frames for synchronization, channel quality reports for sharing local views of channel conditions and schedule reports for disseminating the global communication hopping sequence. The communication schedules are computed by a master node based on an aggregation of local channel quality views and the re-computation of these schedules is triggered by significant changes in channel conditions. The resulting protocol is robust against changes in topology and channel conditions.

Acknowledgements

I would indeed require far more space than this single page to properly express my gratitude and acknowledge the depth of the generosity of the people who have in myriad ways positively influenced my journey to write this thesis.

First of all, I would to like to thank my PhD supervisor, Prof. Dr. Reinhard Gotzhein for his ever-present guidance, his many helpful and insightful suggestions as well as his laser focus attention to details, which not only inspired me, but I hope helped me become a better researcher and professional. I also would like to thank Prof. Dr. Klaus Schneider, a person I have always admired for his intellect and teaching skills, for accepting to review this thesis. In addition, I want to thank Prof. Dr. Didier Stricker for acting as chair of the PhD Committee and Dr. habil. Bernd Schürmann for making it possible for me to defend my PhD thesis remotely.

I would like to express my gratitude and appreciation to each of the current and former AGVS members that I have worked with, for the ever-enriching scientific discussions, but also for helping our research group feel like a second home. In special, I would like to thank Dr. Markus Engel for creating the seeds for this thesis. It started way back then, 7 years ago, when as a HiWi I got to help out with his PhD work. Time flies. I also would like to thank Ricardo Sabedra and Lucas Hagen for their help with experimental work and with the implementation and validation of some of the tools developed during this thesis.

Finally, I'd like to thank my whole family and friends, in special my mother whose emotional and financial support was essential to give me the courage to leave my home country and complete my studies in Germany, my in-laws who for years have been my second family and last but not least, I thank my beloved wife, Sophie, for her invaluable tips regarding all things visual, her encouragement and unwavering support during the conception of this thesis.

Every thing grand is made from a series of ugly little moments. Everything worthwhile by hours of self-doubt and days of drudgery. All the works by people you and I admire sit atop a foundation of failures. So whatever your project, whatever your struggle, whatever your dream, keep toiling. Per aspera, ad astra!

— Pierce Brown, Morning Star

Contents

List of Tables

List of Figures

Traffic increases to exceed the available spectrum.

— Adrian Stephens, technical editor for 802.11n

1 Introduction

Contents

1.1 Context

Wireless communication is everywhere. The need for ubiquitous connection has made IEEE 802.11, also known as Wi-Fi, among other prevailing wireless protocols, a fundamental component of modern society. The estimated global economic value of Wi-Fi in 2021, considering different aspects such as the needs of consumers and industry as well as the economic impact of the global SARS-CoV-2 pandemic, came in at 3.3 trillion dollars, with a predicted value of 4.9 trillion dollars for 2025 [RKC21]. One study has pointed out that in Germany alone the time smartphone users spent using Wi-Fi networks instead of cellular communication technology (e.g. 4G) went from 64.7% to 71.4% during the outbreak of the COVID-19 pandemic [RKC21].

The applications of wireless communication technology are multifaceted and range from consumer electronics and entertainment to sensor networks in industrial settings. The number of connected devices increases dramatically every year and with it an insatiable hunger for bandwidth and speed. In most countries, Wi-Fi mainly uses two frequency bands, 2.4 GHz and 5 GHz, originally allocated for Industrial, Scientific and Medical (ISM) applications. These ISM bands face not only competing use cases, but also competing technologies. In addition to 802.11, the 2.4 GHz band has been overcrowded for a long time by different wireless protocols, such as Bluetooth, ZigBee as well as other technologies that emit radiation in the same frequency band, such as microwave ovens. On the other hand, even though the 5 GHz band still has a lower average spectrum occupation than the 2.4 GHz, this situation is changing rapidly. This is mostly due to an increasing number of Wi-Fi transceivers that support 5 GHz as well as the approval and adoption of LTE in unlicensed spectrum (LTE-U), which intends to improve coverage in cellular networks by using the 5 GHz band alongside typical cellular carrier frequencies. In a world with more than 5 billion smartphone users [Sta22], LTE-U has the potential to provide immense competition for 802.11. Furthermore, spectrum availability is also one of the main barriers to a broader scale Internet of Things (IoT) deployment. IoT is a booming business with a world market value estimated at US$ 1.7 trillion for 2021 [BRA20].

1.2 Problem statement

Everyone wants a piece of the spectrum pie. It is hence no surprise that the current scenario has led to an increasing congestion of the available unlicensed radio spectrum. Numerous problems are caused by this congestion. First, networks in mutual communication range and using distinct technologies can interfere with each other by using overlapping channels, but cannot decode foreign frames, which makes coexistence harder. Second, different medium access control (MAC) schemes might lead to an unfair distribution of the available bandwidth between competing networks. Both factors can result in substantial throughput reduction, in special for the network using the most polite protocol. In the worst case, close proximity and incompatible MAC protocols might lead to no throughput at all for all contending networks.

Communicating effectively in such a congested scenario demands sophisticated approaches. In particular, this need is made most evident in mobile ad-hoc networks, such as vehicular ad-hoc networks (VANET), in which performance and security are paramount. Even though an infrastructure-based mode (managed mode) is the most commonly used mode of operation for 802.11 devices, there are some disadvantages that come with it. Some of the main disadvantages are limited scalability, restricted robustness regarding node mobility, and the associated costs of installing the needed infrastructure may be neither desirable nor justified, e.g. in disaster networks or in ceratin residential community networks [ACG04]. In such cases, an ad-hoc mode of operation or a combination of infrastructureless and infrastructure-based operations seems to be the most efficient solution.

As of 2022, most IEEE 802.11-based ad hoc networks are still being developed in academic environments. Nonetheless, IEEE 802.11 ad-hoc-based systems have a great potential economic value. One promising application of the technology is helping improve the connectivity and quality of service in public access networks. Such networks are already widespread in both developed and developing countries. Brazil, for instance, has seen an aggressive development in public access Wi-Fi networks in its municipalities. Current estimates state that nearly 38% of all municipalities in the country offer free Wi-Fi services [BRA20]. This is a clear reflection of the economical situation of the country, where a large chunk of its population does not have the economic resources to subscribe to broadband Internet services, relying instead on free municipal access. Cisco reports that Brazil has over about nine million public Wi-Fi access points (APs) as of 2021, with a projection of 23.8 million by 2023 [RKC21]. And Brazil is not alone in this

development. The number of open public Wi-Fi APs has been increasing everywhere in the world and is already very high in countries such as France (1.2 million), Poland (3.1 million), India (3.7 million) and Mexico (7 million) [RKC21].

Brazilian wireless ISPs, for instance, report that they already observe a bottleneck at the capacity to serve multiple clients, only being able to handle 50 subscribers per channel at a time [ACG04]. While each new Wi-Fi standard has brought increasing transmission rates, transmission power was kept bounded by regulations set up by regulatory bodies such as ANATEL for Brazil, ETSI for the European Union and the FCC for the United States. Increasing transmission rates while keeping the transmission power constant effectively lowers transmission range and with it coverage area. A rate of 100 Mbps, for instance, leads to a coverage area of only a few meters around a given access point [ACG04]. Since associated costs and space constraints limit the number of access points that can be placed near each other, a more scalable solution would be to keep the number of APs low and have these access points act as gateways to the Internet for client nodes that connect to each other in a multihop ad-hoc network [ACG04].

1.3 Our approach

To deal with this spectrum overcrowding and ensure appropriate quality of service (QoS) levels, we have following tools at hand: channel sensing and channel hopping. In order to assess the state of a channel, we need to gather channel quality information, allowing nodes to select high-quality channels for effective communication. Channel selection can then bootstrap a proactive channel hopping scheme, in which nodes hop to different channels and use channels with a frequency proportional to their quality.

1.3.1 Channel sensing

The quality of a channel reflects the expectation on performance to be obtained when using it for communication. Channel quality is thus a promise of better performance in the medium (lower latency, less frame loss, higher throughput). If this channel quality falls below acceptable levels, we might need to use more local resources on a node e.g. more energy and CPU time due to re-transmissions, as well as global resources such as channel airtime. In IEEE 802.11 networks, airtime is a precious asset, since nodes that wish to transmit data, but are within communication range of transmitting nodes, have to back-off and wait for a certain time before attempting to transmit again.

Using a bad channel is in this way not only detrimental for a single node that has to resend some frames, but also for the whole set of nodes in its communication range that have to contend with it for channel access. However, ranking channels according to channel quality makes it possible not only to give preference to high-quality channels but also to blacklist channels whose conditions do not hold up to minimal communication requirements.

In this thesis, we concern ourselves with *passive* channel sensing, in contrast to active link quality measurements, such as estimating packet delivery ratio based on acknowledgments. Passive means that the sensing takes place without any energy emission.

To perform sensible channel sensing, the first questions to be answered are: which channel property or properties should be used to gauge the quality of a channel and how should raw collected data be interpreted and aggregated into a single channel quality metric for nodes to act upon.

With regards to gathering channel statistics, one possibility would be overhearing all frames on the medium. This allows a node to measure the quality of a channel based on the presence of other nodes using the same wireless protocol. However, this limits the assessment of the channel to networks in communication range using the same communication technology, and to the set of received signals that can be decoded as a frame. Since frame decoding alone is insufficient for a proper channel assessment, a more adaptable approach should use energy detection, either in isolation or combined with frame overhearing. Measuring energy levels on the medium enables a node in sensing range of transmitting nodes not only to label a channel as either idle or busy but also to quantify the busyness of this channel.

After collecting channel quality raw data, a node needs to aggregate these data in order to obtain a single channel quality metric. An ideal metric should be accurate, display acceptable levels of stability while staying adaptive to significant changes in channel conditions and should be volatility-aware, i.e. yield conservative channel quality estimates based on downward fluctuations. Furthermore, as different nodes observe different channel quality conditions, nodes should exchange measured channel qualities to share their local views.

1.3.2 Channel hopping

In this work, we have following motivating scenario: nodes operate in an ad-hoc wireless network and exchange data on a common channel for a given dwell time, also called a time slot, after which they hop to another pre-defined channel (see Fig. 1.1). This

proactive behavior of vacating a channel and switching to another is called throughout the literature *frequency hopping* or *channel hopping*, the term preferred in this work. A channel hopping sequence, i.e. the sequence of channels to switch to, is also called a *channel hopping schedule*. In further chapters, we will introduce two different types of

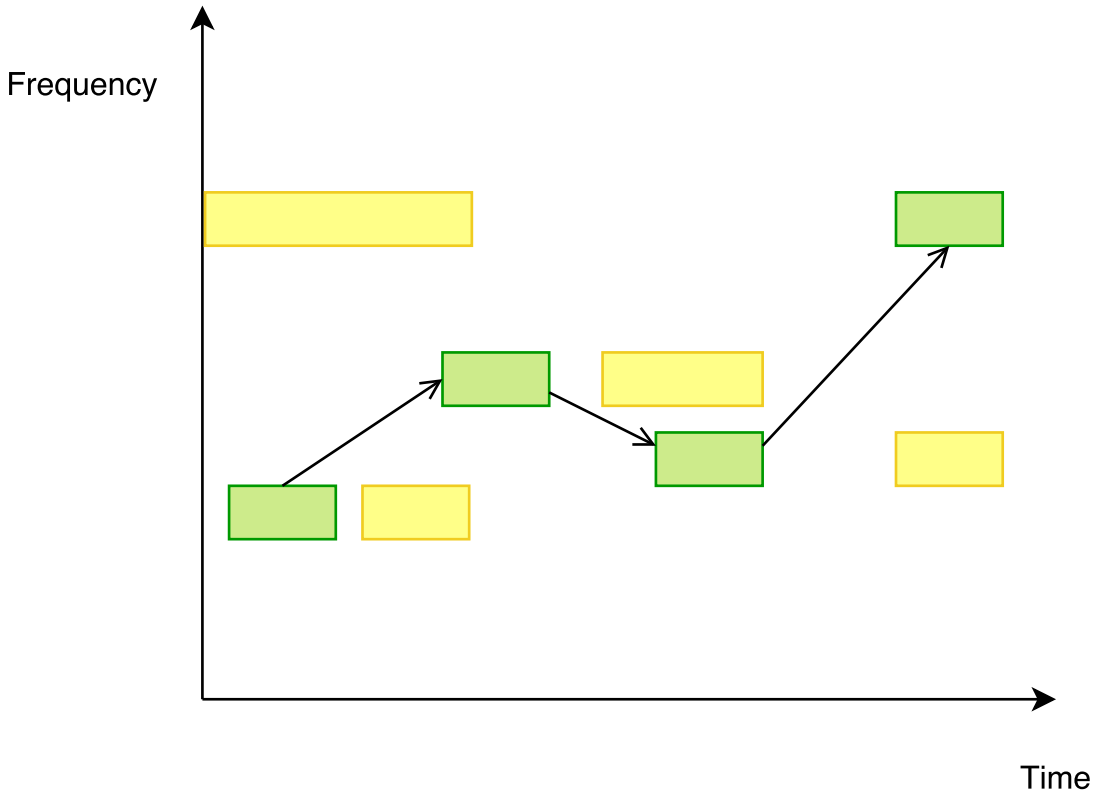


Figure 1.1: Illustration of a channel hopping process where nodes change their operating frequencies according to a given hopping sequence after a fixed dwell time.

schedules, namely a sensing and a communication schedule. For now, it suffices to say that nodes switch channels to communicate on, following a communication schedule and that they use an additional transceiver to simultaneously perform channel sensing on all available channels in an order defined by a sensing schedule. However, in this thesis we do not handle in detail how communication schedules are computed. For a complete study on communication schedules, please see [Eng20]. For the sake of simplicity, we might refer to a channel hopping schedule simply as a schedule unless proper disambiguation is needed.

While channel quality metrics define *what* to measure and *how*, sensing schedules define *when* to measure it. As already mentioned, time in our network is divided into

time slots. Slots have a fixed duration d_{slot} and each slot defines the duration of usage (or survey) of a single channel.

Regarding the relation between the duration of a slot and the symbol duration, there are two main types of channel hopping systems: fast frequency hopping (FFH) and slow frequency hopping (SFH). A symbol is a pulse or waveform that persists on a channel for a fixed duration. It is the smallest physical resource unit of communication and may encode several bits. 802.11n, for example, has symbol durations of either 3.6 or 4 μs, of which 3.2 μs are the symbol time (data transmission), and the remaining 0.4 or 0.8 μs are respectively either the short or the long guard intervals. In FFH systems, one symbol is transmitted over multiple time slots, i.e. d_{slot} is smaller than the symbol duration. In this thesis, we deal with SFH networks, where the slot duration is larger than the data symbol duration. In fact, typical slot duration for our scenario is at least an order of magnitude larger than symbol duration, with $d_{slot} \geq 10$ ms.

1.4 Contributions and outline

The main contributions of this thesis are a schedule-based passive channel sensing scheme and an active master-based stabilization protocol for achieving schedule consistency w.r.t. a communication schedule within a multihop ad-hoc network. These main contributions are presented in Chapters 2, 3 and 4. The thesis is then structured as follows:

- In Chap. 2, we propose a methodology for passive channel sensing and describe the needed components for its implementation in commodity 802.11 hardware. This channel sensing scheme is based on both energy and signal detection, and primarily takes into consideration the spectrum occupation created by foreign nodes. The resulting channel quality metric, q_{cbt} is first corrected to account for the spectrum occupation generated by internal transmissions and is later aggregated to make it both sufficiently stable and adaptive to significant changes in channel usage. This aggregation is implemented as a combination of a moving average with an exponential weighted moving average. Furthermore, the channel quality metric is made volatility-aware by penalizing channel qualities in proportion to their *downward* volatility levels. This makes the resulting metric more conservative and enables us to distinguish channels that display similar channel qualities but present distinct volatility behaviors. Moreover, additional channel quality information such as the foreign traffic aware signal-to-noise ratio can be combined with the primary

channel quality information to provide a multidimensional view of the channel state.

- In Chap. 3, we propose and experimentally assess a method for the construction of high-quality local sensing schedules in which nodes possess two network interfaces, one for communication and one for channel sensing. Nodes determine the channel sensing order, i.e. when each channel should be sensed, through *sensing* schedules, which are derived from a *communication* schedule. High-quality sensing schedules are constructed with help of a stochastic local-search-based heuristic that attempts to minimize the channel overlap effects between both schedules while minimizing channel sensing bias.

- In Chap. 4, we propose a master-based stabilization protocol for keeping nodes in an ad-hoc network tick-synchronized and schedule-consistent w.r.t. a communication schedule. This stabilization protocol makes use of special messages, namely tick frames for tick synchronization, channel quality reports for nodes to share local channel quality views and schedule reports for disseminating the current global communication hopping sequence, computed by the master node. Communication schedules are computed based on aggregated channel qualities and their dissemination is triggered by significant changes in channel quality.

- In Chap. 5, we describe the main tools developed during our research to help us conduct controlled experiments and to evaluate the proposed channel quality metrics. These tools are a wireless traffic generator, two types of data visualization and a data collection pipeline using a time-series database for storing channel sensing data and a document-based database for managing experiment metadata. In addition, we briefly describe the architecture used for deploying these tools to our testbed in a flexible and scalable manner.

- In Chap. 6, we summarize our results and present some open challenges for possible future work.

Even though many of the techniques introduced in this thesis are general enough and could be easily adapted for other wireless protocols, we have focused on applying them to IEEE 802.11-based ad-hoc networks. Moreover, we have implemented all proposed approaches and either deployed them to a testbed with 802.11 transceivers or to a simulation framework for experimental assessment and validation.

It is quality rather than quantity that matters.

— Seneca

2 Channel quality

Contents

In this chapter, we introduce a sophisticated channel quality metric based on energy detection, describe its implementation on commodity 802.11 hardware and assess its performance through real experiments in a testbed. First, we define our raw channel quality q_{cbt}, which is collected by taking advantage of available customary hardware mechanisms for clear-channel assessment (CCA). Second, we gauge the suitability of q_{cbt} to measure the quality of a channel. Third, we improve the accuracy of the delivered raw data by applying corrections w.r.t. the effects of internal traffic due to channel overlaps and the near-far effect. Fourth, we aggregate the corrected channel quality values to make the metric more stable and apply an exponential weighted moving average (EWMA) to provide additional adaptivity. Fifth, we define and apply several candidate metrics to measure and account for the *downward* volatility of raw channel quality, yielding volatility-aware channel quality metrics. The resulting metrics are not only accurate, adaptive and stable, but also more conservative without being overly pessimistic. Each of these steps is implemented and presented along with experimental results.

2.1 Basic metric q_{cbt}

IEEE 802.11 is a group of listen-before-talk (LBT) protocols that rely on energy detection and frame decoding to perform clear-channel assessment (CCA). Nodes are only allowed to transmit on a channel that is deemed idle, otherwise if the channel is sensed busy, i.e. the sensed energy lies above a chosen energy threshold, the transceiver defers the transmission according to 802.11 protocol rules.

A channel quality metric may incorporate different channel properties derived by either energy or signal detection, such as node degree, average received signal strength, spectrum occupation, etc. This channel quality value is measured passively and should reflect the overall channel conditions and correlate with the current throughput and quality of service experienced by sensing nodes on this channel. Hence, higher quality channels should yield less interference and less collisions.

Definition 2.1.1. Channel quality is a numerical value between 0 and 1 associated with a wireless channel that reflects its current state w.r.t. to QoS parameters and its overall usage by external nodes. 0 denotes the worst possible channel state and 1 denotes a perfect channel, according to the chosen criteria.

The ratio of the time period during which the sensed channel is detected as busy to the period during which CCA is performed is usually referred to in the literature

as the channel busy ratio (CBR) or the channel busy fraction (CBF). This metric has been used for all types of applications and has been predominantly assessed and validated through simulations or experiments that require altered drivers or customized firmwares. [HXY05], for example, showed with help of simulations, that CBR is an injective function of collision probability and serves as a good estimator for the available bandwidth. [SGSK07] attempted to identify high throughput paths in 802.11 mesh networks. For this purpose, the authors developed an analytical model, in which CBF is an estimator for end-to-end maximum achievable throughput, and validated their proposed approach with simulations. In addition, the authors in [DKS10] relied both on simulations and real testbed-based experiments with an Atheros 5212 802.11a/b/g chipset and further confirmed that the channel busy fraction is an accurate predictor of the available bandwidth. Nonetheless, unlike our approach, these experiments made use of a customized driver to extract the values stored in the `PROFCNT_RXCLR` and `PROFCNT_CYCLE` registers, needed to compute CBR.

In order to assess the quality of all available channels, we proactively switch channels and survey the quality of one channel per time slot. At the end of each slot, we measure both the time during which the channel was deemed busy (d_{busy}) and the time during which the node performed clear channel assessment (d_{CCA}). Our basic channel quality metric q_{cbt} is thus defined as

$$q_{cbt} = 1 - \frac{d_{busy}}{d_{CCA}} \tag{2.1}$$

where cbt stands for channel busy time.

The proposed basic channel quality metric quantifies the idleness of the medium. Experiments in our testbed have shown that q_{cbt} has a strong correlation with the maximum achievable throughput on a channel, further confirming the results surveyed in the literature.

The main advantages of using q_{cbt} are:

- It barely introduces overhead.
- It allows for a node to detect interference of nodes outside its communication range but still in sensing range.
- It is an agnostic metric, i.e. by using both signal and energy detection, a node can detect changes in the noise floor triggered by any devices operating in the same portion of a given frequency band.

2.2 Implementation - q_{cbt}

In this section, we discuss in detail the implementation of q_{cbt} on off-the-shelf commodity 802.11 hardware.

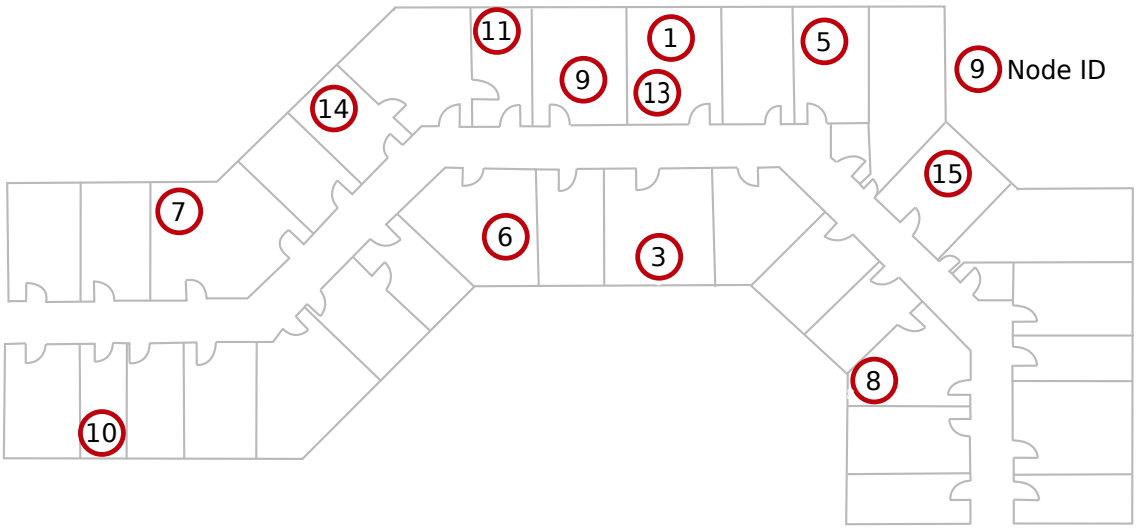


Figure 2.1: Standard node placement of our testbed. 15 nodes are placed in multiple rooms along a long corridor.

We ran our main experiments in a testbed with 15 nodes (see Fig.2.1). These nodes are equipped with two 802.11a/b/g/n off-the-shelf Compex WLE200NX mini PCIe transceivers with an Atheros AR9280 chipset and PC Engines APU2 boards (see Fig. 2.2). Both transceivers operate in monitor mode. Radio Frequency Monitor (RFMON) mode or simply monitor mode lets a device overhear all traffic that reaches a network interface controller (NIC). This operation mode is needed for proactive channel hopping on 802.11 devices and overhearing frames on a channel, including those which are not destined for the sensing node.

Even though the idea of using multiple independent network cards on each network node is still not widespread, the use of multiple antennas has already become more the

norm than the exception among 802.11 devices. This tries to capitalize on antenna diversity, i.e. antennas with at least a wavelength between them display different reception capabilities due to different reflection and fading conditions. For instance, a wavelength in the 2.4 GHz frequency band is approximately 12.5 cm. Furthermore, most modern consumer devices such as laptops already have two antennas, one for each side of the LCD screen, and USB dongles usually have multiple antennas as part of their printed circuit board (PCB) [STC+08].

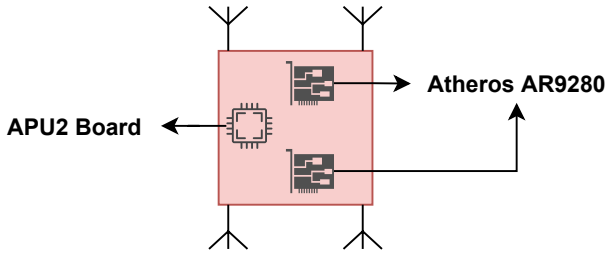


Figure 2.2: Illustration of a testbed node. Every transceiver has two antennas, and antennas of different transceivers are placed at least 24 cm apart.

The chosen chipset uses an Ath9k driver and is able to deliver spectral data from which we should be able to derive the power of any received signal. This would allow us to choose an arbitrary energy threshold to compute d_{busy} in a flexible manner. This functionality, however, is still experimental and almost undocumented. With this in mind, we opted for the Linux 802.11 Netlink userspace API. This interface of the Linux wireless stack exposes the `NL80211_CMD_GET_SURVEY` command, which provides us with both d_{busy} and d_{CCA}. Even though this binds us to the energy thresholds implemented in the chipset, it delivers consistent results through a well-established API.

Still, adopting the Netlink survey API has its own peculiarities. Before settling on our current experimental setup, we performed a series of experiments with different network cards, mostly TP-Link and Ralink USB dongles and noticed some implausibility in the delivered values, e.g. $d_{busy} > d_{CCA}$. In fact, the USB-based cards led to inconsistent measurement errors when surveying the channel with short slot durations, i.e. $d_{slot} < 40$ ms. These errors were a result of additional delays introduced by the USB interfaces. After adopting PCI network cards, we were able to consistently retrieve plausible results regardless of the chosen d_{slot}. In addition, when using the Netlink API, we had to deal with implementation inconsistencies, made evident when testing with different chipsets. One notable inconsistency is the manner different drivers handle the d_{busy} and d_{CCA} registers. While some reset the measured busy times after each query, some keep accumulating it as long as the operation channel is not changed.

802.11n supports both 20 and 40 MHz channels. The latter are composed of a combination of the former. Each 40 MHz channel has a primary and a secondary 20 MHz channel, where the primary channel is used e.g. for management data, such as beacon transmissions.

For 20 MHz channels, IEEE 802.11 defines a channel as busy either upon decoding an 802.11 signal with at least -82dBm of signal power or when energy is detected on the medium at -62dBm or stronger, regardless of the nature of the received transmission [Gas12]. In the case of 40 MHz, similar rules apply with minor variations on the energy thresholds and a slightly more complicated behavior: nodes defer transmission when either one of the 20 MHz halves is busy, and CCA applies signal detection on the primary channel, but energy detection on the secondary one [Gas12]. Even though 40 MHz channels provide a network with higher peak throughput, 20 MHz leads to better total sustained throughput. One of the main reasons for this is that the wider channels require twice as much spectrum to be idle before transmitting. On top of that, by using a larger portion of the spectrum for each frame, 40 MHz channels will lead to more inter-channel interference. Hence, in the 2.4 GHz band any node in the network can require all nodes in communication range (including access points) to abstain from using 40 MHz channels by setting the *Forty MHz Intolerance* bit. In this thesis, we concentrate on channels with 20 MHz of bandwidth and use them for all our experiments. It is to be expected though that the interference effects observed on these narrower channels should be even more pronounced in wider ones.

Moreover, in our network, nodes sense all available channels following a sensing schedule that is derived by each node locally from its current communication hopping sequence such that channel overlaps between communication and sensed channels are minimized. The importance of this is explained in Sec. 2.4.

2.3 Experimental Assessment

In this section, we assess the suitability of q_{cbt} as a channel quality metric.

Our experimental assessment has two main parts:

1. An assessment of how well q_{cbt} correlates with achieved throughput in real TCP transmissions.

2. An examination, with help of q_{cbt}, of the extent of inter-channel interference and the near-far effect, i.e. interference happening between channels with no nominal overlap due to the close proximity of nodes.

2.3.1 Traffic generator

In order to conduct reproducible and controlled experiments, we have implemented a 802.11 traffic generator with following main features:

- Adjustable frame rate.
- Payloads can have fixed length (1200 bytes) or variable length.
- The length of the variable payloads follows a normal distribution with parameters chosen from the literature to emulate wide area IP traffic patterns [MC00].

Before showing our main experimental results, we briefly discuss how to compute frame airtimes for 802.11a and 802.11b, followed by an initial assessment of baseline spectrum occupation and usual channel selection in 802.11 networks to give some further context as well as comparison parameters for the proposed techniques and shown experimental results.

2.3.2 Airtime calculation

In this section, we briefly describe 802.11b and 802.11a framing and how to perform airtime calculations for both types of frames. Both types of frames generated by our traffic generator and the airtime calculation allows us to compute the spectrum occupation produced by the traffic generator, which is fundamental for the assessment of q_{cbt}. In addition, computing frame airtime is also needed for performing corrections to q_{cbt} to be introduced in Sec. 2.4.

802.11b

A 802.11b frame has three main parts: physical layer components (preamble and PLCP header), MAC header and payload. The preamble is the first portion of the Physical Layer Convergence Protocol (PLCP) data unit and allows the receiver to synchronize with the transmitter. In general, preambles are transmitted at fixed bitrates.

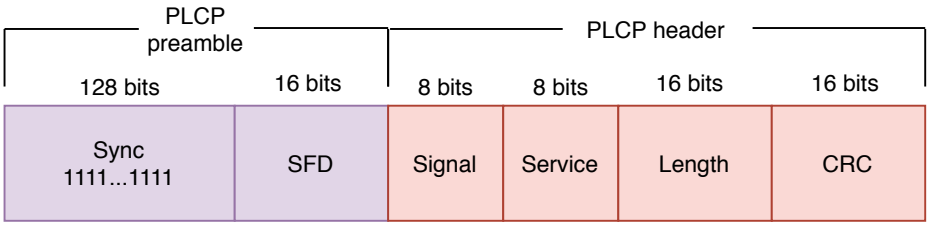


Figure 2.3: PLCP for 802.11b with long preamble (128 bits).

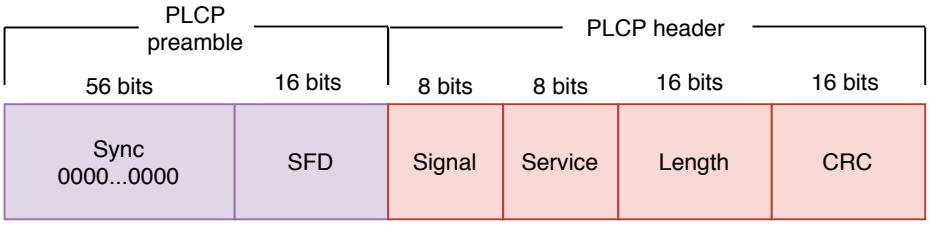


Figure 2.4: PLCP for 802.11b with short preamble (56 bits).

We have two possible preambles: a long preamble and a short preamble. As shown in Fig. 2.3 and Fig. 2.4, the long and short preamble start with a Sync field that is respectively a sequence of 1's and 0's.

The long preamble and its PLCP header are transmitted at the lowest possible data rate, which in the case of 2.4 GHz is 1 Mbps, see Tab. 2.1. In the case of the short preamble, the preamble itself is also transmitted at 1 Mbps, but the PLCP header is transmitted at 2 Mbps, see Tab. 2.2. This means that with the long preamble PLCP's airtime amounts to 192 μs and with the short preamble 96 μs. Newer standards, e.g. 802.11n and 802.11ac, use even shorter preambles, which depending on the operation mode can take from 20 to 36 microseconds of airtime.

We can see in Tab. 2.1, that given a payload of 1174 bytes, by transmitting at 1 Mbps we use 9808 microseconds of airtime. This means that with a frame rate of 100 frames per second we can occupy the channel for 98.08% of the time. For comparison purposes, we show in Tab. 2.2 the frame transmission times for a bitrate of 5.5 Mbps with a short preamble. In this case, by using the same rate of 100 frames per second we only occupy the channel for 18.44% of the time. Note that our traffic generator cannot reach 100% of airtime in any of our experiments due to multiple factors, such as DIFS between every frame (see Fig. 2.5) and the backoff time when the the channel is detected as busy. DIFS stands for Distributed Coordinated Function Interframe Space, which is the time period during which the channel has to be idle in order for a node to be able to transmit. However, a busy node can sustain high levels of spectrum occupation for quite some time,

Frame Component	Bitrate (Mbits/s)	Length (bits)	Time (μs)
PHY header: PLCP preamble	1	144	144
PHY header: PLCP header	1	48	48
MAC header (28 bytes)	1	224	224
Payload (1174 bytes)	1	9392	9392

Table 2.1: Airtime calculation of 802.11 broadcast frame on 2.4 GHz with payload of 1174 bytes and long preamble with total airtime: 9808 μs.

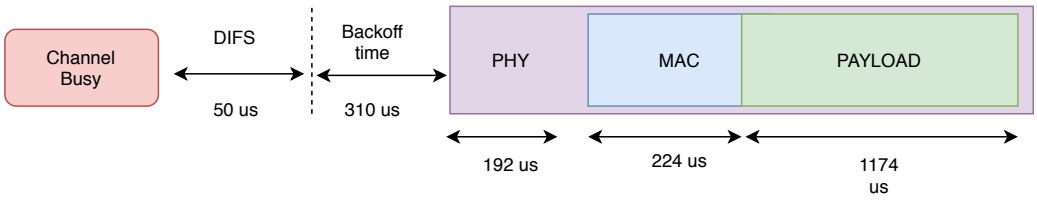


Figure 2.5: Transmission time in μs associated with a Wi-Fi frame given a bit rate of 1 Mbps for a payload with 1174 bytes. The shown backoff time is the average backoff for one transmission retry.

since 802.11 CCA scheme uses an exponential back-off. This means, if the medium is not idle for a DIFS interval, a node has to wait a random time $d_{backoff} \in [0, d_{CW}]$, where $d_{CW} = (32 \cdot 2^n - 1) * d_{cs}$, $n \in \mathbb{N}$ is the number of transmission retries and d_{cs} is the contention slot time, which is 20 μs in 802.11b.

Frame Component	Bitrate (Mbits/s)	Length (bits)	Time (μs)
PHY header: PLCP preamble	1	72	72
PHY header: PLCP header	2	48	24
MAC header (28 bytes)	5.5	224	40.73
Payload (1174 bytes)	5.5	9392	1707,64

Table 2.2: Airtime calculation of 802.11 broadcast frame on 2.4 GHz with payload of 1174 bytes and short preamble with total airtime: 1844,37 μs.

802.11a

In the 5 GHz band, 802.11a uses OFDM. OFDM chops up a channel into sub-channels or sub-carriers, which can be used in parallel for higher throughput. In theory, OFDM improves robustness against narrowband interference because only a percentage of the sub-carriers might be affected. Moreover, under the same conditions, 5 GHz has approx-

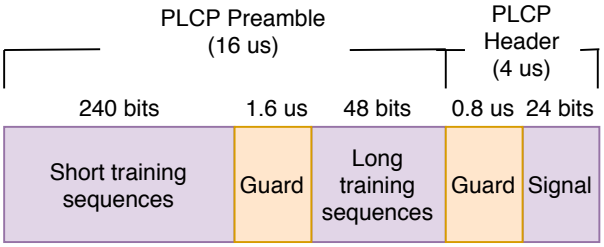


Figure 2.6: PLCP preamble and header of OFDM PHY.

imately half of the signal range of 2.4 GHz, which on the other hand makes interference due to physical proximity a bit weaker. We expand a bit on this later in this chapter.

802.11a physical protocol unit also begins with a preamble. It is composed of 12 OFDM symbols that synchronize various timers between the transmitter and the receiver. The first 10 symbols are short training sequences, which the receiver uses to lock on to the received signal, select an appropriate antenna if the receiver is using multiple antennas, and synchronize the large-scale timing relationships, required to begin decoding the following symbols. Two long training sequences follow the short training sequences, which are protected by a guard interval (see Fig. 2.6).

If we assume a payload of 1174 bytes with the lowest data rate at 5 GHz, namely 6.5 Mbps, this leads to 1468,4 μs of airtime for each frame, see Tab. 2.3. With these conditions, if we generate traffic at 600 frames per second, for instance, we occupy the channel for 88.1% of the time.

Frame Component	Bitrate (Mbits/s)	Length (bits)	Time (μs)
PHY header: PLCP preamble	-	240	16
PHY header: PLCP header	-	48	4
MAC header	6.5	22	3.4
Payload (1174 bytes)	6.5	9392	1445

Table 2.3: Airtime calculation of 802.11 broadcast frame on 5 GHz with payload of 1174 bytes with total airtime: 1468,4 μs.

2.3.3 Baseline spectrum occupation and channel selection in 802.11 networks

In this section, we perform an initial spectrum assessment with q_{cbt} to confirm that we can differentiate the spectrum occupation in different 802.11 channels by using q_{cbt}, i.e. whether we observe different distributions of channel quality values on different channels

over time without any additional influence from our part. In addition, we analyze data obtained from a public database of Wi-Fi observations, where we illustrate the current situation regarding channel selection on 802.11 networks, both globally and locally, on the TUK campus, where we perform our experiments. Moreover, we also show placement and distances between detected fixed nodes on campus, to better illustrate the need of our techniques for a more efficient channel selection and spectrum usage.

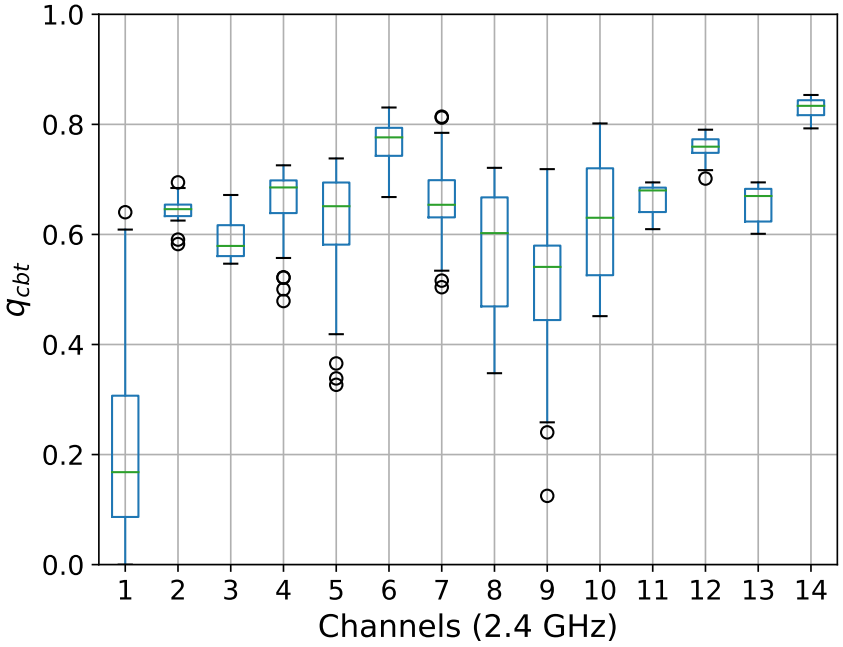


Figure 2.7: Box plot showing q_{cbt} on all channels in 2.4 GHz band with $d_{slot} = 2\,min$.

As seen in Fig. 2.7, we can definitely see that different channels have different distributions of spectrum occupation values leading to different distributions of q_{cbt}. For these experiments we have used a slot duration of 2 minutes and have run the experiments for two hours. The edges of the displayed box plots are from bottom to the top the 25th percentile and the 75th percentile respectively, and the line inside the box is the median. The whiskers that protrude out of the box shows the range of the data and outliers are plotted separately as single points. We can also see that different channels display different levels of dispersion in the q_{cbt} values. Also, channel 1, for instance, shows the largest range of values and the worst median channel quality. This is not at all surprising, since in most environments channel 1 is the most used channel by default. According to WiGLE [WiG], one of the largest public databases for Wi-Fi networks around the globe, with more than 760 million cataloged networks and 10 billion Wi-Fi

observations, channel 1 is used by 19.78% of all observed networks. In Fig. 2.8 we

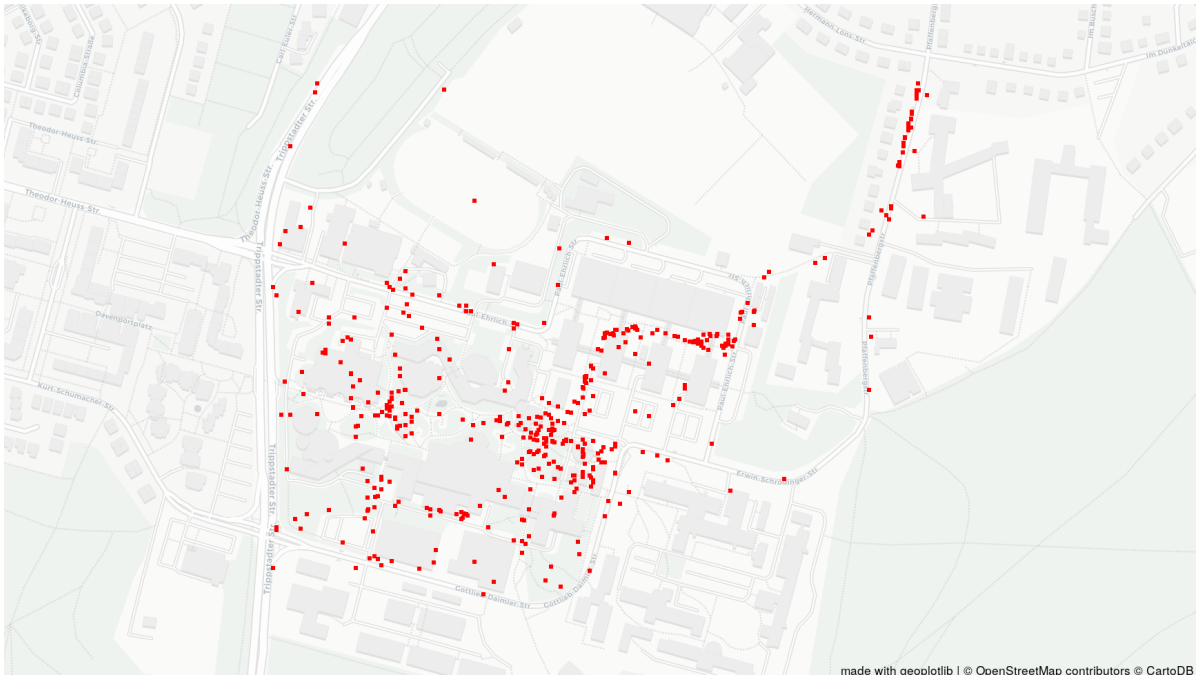


Figure 2.8: 100 802.11 routers distributed on the TU Kaiserslautern campus (data from [WiG] and map obtained from [Ope17]).

show 500 802.11 access points observed in a region of the TU Kaiserslautern campus. If we look at Fig. 2.11, we can see that the most used channels are channel 1, 6 and 11, with channel 1 being used in 31% of all devices. These three channels are often chosen to be the operation channel because they have a large enough frequency spacing between themselves and should therefore not interfere with each other. However, as we will show, close proximity between nodes makes this assumption invalid and considerable interference can still be observed within the given frequency spacing.

In the 5 GHz band, we seem to encounter far fewer nodes than in the most used 2.4 GHz channels. The most used 5 GHz channels are 36, being used in 7% of all devices, followed by 44 and 48, used in less than 5% of the observed devices. This usage frequency of channel 36 and 44 aligns itself with the global usage of respectively 4.55% and 2.64% documented by WiGLE.

In Fig. 2.9, we display the placement of the routers using the most used channels on the TUK campus in the 2.4 GHz band, i.e. 1, 6 and 11. It is clear to see that the classic blind adoption of 1, 6 or 11 leads to overcrowding of these channels. In Fig. 2.10, we can see that even though the 5 GHz band has fewer devices, the most used channels (36, 44 and 48) are used in some easily identifiable clusters, intensifying possible interference

problems. Some of these clusters in both bands have networks that are less than 10 meters apart (for comparison purposes each marker representing an AP has a radius of 8 meters). Given the longitudes λ_1 and λ_2 and latitudes φ_1 and φ_2 of respectively access points AP_1 and AP_2, we can compute the spherical distance between these points with help of the haversine formula:

$$d(AP_1, AP_2) = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

We then compute all distances between all observed Wi-Fi routers on campus that have different SSIDs and therefore service different networks. In Fig. 2.12 we show the distribution of the distances which are less than or equal to 200 meters. As seen, approximately 1000 of these distances have less than 75 meters, of which about 200 are possible communication (or at least interference) links on the same channel, considering the signal range for 802.11 varies between 35 and 120 meters.

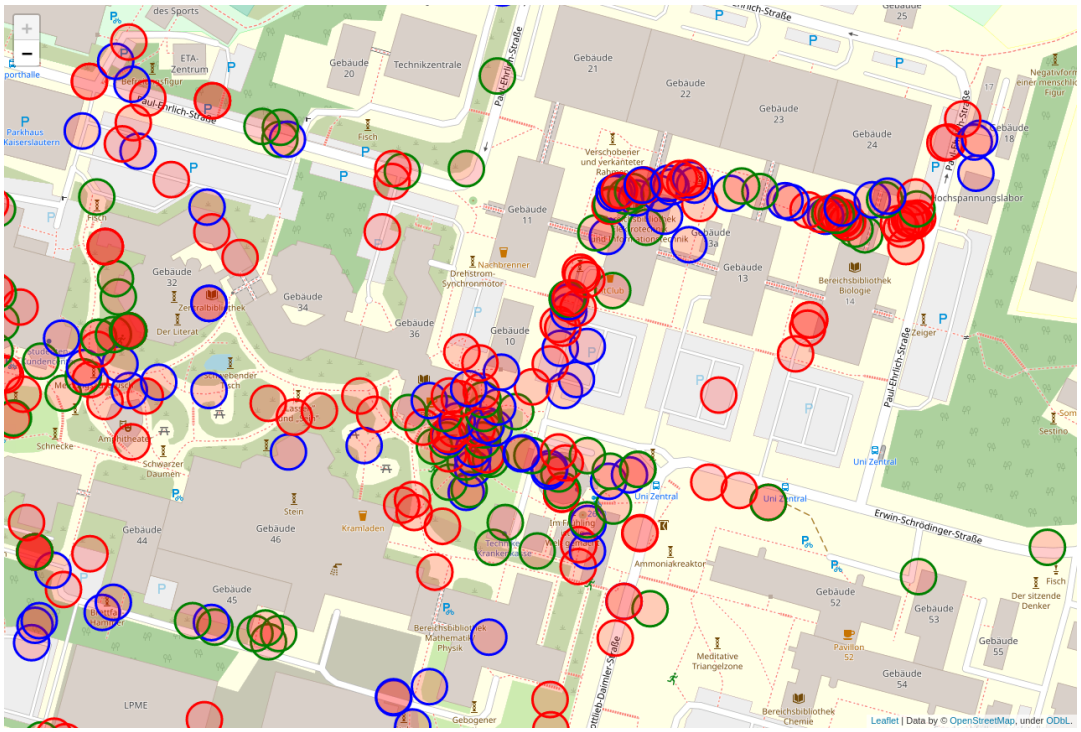


Figure 2.9: 802.11 APs on TUK campus using most used channels in the 2.4 GHz band, i.e. 1, 6 and 11, color coded by red, green and blue respectively.

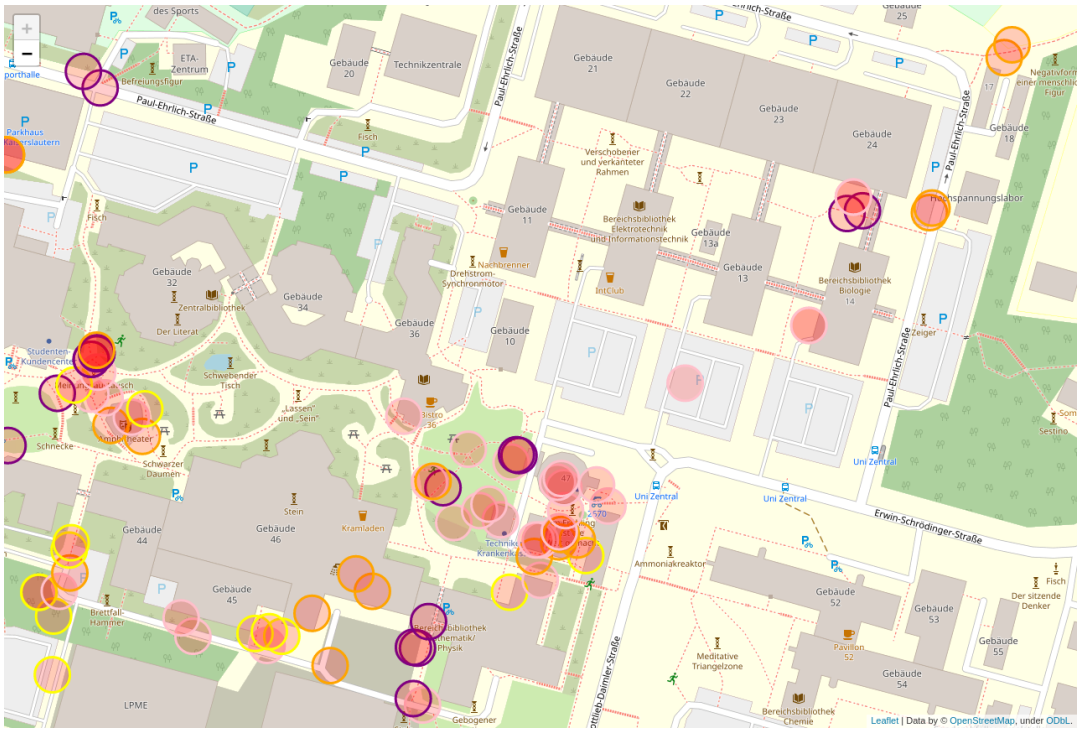


Figure 2.10: 802.11 APs on TUK campus using most used channels in the 5 GHz band, i.e. 36, 44 and 48, color coded pink, yellow and purple respectively.

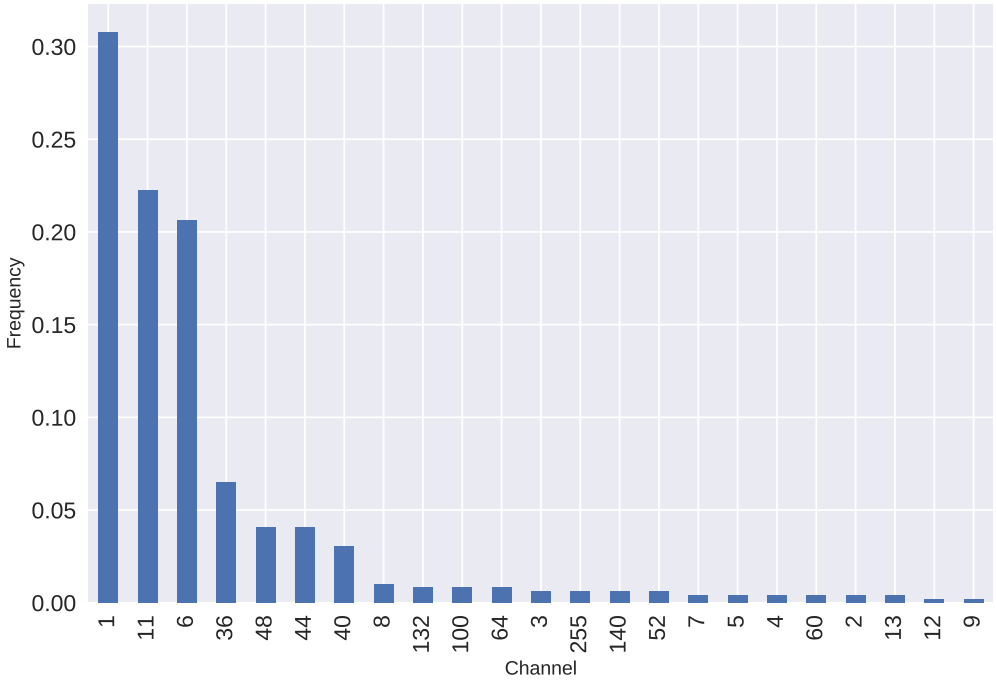


Figure 2.11: Frequency distribution of the observed channels in the devices displayed in Fig. 2.8

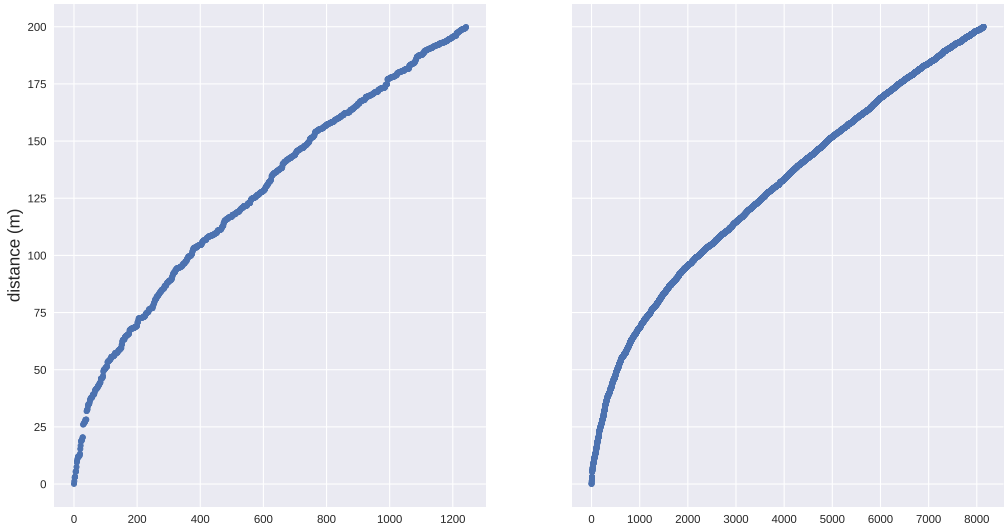


Figure 2.12: Distances between APs within different networks, i.e. with different SSIDs. We show distances where the same channel was used (on the left) or any channel combination in the same band (on the right).

2.3.4 Throughput experiments

In this section, we introduce experiments performed with our in-house traffic generator and iPerf3, a popular tool for bandwidth measurements in IP networks. A good quality metric should predict the performance of communications on a given channel, of which throughput is one of the main metrics. Whereas q_{cbt} gauges the channel state *passively*, iPerf3 computes the achieved throughput in an active form by initiating TCP/IP transmissions. Therefore, with these experiments, we intend to assess how well the spectrum occupation measured by q_{cbt} predicts the achievable throughput in *real* transmissions on each assessed channel. To determine the correlation between throughput and q_{cbt} we will compare the variations in both metrics created by a channel load generated by our traffic generator.

iPerf3, or for simplicity here also mentioned as Iperf, uses TCP by default and works with a client server architecture. Without any parameter customization, clients and servers exchange blocks of 128 KBytes over a TCP connection and the tool measures the average throughput of the connection over 10 seconds by registering the time it takes to transfer each data block. One important caveat is that Iperf measures TCP throughput and will at best only approximate the available bandwidth. This happens in special because TCP is optimized for higher reliability and tends to use conservative rates while underestimating the available bandwidth. In addition, all presented experiments had some level of cross traffic, which further limits the achievable throughput. Another possible limitation is CPU contention, i.e. other processes compete with Iperf for CPU time limiting the resulting achievable throughput, which Iperf tries to minimize by adding the possibility of using multiple connections to the same server. In our traffic generator, on the other hand, we try to account for CPU contention to obtain consistent results by setting a static scheduling priority making the traffic generator's process high priority.

For the iPerf experiments, we used two testbed nodes in managed mode; one acting as the client, sending data to the Iperf server node, which is also configured as an 802.11 access point. Reusing the server node as an AP avoids halving the achieved throughput by routing the client data over a third node.

It is important to point out that the experiments performed with our traffic generator and with Iperf were done independently with similar experimental parameters. Running these experiments at the same time would require an extra correction step, in which nodes would have to dynamically subtract the spectrum occupation created by Iperf

nodes from the measured q_{cbt}. This would be needed because Iperf nodes compete with our traffic generator for airtime and would otherwise skew the q_{cbt} results.

As a control baseline for the throughput experiments, we first ran some experiments without creating any additional channel load with the traffic generator. We can see, for instance, in Fig. 2.13 and 2.14 that according to q_{cbt} channel 11 has an average spectrum idleness close to 80%. Similarly, the Iperf baseline measurements show an average throughput of 20 Mbps, which is equivalent to 81.6% of the maximum achievable throughput (24.5 Mbps) under the configured parameters. This means that iPerf3 was able to fill the available spectrum, which implies that q_{cbt} delivers plausible throughput estimates.

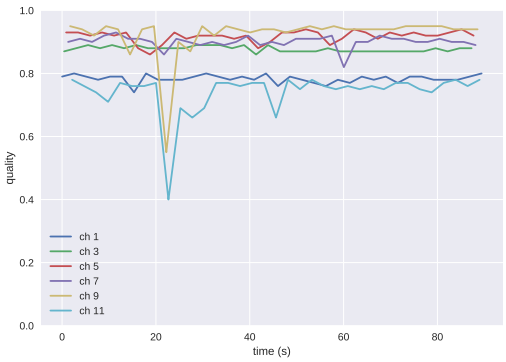


Figure 2.13: Baseline q_{cbt} measurements

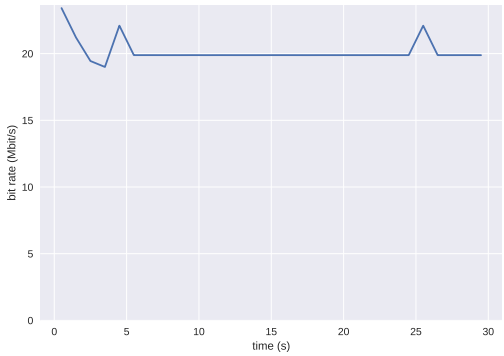


Figure 2.14: Baseline iPerf throughput measurements.

We then use our traffic generator and try to occupy as much of channel 11 as possible, achieving a spectrum occupation of 95% on channel 11, and as seen in Fig. 2.15 and Fig. 2.16, the generated effects on both q_{cbt} and the measured TCP throughput are quite similar: both metrics are very close to 0.

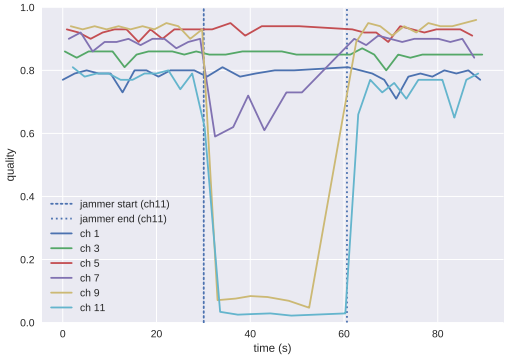


Figure 2.15: Measured q_{cbt} with jamming session.

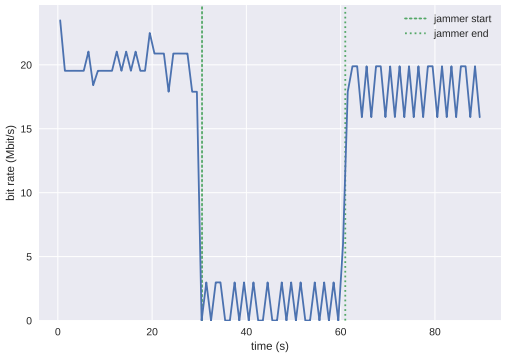


Figure 2.16: Measured througput of neighbor nodes communication.

On the other hand, if we generate a much lower spectrum occupation, as shown in Fig. 2.17, we also see a similar strong correlation between the achievable throughput on channel 11 and q_{cbt}, sampled at $d_{slot} = 200$ms. This time we occupy 25% of the channel's airtime and produce an equivalent percentual drop in q_{cbt}. The measured TCP throughput is normalized to 24.5 Mbps, the maximum achievable throughput under the used configuration parameters. It is easy to see that there is a clear overlap between the results of both experiments. In fact, all performed experiments have confirmed a high correlation between q_{cbt} and the measured achievable throughput on any given channel under different transmission conditions.

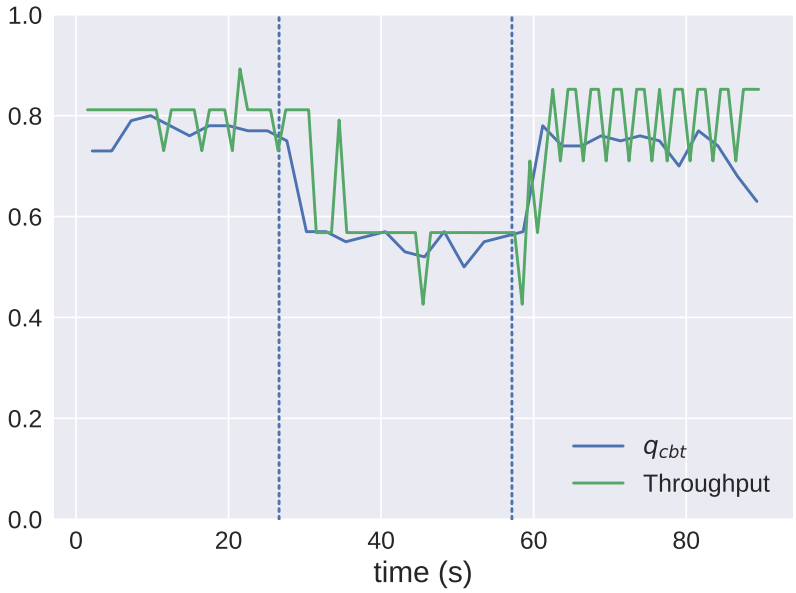


Figure 2.17: Channel quality (q_{cbt}) and achieved throughput (iPerf) on channel 11. Vertical lines show the start and end of traffic generation with 25% of total airtime.

2.3.5 Inter-channel interference

In this section, we further assess 802.11 channels with q_{cbt}, this time focusing on the channel overlapping effects to determine how transmissions on a channel affect the channel quality of neighbor (or even non-neighbor) channels. For this experimental assessment, we produce different channel loads on a 802.11 channel and analyze the impact on q_{cbt} measurements for multiple channels. As stated in our second objective in the beginning of Sec. 2.3, the observed effects happen both due to channel overlap and the near-far effect. In our context, we define the near-far effect as the increasing difficulty of distinguishing channels in the same frequency band that arises from shortening distances between nodes. Due to close proximity, energy that should be demodulated for a given center frequency *bleeds* over not only to adjacent but also further non-adjacent frequencies.

2.3.5.1 Channel overlapping and the near-far effect

In addition, the channel overlap between adjacent channels occurs within a larger frequency spacing than the nominal bandwidth. Every 802.11 standard defines a transmit spectrum mask that must be respected by any vendor in order to be certified as a Wi-Fi

device. This mask defines the amount of power relative to the total carrier power that is transmitted in a portion of the frequency space at defined offsets. As seen in Fig. 2.18, the spectral mask for 802.11 20 MHz channels shows that energy is still transmitted outside the nominal bandwidth. In the case of a 20 MHz channel c with center frequency f_c this could go up to $f_c + 30$ MHz. For instance, by transmitting on channel 1 with $f_1 = 2.412$ GHz, we transmit radio frequency (RF) energy on frequencies $2.412 - 0.030 \leq f_1^\star \leq 2.412 + 0.030 \iff 2.382 \leq f_1^\star \leq 2.442$ GHz with varying power amplitudes according to the given mask. Thus, the highest frequency we expect to be affected by a transmission on channel 1 is 2.442 GHz, the center frequency of channel 7. This means that by considering all components of the spectral mask, channel 1 and channel 7 do have some overlap, which does not occur if we only take into consideration the effective bandwidth of 22 MHz (see Fig. 2.19).

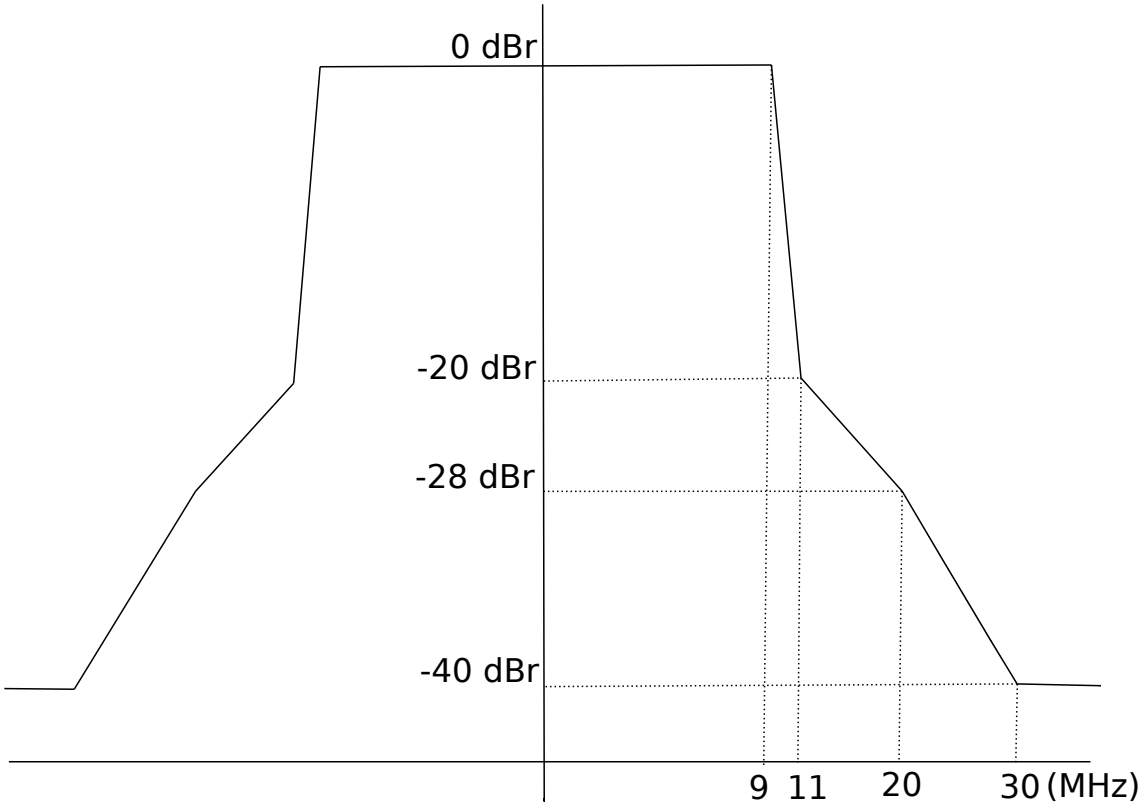


Figure 2.18: Spectral mask for transmission on 20 MHz channels with 802.11a/g/n/ac. Power amplitudes are given in dBr, which means dB relative to the maximum spectral density of the signal.

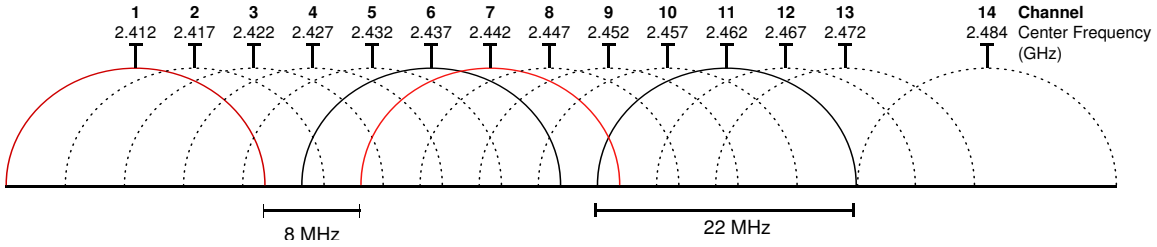


Figure 2.19: 20 MHz channels in 2.4 GHz band.

It is widely assumed in the literature that a wide enough frequency spacing between channel center frequencies suffices to guarantee coexisting transmissions without any harmful interference. Channels 5 and 11 of the 2.4 GHz band, for example, do not have any nominal overlap and hence groups of nodes using either of these channels should be able to communicate without interfering with each other. However, as our own experiments have shown, if these nodes are close enough to each other, e.g. with a distance of less than a meter, we have a strong interference that carries over much further than expected in the frequency space. This effect has not been researched in depth in 802.11-based networks, but has been observed at [FVR07] and further confirmed by other studies. For instance, [CRS06] reported detecting interference between channel 1 and channel 11. [FVR07] reported that the non-overlapping channel interference was most noticeable for distances below or equal to 35 cm. In fact, the authors declared that the interference behavior observed between non-overlapping channels used by nodes in close proximity was equivalent to the interference of nodes using the same channel. Moreover, [RPD+05] reported that by using connectors with length of 1 m (achieving a distance of 35 db between both antennas in the node) the non-overlapping channel interference was not detected anymore. This interference between non-overlapping channels can lead to both frame corruption due to interference and channel contention due to channels being sensed as busy. Also, further experiments in [RPD+05] investigated multi-antenna interference and throughput loss due to multiple network cards in a single node: the authors placed multiple PCI network cards on a single node and noted a decrease in throughput for every additional card, added in passive mode (starting with the third card). Since all additional cards were not active, any interaction between the cards was due to radiation leakage from the chipset, eventual connectors and antennas [RPD+05]. How much this cross-talk affects the sending throughput of the active network card depends on each hardware platform, since different platforms might e.g. provide different levels of radiation shielding.

2.3.5.2 Experiments

First, we ran experiments measuring q_{cbt} on multiple nodes. The traffic generator ran on node 1 and the testbed nodes 1, 9, 11 and 13 worked as sensing nodes. Hence, it is expected to observe the strongest interference effects when gauging q_{cbt} from node 1 which simultaneously transmits on one of the sensed channels with an antenna that is only 24 cm apart from the sensing antenna.

The second stage of the experiments used Iperf to evaluate the channel overlapping and near-far effects on the measured TCP throughput and compare them to the effects observed when measuring q_{cbt}. Similarly to our throughput experiments, we created an AP and this time the access point also used our traffic generation tool to transmit data on a chosen channel. The traffic generator used the same settings of the first phase of the experiments and ran again on node 1.

Before taking a look at the main results of our experiments, we first need to understand how Iperf performs its throughput measurements. As already mentioned, an Iperf client sends data to an Iperf server using TCP. The sender determines whether the sent packets were received correctly by receiving ACKs from the server. If some of these acknowledgments are lost on the return path, Iperf deems the reception of the original packet as unsuccessful.

For the presented experiments we used the node placement shown in Fig. 2.20. Nodes 1 and 13 are spaced 2 meters apart and the minimal distance between the antennas of different transceivers on each node is 24 cm.

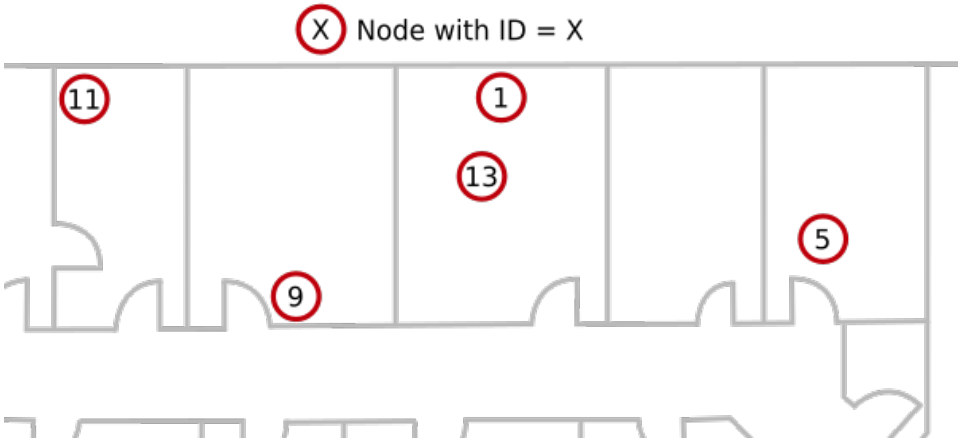


Figure 2.20: Node placement for the experiments.

2.4 GHz

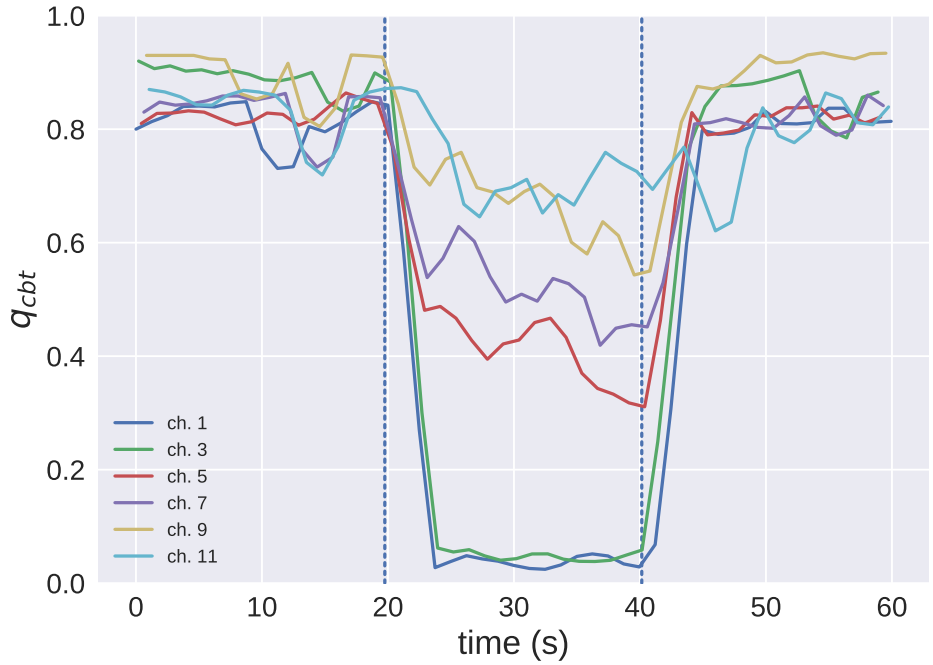


Figure 2.21: q_{cbt} measurements performed on node 1. Vertical lines show the start and end of traffic generation on channel 1 with 90% of total airtime.

In Fig. 2.21, we show the results of experiments carried out in the 2.4 GHz band where we measure q_{cbt} on multiple channels with the traffic generator running on channel 1 with a spectrum occupation of 90%. As we can see in this figure, even channels that are quite far apart from channel 1 are still affected by traffic generated on it, e.g. channel 11. These measurements were taken on node 1, which is also responsible for the generated traffic. In Fig. 2.22, we show the effects on the achieved throughput on channels 1, 9 and 11 of generated traffic with an average spectrum occupation of 60% on channel 1. As illustrated, channel 1 suffers a loss in throughput that is not as high as the losses suffered by channels 9 and 11. This seems at first to contradict our intuition, since channel 1 is the channel on which our traffic is generated and which accordingly suffers the greater loss in q_{cbt}. In fact, the throughput measurements on each channel are obtained independently running the traffic generator with the same parameters once per channel. When running on the same channel as the Iperf measurements, the traffic generator has to compete frequently with Iperf for airtime leading to a lower average frame rate and a lower final

spectrum occupation than the spectrum occupation it obtains when Iperf transmissions happen on other channels.

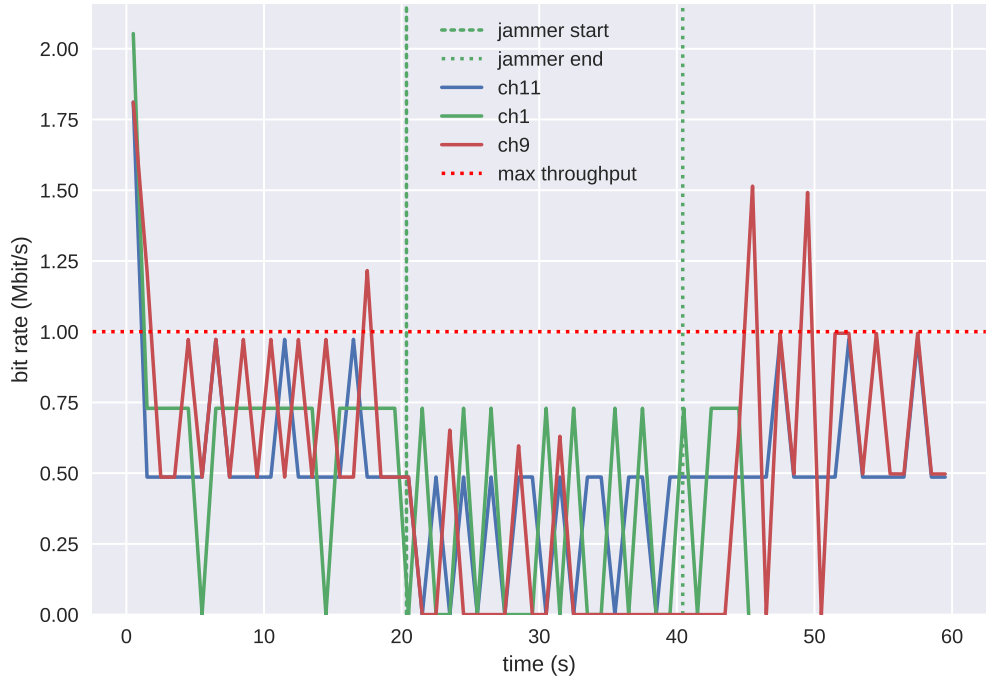


Figure 2.22: iPerf3 throughput test node 13 → 1, with traffic generated on channel 1 with spectrum occupation of 60%.

It is also important to remember this is the extreme case where the sensing node measuring q_{cbt} and the achieved throughput is also the traffic generator with a spacing of only 24 cm between sensing and transmitting antennas.

In addition, in Fig. 2.23 and 2.24 those channels that should have no channel overlap with channel 1 are still affected by the near-far effect, but much less severely. Channel 11 was not affected and is not displayed in Fig. 2.24. However, channel 5 not only suffers considerable losses in q_{cbt}, but we also seem to have caused a malfunction on iPerf's measurements, probably due to the generated traffic corrupting frames on the channel. It is possible to see in Fig. 2.22 and Fig. 2.24 that Iperf registers achieved throughput values higher than the configured bitrate of the used network cards, which should be the maximum achievable throughput. These spikes indicate Iperf's measurement errors presumably triggered by re-sent TCP ACKs being detected as new additional successful

transmissions and artificially increasing the achieved throughput. This phenomenon was observed in measurements taken on multiple nodes and also on the 5GHz band.

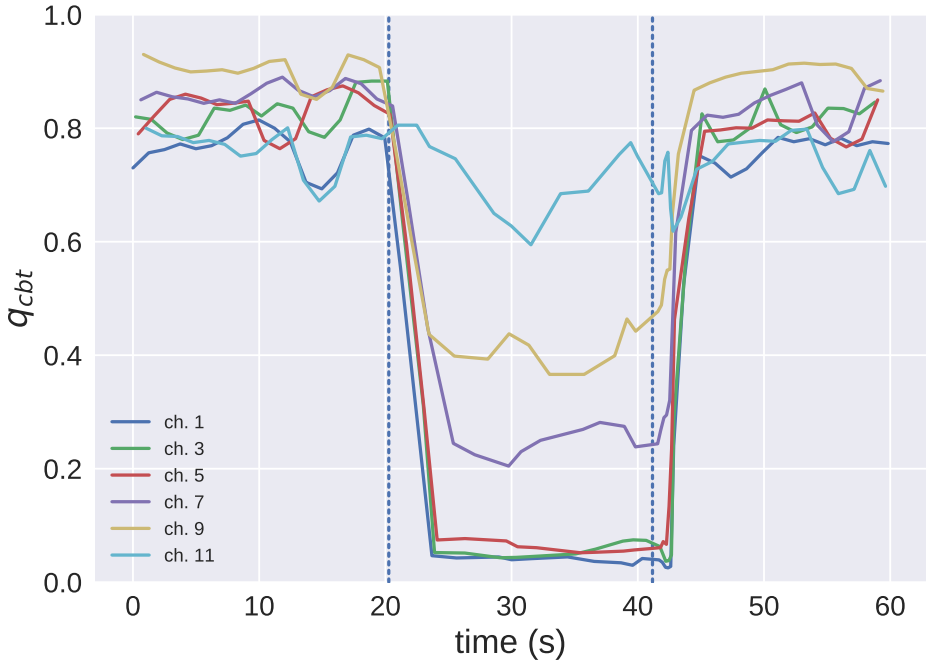


Figure 2.23: q_{cbt} measurements performed on on node 13, placed 2 meters apart from node 1. Vertical lines show the start and end of traffic generation on channel 1 with 90% of total airtime.

If we move our experiments to node 9 placed in the room next to the room where node 1 is located, we see in Fig. 2.25 that channel 5 is not as affected as in the previous experiments.

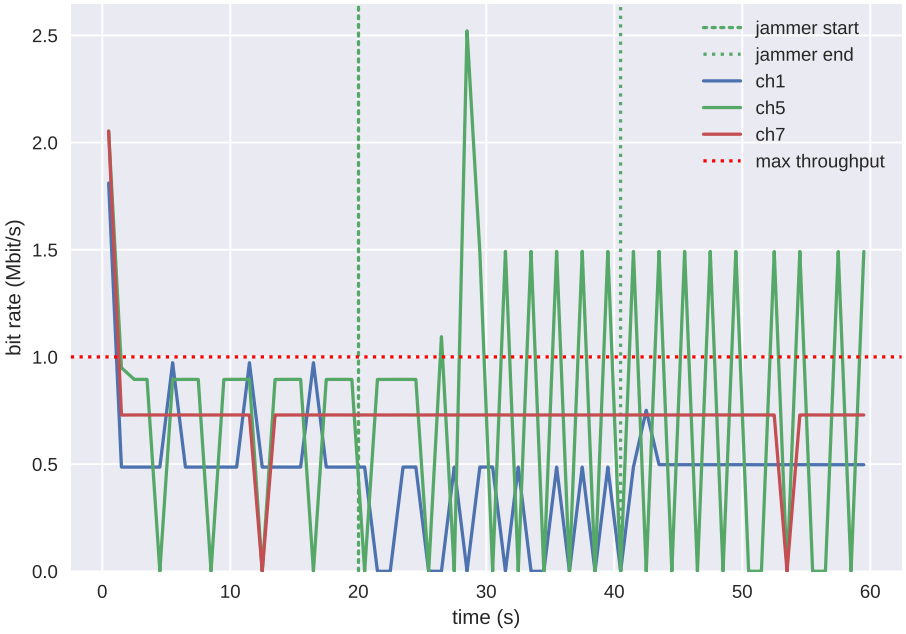


Figure 2.24: iPerf3 throughput test node 9 → 13, with traffic generated on channel 1 with spectrum occupation of 70%.

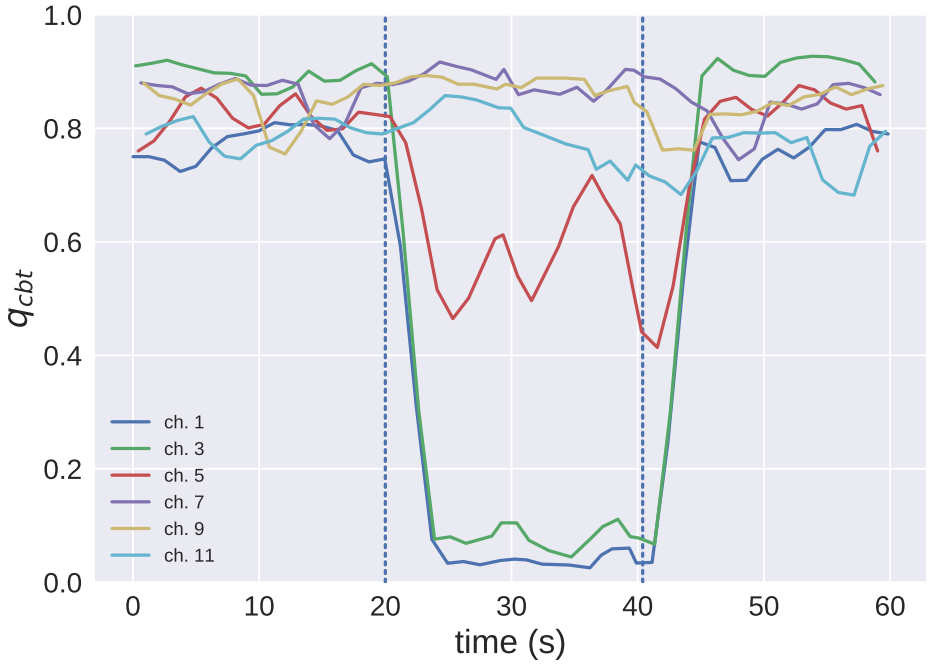


Figure 2.25: q_{cbt} measurement on node 9. Vertical lines show the start and end of traffic generation on channel 1 with 90% of total airtime.

5 GHz

Similar effects to those obtained in the 2.4 GHz band can be observed in the 5 GHz band. In our first experiment, we gauge q_{cbt} on node 13 while we use the same node for traffic generation on channel 40. As expected, we can see in Fig. 2.26 that the channel which suffers the highest losses in channel quality is channel 40, followed by its adjacent channels 36 and 44. Channel 48 is also affected, but with much less intensity. All other channels are unaffected and not shown for better visualization. We see quite similar results in Fig. 2.27, with exception of the throughput measurement errors on channel 48.

The observed effects on q_{cbt} and throughput on channels 36 and 44 are still present in the measurements performed at node 1 but weaker, see Fig. 2.28. The interference effects on channel 48 however are already gone. It is clear that the channel overlap and near-far effects in the 5 GHz band are overall weaker than those observed in the 2.4 GHz band. This has two main reasons: 20 MHz channels in the 5 GHz have a greater spacing between them and higher frequencies lead to shorter transmission ranges, since the shorter wavelengths of high frequency signals make them more susceptible to energy loss due to collisions within the environment. In fact, the received power in dBm at any given antenna can be estimated with the Friis transmission equation:

$$P_r(dBm) = P_t + G_t + G_r + 20\log_{10}(\frac{\lambda}{4 \cdot \pi R})$$

where λ is the carrier's wavelength and R the distance between the transmitter and the receiver. This means, that under the same antenna conditions, higher frequencies result in higher path loss and hence shorter range.

In Fig. 2.29, we show the results of measuring q_{cbt} on node 9 placed more than 5 meters apart from node 13 and with a wall in between. As seen, we can still observe a significant effect on q_{cbt} on channels 36 and 44 due to the generated transmissions on channel 40.

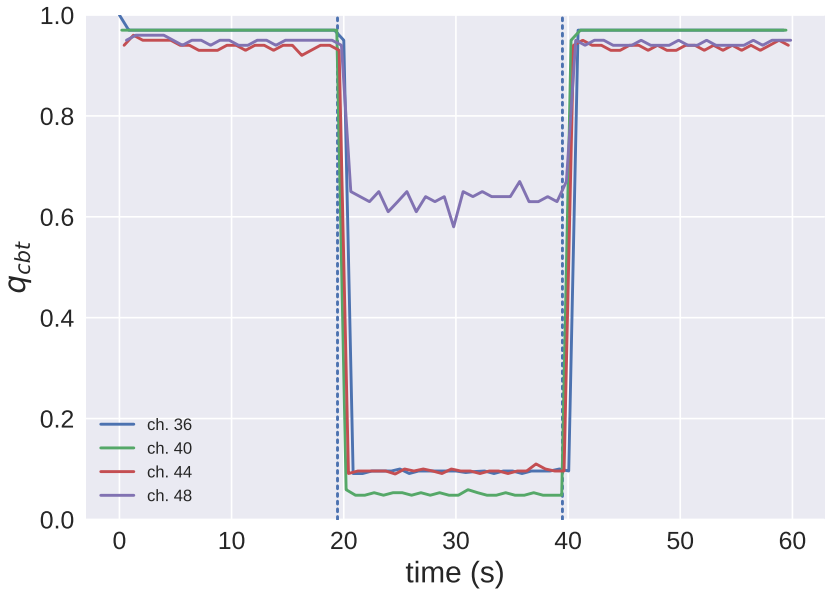


Figure 2.26: q_{cbt} measurement on node 13. Vertical lines show the start and end of traffic generation on channel 40 with spectrum occupation of 90%.

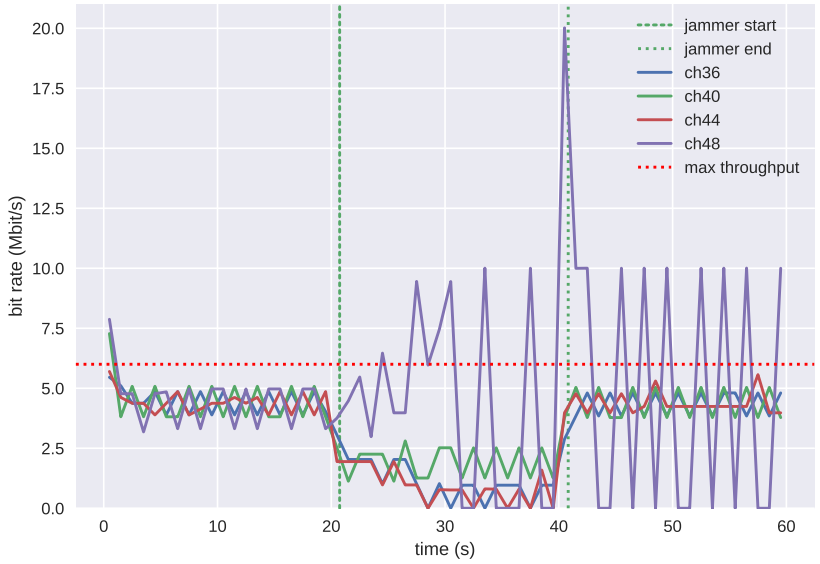


Figure 2.27: iPerf3 throughput test node 13 $\rightarrow$ 1, with traffic generated on channel 40 with spectrum occupation of 70%.

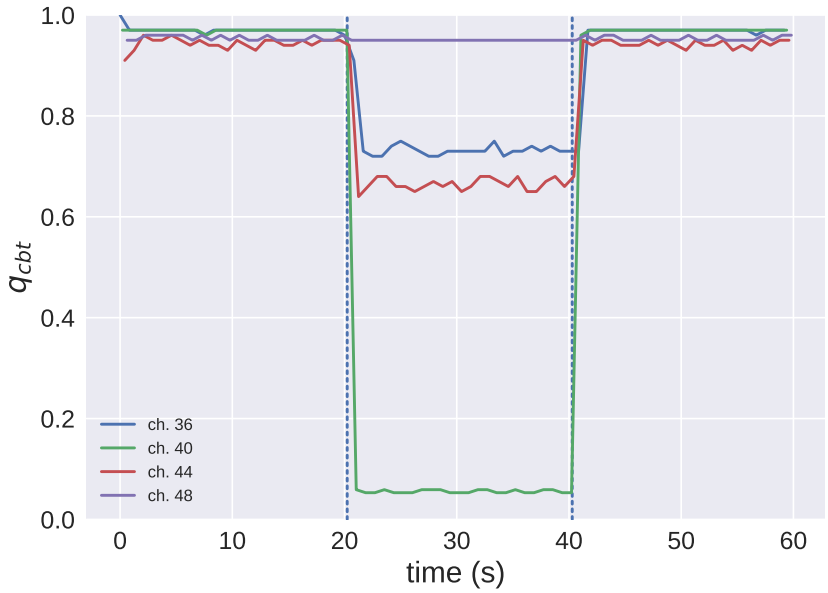


Figure 2.28: q_{cbt} measurement on node 1. Vertical lines show the start and end of traffic generation on channel 40 with 90% of total airtime.

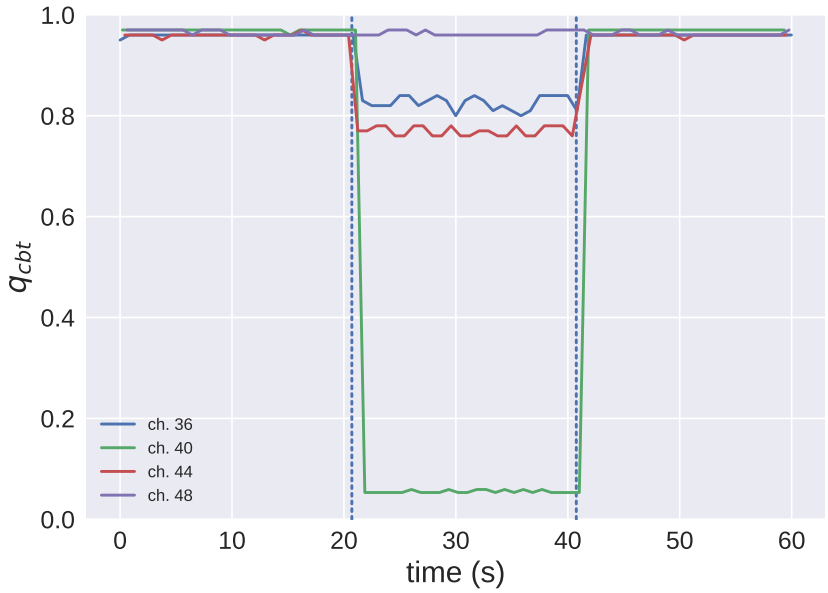


Figure 2.29: q_{cbt} measurement on node 9. Vertical lines show the start and end of traffic generation on channel 40 with 90% of total airtime.

2.4 Correcting q_{cbt}

In this section, we describe needed corrections to q_{cbt} in order not to skew its raw values by transmissions of internal nodes. The channel quality metric q_{cbt} should estimate the available bandwidth on a channel w.r.t. spectrum occupation of foreign nodes. By measuring energy on the medium we obtain d_{busy} values that are affected not only by external transmissions but also by those of internal nodes, which have to be accounted and corrected for. As already shown in the previous section, transmissions on a given channel will impact the busy times measured in multiple channels. As our experiments show, this interference between channels varies according to both the spacing between center frequencies as well as physical proximity between sender and receiver, assuming constant transmit power levels. Depending on the distance between sender and receiver, this impact will carry over to channels that in theory should have no overlap. This effect is made quite prominent in our testbed nodes in which the transceiver's antennas of different transceivers are only 24 cm apart.

With this in mind, we introduce our q_{cbt} correction models. Each node that is currently transmitting has to correct its measured busy times $d_{busy}(c')$ on every affected channel c', based both on its transmission times d_{send} on channel c, the variation $\Delta d_{busy}(c')$ of measured d_{busy} on channel c', induced by its transmissions on channel c', and the spacing $\Delta(c, c')$ between the center frequencies of channels c and c'. For our correction model, we assume that in the worst case the whole sending time d_{send} will be added up to the measured busy time. This way, we derive corrected busy times $d^{\star}_{busy}$, such that

$$d^{\star}_{busy}(c') = d_{busy}(c') - \omega^{\star}(c, c') \cdot d_{send} \tag{2.2}$$

To complete our correction model, we fit a curve mapping $\Delta(c, c')$ values to observed $\omega^{\star}(c, c') = \frac{\Delta d_{busy}(c')}{d_{send}}$ values. Note that we have to fit a model for each frequency band used by IEEE 802.11n, namely 2.4 GHz and 5GHz. The need to compute two distinct models arises from the different physical layer parameters employed in each frequency band, such as frequency spacing between neighbor channels, channel bandwidth and chosen modulation schemes. The distinct parameters lead hence to different interference behaviors between different channels requiring different models.

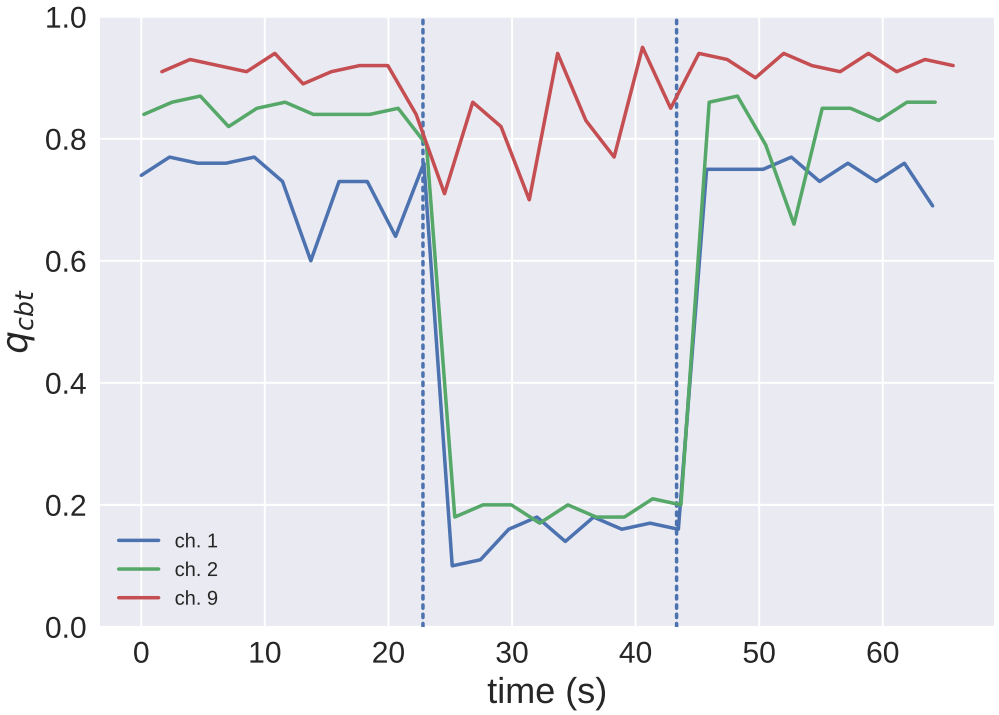


Figure 2.30: Vertical lines show start and end of traffic generation with 60% airtime on channel 1.

2.4.1 Correction model for 2.4 GHz

If we look at Fig. 2.30, where 60% of available airtime on channel 1 is occupied by traffic generated by our traffic generator, we can get an intuition of the needed corrections in the 2.4 GHz band. As seen in Fig. 2.30, the furthest channel from channel 1 which is still affected by our generated traffic is channel 9, whose center frequency lies 40 MHz away from the center frequency of channel 1. We conducted a series of experiments that delivered similar results, and based on the obtained experimental data, we have fitted a correction model for q_{cbt} in the 2.4 GHz band, as shown in Fig. 2.31.

Given $\Delta(c, c') = |f_{c'} - f_c|$ the difference in MHz between the center frequencies of channel c and c', we derive the correction factor $\omega^\star(c, c')$ as follows

$$\omega^\star(c,c') = \begin{cases} 1 & \text{if } 0 \leq \Delta(c,c') \leq 15 \\ -0.235 + \frac{0.875}{(1+\frac{\Delta(c,c')}{35.4}^{5.8})} & \text{if } 20 \leq \Delta(c,c') \leq 40 \\ 0 & \text{else} \end{cases}$$

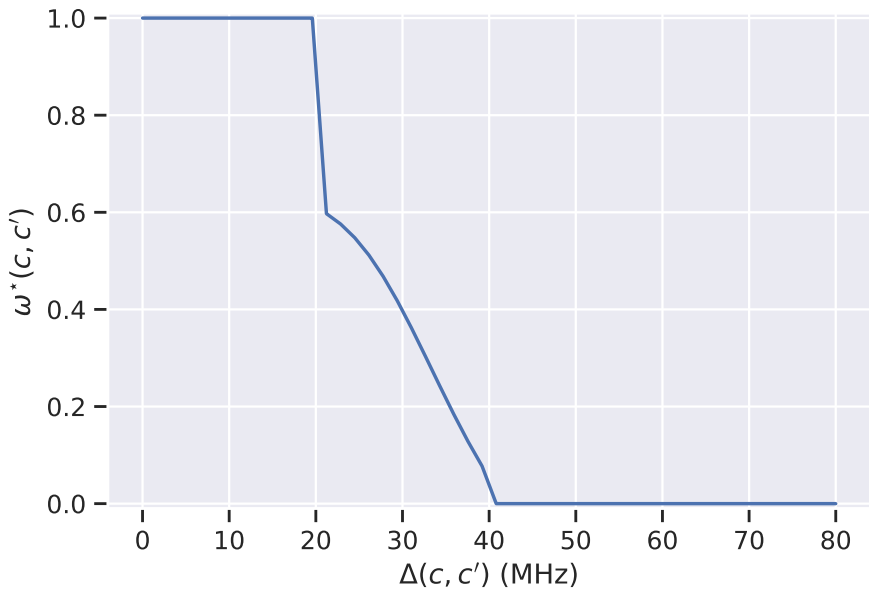


Figure 2.31: $\omega^\star(c, c')$ in 2.4 GHz band.

2.4.2 Correction model for 5 GHz

In the 5 GHz band, we observe a slightly different interference behavior. As shown in Fig. 2.32, a transmission e.g. on channel 36 affects q_{cbt} on all channels up to channel 52.

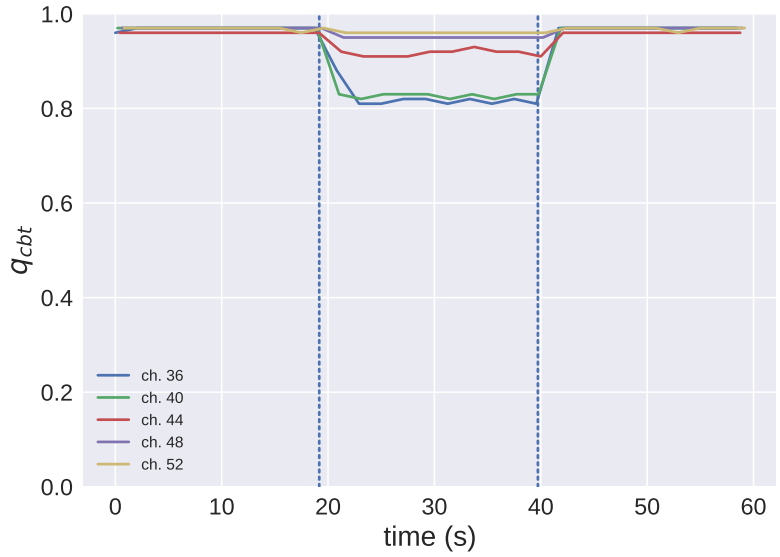


Figure 2.32: Vertical lines show start and end of traffic generation with 20% airtime on channel 36.

Moreover, this holds for any different combination of channels within a frequency span of 80 MHz. Again, from experimental observations, we derive a correction model for q_{cbt} in the 5 GHz band (see Fig. 2.33):

$$\omega^{\star}(c,c') = \begin{cases} 0.04 + \frac{0.96}{(1+(\frac{\Delta(c,c')}{35})^{4.7})} & \text{if } 0 \leq \Delta(c,c') \leq 80 \\ 0 & \text{else} \end{cases}$$

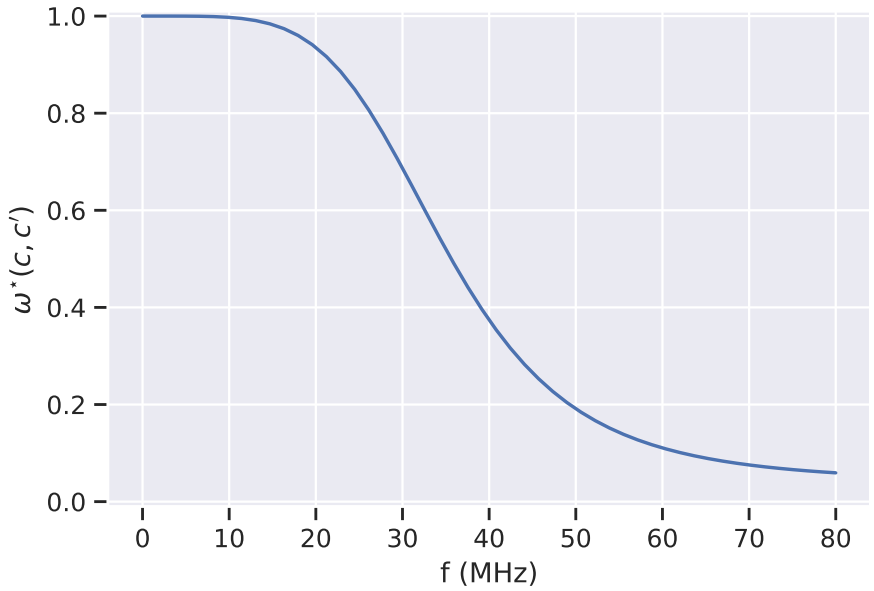


Figure 2.33: $\omega^\star(c, c')$ in 5 GHz band.

2.5 Aggregating q_{cbt}

In this section, we describe different aggregation functions applied to channel sensing raw data. Aggregation serves the purpose of summarizing multiple observations into one single value. Communication schedules, for instance, require a single channel quality value to be computed. Furthermore, as already indicated in Chap. 1, an ideal channel quality metric should have good stability and adaptivity. By combining two different aggregation methods, we try to reconcile these conflicting objectives.

Since channel sensing is essential to each node's operation and is performed throughout its active period, the volume of generated data is tremendous. Given nodes with restricted volatile memory and processing power, it is infeasible to store and aggregate all these data in memory. On the other hand, the relevance of the sensed data decreases with time and the most important values are the most recent ones. In general, we want to have sensing data that reflects the current usage of a given channel with just enough historical values in the mix to prove useful in detecting anomalies or changes in behavior.

By displaying good levels of stability, a quality metric can prevent superfluous recomputations of communication schedules due to nonsignificant changes in channel quality. At the same time, we want to stay adaptive enough, such that a network can react to significant changes in channel usage. This way, a node might, for instance, update its channel hopping sequence in accordance with the new channel conditions.

To provide the desired stability, we aggregate each raw value $q_{cbt}(i)$ measured at slot i with the past n_{mw} observations by applying an arithmetic mean, yielding $\overline{q_{cbt}}(i)$. We have hence a moving window, such that the most recent value always evicts the oldest observation in a First In, First Out (FIFO) manner. However, this aggregation method alone can completely bias the resulting metric towards the recently seen values and might ignore an overarching trend by having a focus that is too narrow.

Therefore, to guarantee an appropriate level of adaptivity, we incorporate measurements outside the current moving window with diminishing weights. This is achieved through the application of an exponential weighted moving average (EWMA). As hinted by the name of this function, provided weights fall exponentially.

The combination of a moving average with an EWMA results in the aggregated channel quality q_{aggr} (see Fig. 2.34):

$$q_{aggr}(i) = \begin{cases} \overline{q_{cbt}}(i) & \text{if } i = n_{mw} \\ \alpha \cdot \overline{q_{cbt}}(i) + (1-\alpha) \cdot q_{aggr}(i-1) & \text{if } i > n_{mw} \\ undefined & \text{else} \end{cases}$$

where α is the weight of the latest measurement. For our experiments, unless otherwise stated we used a moving window of 10 seconds and $\alpha = 0.8$.

An alternative approach proposed in the literature that might achieve similar results to an EWMA is the biased reservoir sampling [Agg06]. The basic idea behind this approach is that instead of using a moving window with FIFO eviction, it uses a window with probabilistic insertions and evictions. In fact, biased reservoir sampling is a sophisticated variant of the simple reservoir sampling, which uses following random eviction policy:

1. Eviction starts, once the simple reservoir reaches its maximum size. If the reservoir is not full, all new measurements are inserted.

2. If the reservoir is full, for any incoming measurements, the simple reservoir randomly ejects a data point with *equal probability*.

This simple form of sampling is proved to be unbiased and not to consider the changes in the behavior of the data stream over time. In fact, with passing time an unbiased sample will contain more and more points from the distant history of the data stream. If overall channel quality conditions change, the vast majority of the measurements in the reservoir may represent a stale history of the channel. In a biased reservoir sampling, however, temporal bias functions are used such that newer observations substitute older

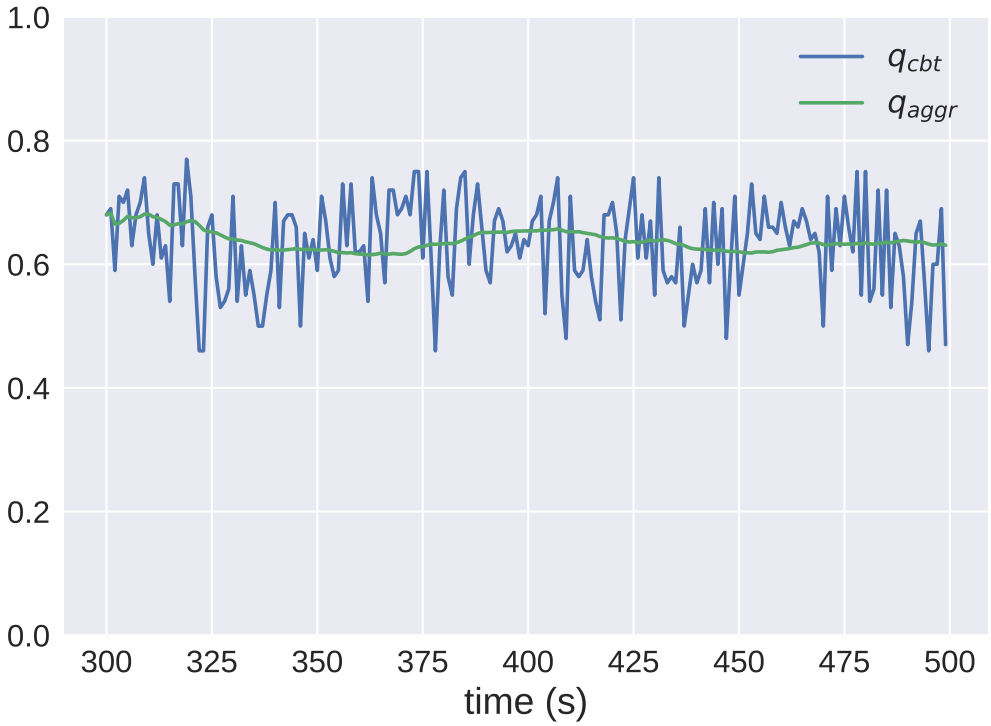


Figure 2.34: Comparison of raw q_{cbt} and the aggregation q_{aggr}.

ones with higher probability. Actually, for every point r currently in the reservoir a probability P(r,t) is computed to determine whether r stays in the reservoir upon the arrival of t, i.e. whether t substitutes r. Even though, biased sampling adds adaptivity to simple reservoir sampling, its implementation is nontrivial, and with exception of particular use cases, might not be efficiently implementable in general, as maintained by its authors [Agg06].

2.6 Volatility

The notion of volatility has been used throughout the literature and mostly focused on price movements of financial instruments such as stocks and options. Volatility is a statistical measure of dispersion in a time series. Values delivered by a given series vary with time and all these value fluctuations can be compiled into a volatility metric. This volatility delivers non-negative real numbers that increase with the increase of dispersion in the output variable, e.g. raw channel quality. Statistically, a set of observations can be interpreted as a sample from an unknown population.

Most works that study wireless networks and volatility focus on characterizing and distinguishing the different causes for fluctuations in frame-based quality metrics. The most studied metrics in this respect are the received signal strength indicator (RSSI) and the link quality indicator (LQI). The variations in the values of these metrics can happen due to many factors, such as changes in temperature, humidity, antenna alignment, external interference and node mobility [Bil13, WD21].

However, a deeper study of how to handle these fluctuations in the context of channel sensing is still mostly absent, especially with regards to the channel busy ratio. Our approach is to the best of our knowledge the first to introduce a general penalty-based volatility-aware channel quality assessment framework.

Probably, the most used measure of statistical dispersion throughout the literature is the standard deviation. The sample standard deviation uses the size of the sample as the size of the population and is computed as the square root of the variance. If we compute the standard deviation of raw channel qualities in a moving window of size n_{mw}, we have:

$$s = \sqrt{\frac{1}{n_{mw}} \sum_{i=1}^{n_{mw}} (q_i - \bar{q})^2}.$$

where $\bar{q}$ is the sample mean.

For our purposes, it is not sufficient to take into consideration the total volatility of the raw channel quality. In fact, it is essential to make a distinction between downward and upward volatility, in special because the channel quality distribution is in general not symmetrical. This means we have distinct probabilities for channel qualities to fall below or above a chosen target value, e.g. the sample mean or the median. In addition, decreasing the number of slots assigned to channels with higher *total* volatility would penalize not only those channels with increasing downside volatility but also those with increasing upside volatility, i.e. movements in channel quality that signal an improvement in channel conditions.

In order to rank channels based on their aggregated channel qualities and on the level of *downward* volatility, we need downward volatility metrics. To the best of our knowledge, Markovitz [Mar59] proposed the first downward volatility metric, i.e. the semivariance. This metric can be interpreted as a downside risk of an asset and was introduced to guide the selection of an investment portfolio to maximize expected returns. The squared root of the semivariance computed at a slot i in a window of n_{mw} observations is the semi-deviation:

$$\sigma_{sed}(i) = \sqrt{\frac{1}{n_{mw}} \cdot \sum_{j=1}^{n_{mw}} \phi_{sed}(i,j)^2}$$

where given $\overline{q}(i)$ the average quality in the moving window $k = i - j + 1$

$$\phi_{sed}(i,j) = \begin{cases} \overline{q}(i) - q_{cbt}(k) & \text{if } \overline{q}(i) > q_{cbt}(k) \\ 0 & \text{else} \end{cases}$$

In this section, we introduce and compare three downward volatility metrics. From these metrics we derive three penalization schemes for channel qualities, which yield the penalized channel qualities q_{adda}, q_{dsd} and q_{phv}. All metrics are computed for a moving window. By only considering downward fluctuations in the current window, we try to obtain a meaningful estimate of the downward volatility in the near future.

2.6.1 Downward Standard Deviation σ_{dsd}

The downward standard deviation σ_{dsd} is a generalization of the semi-deviation. It quantifies downward volatility by adding up the distances to the *aggregated* channel quality q_{aggr} in the current moving window *only* from those data points that fall *below* it. In the literature, σ_{dsd} is referred as the below-target semi-deviation for a target t,

where $t = q_{aggr}$. If the aggregated channel quality is the sample mean, then σ_{dsd} is the semi-deviation. Given slot i:

$$\sigma_{dsd}(i) = \sqrt{\frac{1}{n_{mw}} \cdot \sum_{j=1}^{n_{mw}} \phi_{dd}(i,j)^2}$$

where given $k = i - j + 1$

$$\phi_{dd}(i,j) = \begin{cases} q_{aggr}(i) - q_{cbt}(k) & \text{if } q_{aggr}(i) > q_{cbt}(k) \\ 0 & \text{else} \end{cases}$$

2.6.2 Average Downward Deviation from the Aggregation σ_{adda}

The volatility metric σ_{adda} has a similar shape to σ_{dsd}, but sums up the distances between aggregated and raw qualities in a linear fashion. Given slot i:

$$\sigma_{adda}(i) = \frac{1}{s_{mw}} \cdot \sum_{j=1}^{s_{mw}} \phi_{dd}(i,j)$$

2.6.3 Downward Parkinson Historical Volatility σ_{phv}

Parkinson [Par80] introduced an extreme value method to estimate the variance of the rate of return for stock prices. The basic assumption used for the derivation of this metric, that has been since referred to as the Parkinson Historical Volatility (PHV), is that given P the price of a stock, ln(P) approximates a random walk. The volatility of the prices, quoted at equal time intervals, is then computed as the diffusion constant of the underlying ln(P) random walk.

The main insight used by Parkinson that distinguishes his approach from a classical method such as the standard deviation is that instead of adding up distances from each input value to a measure of central tendency, it sums up the distances between high and low prices within each measuring period, e.g. weekly. This way, due to its higher granularity, PHV is more sensitive to fluctuations than the standard deviation.

In our approach, we divide our moving window into a fixed number of equal-sized window slices. Also, since our investment assets are channels instead of stocks, in each of those slices, we compute the maximum and the minimum raw channel qualities, denoted respectively by q_{max} and q_{min}.

However, much as with the standard deviation, it is not enough for us to apply PHV without further modifications. Since it measures the total volatility in a time period,

and we need to estimate the *downward* volatility of the channel qualities, we propose a modified version of PHV, the downward Parkinson historical volatility σ_{phv}. Our main modification consists in only adding up the distances between q_{max} and q_{min} for those slices where a downward movement is detected. In addition, we adopt a different scale than the original metric, by changing $ln(\frac{q_{max}}{q_{min}}) = ln(q_{max}) - ln(q_{min})$ into $q_{max} - q_{min}$.

Detecting downward trends requires an adequate mathematical tool. The tool we picked for the job is linear regression, most specifically ordinary least-squares (OLS). This regression approach approximates a model for a set of data using a quadratic loss function of the form $L(\hat{q}, q) = (\hat{q} - q)^2$, where $\hat{q}$ is the estimated value and q the value obtained from observation. Given a set of n_{slice} observations in a window slice, OLS fits a linear model to this set such that the sum of squared residuals $(\hat{q} - q)^2$ is minimized. For a window slice with raw channel qualities $q(t)$ in a time slot t, the resulting line is of the form $q(t) = \alpha + \beta \cdot t$, where α and β can be computed as:

$$\begin{aligned} \beta &= \frac{n_{slice} \sum_{t=1}^{n_{slice}} t \cdot q(t) - \sum_{t=1}^{n_{slice}} t \sum_{t=1}^{n_{slice}} q(t)}{n \sum_{t=1}^{n_{slice}} t^2 - (\sum_{i=1}^{n_{slice}} t)^2} \\ \alpha &= \overline{q} - \frac{\beta}{2} \cdot (n_{slice} + 1) \ , \end{aligned} \tag{2.3}$$

Given slot i and a window slice j, we then define σ_{phv} as:

$$\sigma_{phv}(i) = \sqrt{\frac{1}{4 \cdot \ln 2 \cdot n_{slice}} \cdot \sum_{j=1}^{n_{slice}} \phi_{phv}(j)^2}$$

where

$$\phi_{phv}(j) = \begin{cases} q_{max}(j) - q_{min}(j) & \text{if } \beta(j) < 0 \\ 0 & \text{else} \end{cases}$$

This linear model works very well if there is enough spacing between surges and drops in channel qualities and if the fluctuations in value are not too extreme. Otherwise, the resulting linear model can be misspecified and no single downward trend can be detected. This means we would compute $\sigma_{phv} = 0$, despite having high levels of downward volatility. One way to account for this corner case, is to take an additional look at σ_{adda}. In fact, for these extreme cases σ_{adda} would be close to 1, hinting that computing $\sigma_{phv} = 0$ is such a misspecification of the model. In such cases, where $\sigma_{adda} \approx 1$ and $\sigma_{phv} \approx 0$, we re-compute σ_{phv} by assuming a downward trend in a window slice when q_{max} occurs at least once before q_{min}.

2.6.4 Penalty-based channel quality metrics

As already mentioned, we compute risk-adjusted channel qualities by penalizing channel qualities according to their downward volatility. For each proposed downward volatility metric, we introdude a penalized channel quality metric. For a time slot i, we have :

$$q_{adda}(i) = q_{aggr}(i) - \sigma_{adda}(i)$$
$$q_{dsd}(i) = q_{aggr}(i) - \sigma_{dsd}(i)$$
$$q_{phv}(i) = q_{aggr}(i) - \sigma_{phv}(i)$$

The way these quality metrics are constructed intends to penalize channels proportionally to recent downward volatility levels. Hence, stable channels are barely penalized, while fluctuating channels suffer higher penalties.

2.7 Assessment of quality metrics

In this section, we introduce three metrics to assess the proposed channel quality metrics and penalization schemes. These are the downward outlier estimation error Σ_e, the percentage of downward outliers Σ_p and the root mean squared error Σ_{rmse}. Here, we interpret the resulting channel qualities in each time slot i delivered by a metric m as the estimation of the *raw* channel quality to be observed in the next time slot.

2.7.1 Downward Outlier Estimation Error Σ_e

Given a channel quality metric q_m with $m \in \{aggr, adda, dsd, phv\}$ and the associated channel quality $q_m(i-1)$ observed in time slot $(i-1)$, we define the Downward Outlier Estimation Error Σ_e as the sum of the squared residuals $(q_m(i-1) - q_{cbt}(i))^2$ when $q_m(i-1) > q_{cbt}(i)$. This means that Σ_e only takes into consideration channel quality measurements that fall below their estimates. Given s_{mw} the size of the moving window and n_{slot} the total number of slots in an experiment, we have:

$$\Sigma_e = \sqrt{\frac{1}{n_{slot} - (s_{mw}+1)} \cdot \sum_{j=s_{mw}+1}^{n_{slot}} (\phi_e(i))^2}$$

where

$$\phi_e(i) = \begin{cases} q_m(i-1) - q_{cbt}(i) & \text{if } q_m(i-1) > q_{cbt}(i) \\ 0 & \text{else} \end{cases}$$

2.7.2 Percentage of Downward Outliers Σ_p

Whereas with Σ_e we compute the estimation error, i.e. the sum of the distances of channel qualities q_m delivered by a given metric m to the observed downward outliers, with Σ_p we compute the percentage of q_{cbt} measurements that lie below their estimation. Given s_{mw} the size of the moving window and n_{slot} the total number of slots in an experiment, we have:

$$\Sigma_p = \frac{1}{n_{slot} - (s_{mw} + 1)} \cdot \sum_{j=s_{mw}+1}^{n_{slot}} \phi_p(i)$$

where

$$\phi_p(i) = \begin{cases} 1 & \text{if } q_m(i-1) > q_{cbt}(i) \\ 0 & \text{else} \end{cases}$$

2.7.3 Root Mean squared error Σ_{rmse}

Finally, we compute the within-sample root mean squared error, here denoted as Σ_{rmse}. This metric takes all q_{cbt} measurements into account and computes hence the total estimation error which Σ_e computes a portion of. Given s_{mw} the size of the moving window and n_{slot} the total number of slots in an experiment, we have:

$$\Sigma_{rmse} = \sqrt{\frac{1}{n_{slot} - (s_{mw} + 1)} \cdot \sum_{j=s_{mw}+1}^{n_{slot}} (q_m(i-1) - q_{cbt}(i))^2}$$

2.7.4 Assessment

As seen in Tab. 2.4, q_{dsd} displays the smallest estimation error in the experiments shown in Fig. 2.35 and Fig. 2.36 and is hence the most conservative metric with respect to Σ_e. In addition, if we look at the percentage of downward outliers (see Tab. 2.5), we can see that q_{dsd} is most of the time the most conservative of the penalization schemes. In fact, q_{dsd} achieves Σ_e values approximately half of those delivered by q_{aggr} and displays 20% less downward outliers than q_{aggr}, showing a significant improvement in both respects. On the other hand, q_{dsd} is still almost as accurate as q_{aggr}, displaying an increase of only 7% in Σ_{rmse}.

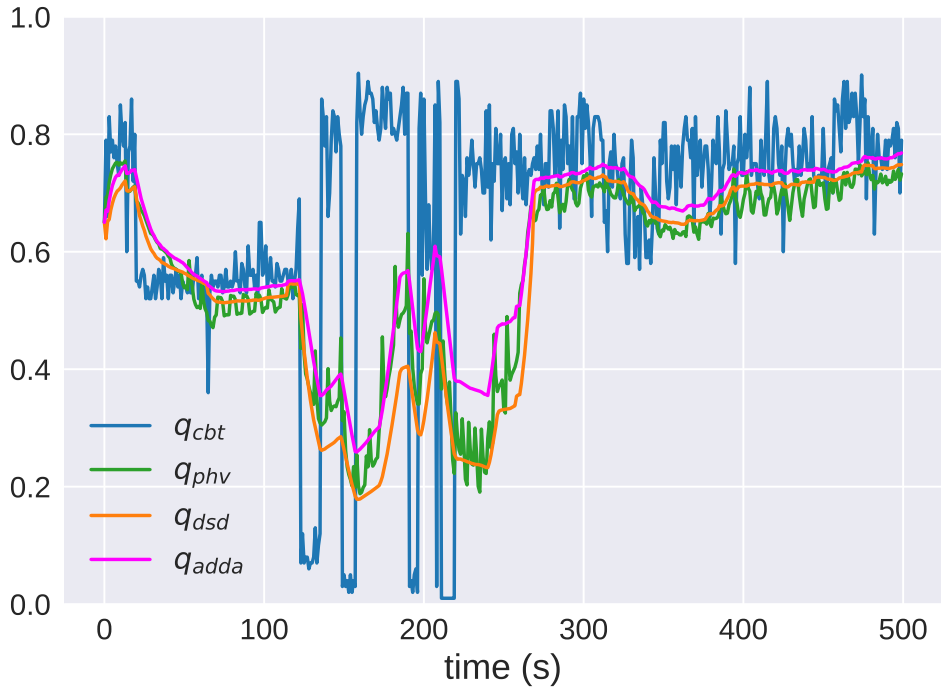


Figure 2.35: Comparison of the three penalized qualities and the raw quality.

$\mathbf{\Sigma_e}$	Fig. 2.35	Fig. 2.36
q_{aggr}	0.16	0.079
q_{adda}	0.12	0.056
q_{phv}	0.1	0.055
q_{dsd}	0.09	0.038

Table 2.4: Comparison of estimation error Σ_e on channel quality data shown in Fig. 2.35 and Fig. 2.36.

$\mathbf{\Sigma_p}$	Fig. 2.35	Fig. 2.36
q_{aggr}	0.42	0.33
q_{adda}	0.3	0.22
q_{phv}	0.18	0.16
q_{dsd}	0.22	0.13

Table 2.5: Comparison of percentage of downward outliers Σ_p on channel quality data shown in Fig. 2.35 and Fig. 2.36.

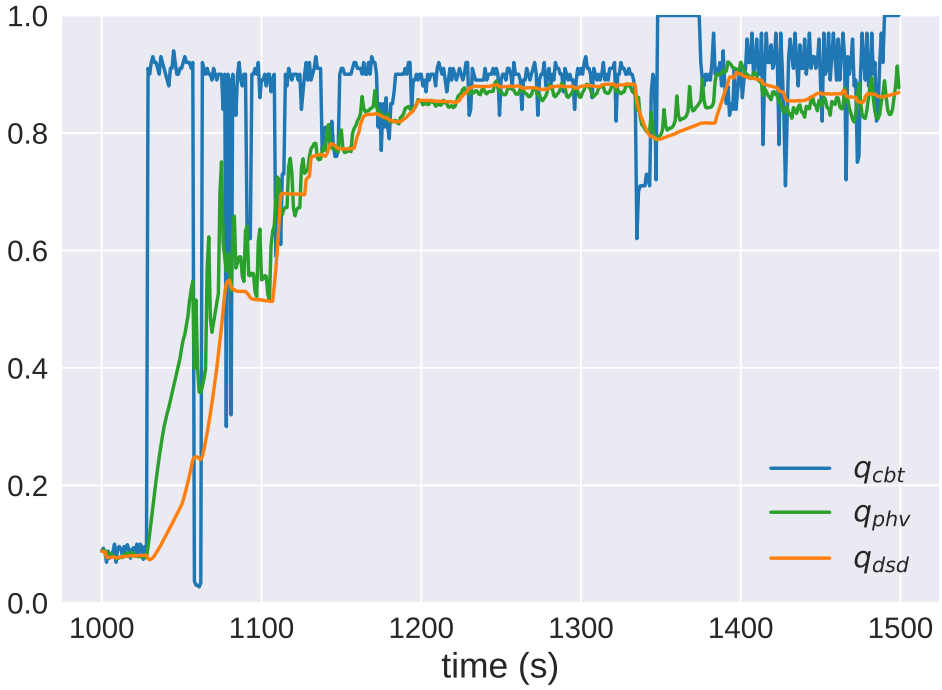


Figure 2.36: Comparison of q_{cbt}, q_{dsd} and q_{phv}.

$\mathbf{\Sigma_{rmse}}$	Fig. 2.35	Fig. 2.36
q_{aggr}	0.2	0.12
q_{adda}	0.23	0.15
q_{phv}	0.25	0.16
q_{dsd}	0.27	0.19

Table 2.6: Comparison of root mean squared error Σ_{rmse} of data shown in Fig. 2.35 and Fig. 2.36.

2.7.5 Comparison of penalization schemes

As expected, all penalization schemes show a similar behavior in the regions of low downward volatility with a larger divergence happening where significant quality drops take place (see Fig. 2.35). As shown in Fig. 2.37, q_{adda} is the most lenient penalization scheme and we can see that between 170 and 230 seconds it barely penalizes q_{aggr}, even in face of a significant drop in raw quality. On the other extreme, q_{dsd} penalizes the most for most cases, but displays less variation than q_{phv}. Furthermore, both q_{phv} and q_{dsd} penalize minimally in regions of low downward volatility (see Fig. 2.36, between 1200 and 1300 seconds). According to our requirements, q_{dsd} comes out as the strongest quality

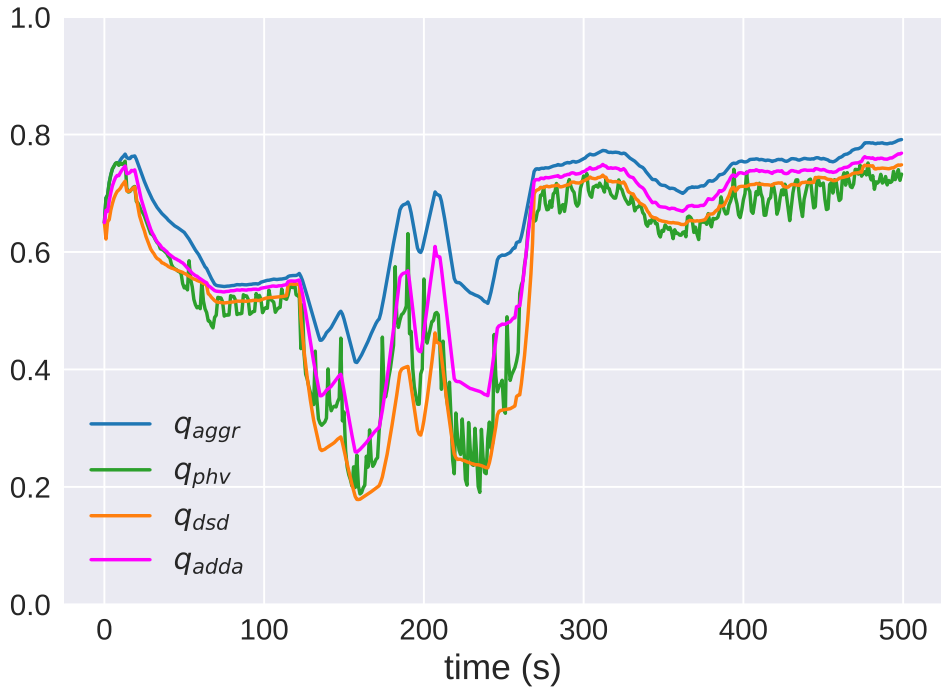


Figure 2.37: Comparison of the three penalized qualities with the aggregated quality.

metric, since it displays good adaptivity to significant changes in channel quality, is in general more conservative than q_{phv} and q_{adda} and more stable than q_{phv}. Furthermore, it is simpler than q_{phv}, i.e. has less hyper-parameters to tune (such as the number of window slices) and needs less computation, e.g. no linear regression.

2.8 Additional channel quality metrics

In this section, we investigate alternative channel quality metrics and combinations thereof. For our purposes, we model a channel as having a center frequency and a set of all available links that transmit on this center frequency. Therefore, we can further assess the quality of a channel by *passively* gauging the quality of all its links. Hence, our alternative quality metrics all rely on successful frame reception and decoding. Even though frame decoding alone is too restrictive, since it is e.g. blind to all other foreign protocols, by combining these frame-derived properties, we can still add information to our quality assessment of the available channels that we do not have with q_{cbt}.

In fact, measured quality derived from frame overhearing depends, among other things, on the placement of senders relative to the sensing node and the signal strength of the sender, similarly to the metrics derived from q_{cbt}. Moreover, by adding channel properties derived from 802.11 frames we can skew the channel quality towards interference of other 802.11 networks. This makes sense, since these networks can interfere not only in the physical layer through energy emissions when in sensing or interference range, but also on additional layers when in communication range.

2.8.1 Signal-to-Noise Ratio - SNR

The first link metric to take into consideration is the signal-to-noise ratio (SNR). This property indicates the relation between the power of the received signal P_{signal} and the estimated background noise level P_{noise}:

$$\mathrm{SNR}_{\mathrm{dB}} = 10 \log_{10} \left(\frac{P_{\mathrm{signal}}}{P_{\mathrm{noise}}} \right) = P_{\mathrm{signal,dB}} - P_{\mathrm{noise,dB}} \tag{2.4}$$

P_{signal} and P_{noise} are voltages across the same impedance and $P_{signal,dB}$ and $P_{noise,dB}$ respectively the same values but in the logarithmic decibels scale.

Multiple models in the literature have shown that QoS metrics such as Bit Error Rate (BER) or packet delivery ratio (PDR) correlate strongly with SNR. BER is the ratio of bits received with error to the total number of received bits and PDR the ratio of packets received without error to the total number of sent packets. [Gol05], for instance, derives the delivery probability of a signal based on the SNR and shows that for the DPSK wireless modulation, which is used in 802.11b, the probability of a bit error P_b can be computed as:

$$P_b = \frac{1}{2 \cdot \overline{SNR}}$$

where $\overline{SNR}$ is the average SNR on the channel. In fact, as shown throughout the literature for multiple channel models, the capacity of a channel, i.e. the data rate at which data can be sent with negligible P_b and hence minimum delay is a function of the bandwidth of the channel and of the SNR [Gol05]. In Additive White Gaussian Noise (AWGN) channels, for example, the capacity C of a channel with bandwidth B is given by:

$$C = B \cdot log_2(1 + SNR)$$

Similarly to what we have done for q_{cbt}, in order to summarize multiple observations into a single quality value and to have a channel quality metric (based on SNR) with appropriate stability, we need an aggregation method. This time though, we are going to aggregate raw properties, derived on a frame by frame basis, and plug these values into an SNR-based quality metric. We will make use of following frame-derived properties: the signal strength $P_{signal,int}$ of frames delivered by internal nodes, the received signal strength $P_{signal,ext}$ of foreign frames and the noise floor level P_{noise}, all measured in dBm. Given the signal strength P_m where $m \in \{(signal, int), (signal, ext), noise\}$, we aggregate multiple observations of P_m as:

$$P^\star_m(i) = \begin{cases} \overline{P_m(i)} & \text{if } i = n_{mw} \\ \alpha \cdot \overline{P_m(i)} + (1 - \alpha) \cdot P^\star_m(i-1) & \text{if } i > n_{mw} \\ undefined & \text{else} \end{cases}$$

where α is the weight of the latest measurement, n_{mw} the number of slots inside a moving window and $\overline{P_m(i)}$ the arithmetic mean of all observations of P_m in the current moving window up to time slot i.

This way, we obtain $P^\star_{signal,int}$, $P^\star_{signal,ext}$ and $P^\star_{noise}$ based on the received frames during each slot and the past history of received frames.

To define a channel quality based on SNR, we first introduce the notion of a *foreign traffic aware SNR*, denoted by SNR_{fta}. Given $P^\star_{signal,int}$, $P^\star_{signal,ext}$ and $P^\star_{noise}$, we define:

$$SNR_{fta} = P^\star_{signal,int} - (P^\star_{noise} + P^\star_{signal,ext}) \tag{2.5}$$

With the foreign traffic aware SNR, foreign 802.11 signals are interpreted as a further noise component, since these transmissions while decodable are mostly harmful to the

transmissions of internal nodes. In contrast, higher received signal strength of internal neighbors contributes to a higher SNR_{fta}.

An important phenomenon that highlights the importance of SNR_{fta} for channel quality assessment is the capture effect, i.e. when two or more signals collide but only the weaker signals are discarded (see Fig. 2.38). This means that larger signal amplitudes of foreign frames on a given channel might make the reception of weaker signals from internal nodes more difficult or even completely suppress them. Hence, the quality of all affected internal links degrades and with it the quality of the channel. Note that q_{cbt} while able to detect the occupation of the spectrum does not differentiate between stronger or weaker occupations, except indirectly by detecting the near-far effect.

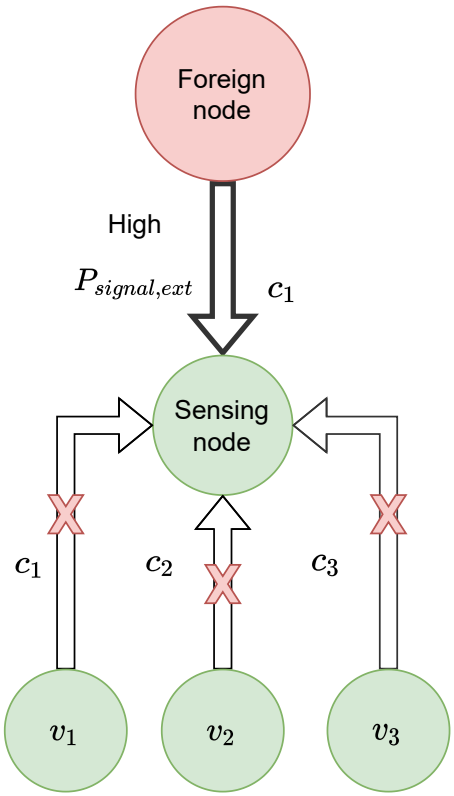


Figure 2.38: Foreign node with high $P_{signal,ext}$ prevents successful transmissions on internal links due to collisions or CCA. The foreign signal can still be detected by the sensing node due to the capture effect.

Example 2.8.1. To better illustrate the motivation of treating foreign 802.11 signal as noise, let us look at following scenario (see Fig.2.38): the signal strength $P_{signal,ext}$ is derived of frames sent by a foreign node (in red) and received by a sensing node (in green) such that $\forall i \in \{1,2,3\} : P_{signal,ext} > P^i_{signal,int}$, where $P^i_{signal,int}$ is the signal

strength received by the sensing node from its internal neighbors v_i on channel c_i. Here, the foreign node has two possibilities to thwart the transmissions of the internal nodes. First, this foreign node is a hidden station to the internal neighbors of the sensing node and prevents transmissions on internal links from being successful due to collisions. However, the foreign signal can still be detected by the sensing node due to the capture effect. Second, the foreign node is in sensing range of the internal nodes, but they are hidden stations to the foreign node. This time, transmissions of the foreign node will trigger CCA back-offs at the internal nodes, but not the other way around also leading to collisions. Moreover, the CCA back-offs occur not only on channel c_1, in use by the foreign node, but also on its adjacent channels, c_2 and c_3.

Given the minimum and maximum possible SNR_{fta} values for a given chipset, respectively $SNR_{fta,min}$ and $SNR_{fta,max}$, and given $q'_{snr} = \frac{SNR_{fta} - SNR_{fta,min}}{SNR_{fta,max} - SNR_{fta,min}}$, we define the foreign traffic aware SNR-based quality metric q_{snr} as

$$q_{snr} = 1.2 \cdot \frac{q'_{snr}}{q'_{snr} + 0.2} \tag{2.6}$$

Typical values for the frame-derived signal strength (e.g. for Atheros chips) lie in the range $-95\,dBm < P_{signal} \leq -35\,dBm$ and typical noise floor values lie in the range $-98\,dBm < P_{noise} \leq -90\,dBm$. A signal strength of $-25\,dBm$ is equivalent to measuring the output power directly on the transmitting antenna and $-35\,dBm$ is equivalent to having a transmitting node right next to the sensing node. On the other hand, $-95\,dBm$ is a very common receive sensitivity for a 802.11 NIC, i.e. the minimum possible signal strength that can be received by the chipset. In this case, a signal strength of e.g. $-100\,dBm$ would indicate 0% of signal strength and would therefore never be measured, since a frame has to be received with some signal strength for any measurement to take place.

Given the mentioned typical signal strength values, we estimate $SNR_{fta,min}$ and $SNR_{fta,max}$ with the following examples:

Example 2.8.2. ($SNR_{fta,min}$) Consider the internal signal strength at its worst $P^{\star}_{signal,int} = -95\,dBm$, a foreign signal strength $P^{\star}_{signal,ext} = -35\,dBm$ at its strongest level and a noise floor $P^{\star}_{noise} = -90\,dBm$. We then have:

$$SNR_{fta,min} = -95 + 35 + 90 = 30\ dBm$$

Example 2.8.3. ($SNR_{fta,max}$) Consider the internal signal strength at its best $P^{\star}_{signal,int} = 35\,dBm$, a spurious foreign signal strength $P^{\star}_{signal,ext} = -95\,dBm$ and noise floor $P^{\star}_{noise} = -98\,dBm$. We then have:

$$SNR_{fta,max} = -35 + 95 + 98 = 158\,dBm$$

This way, we have $SNR_{fta,min} = 30$ dBm and $SNR_{fta,max} = 158$ dBm, yielding (see Fig. 2.39):

$$q_{snr} = 1.2 \cdot \frac{\frac{1}{128} \cdot (SNR_{fta} - 30)}{\frac{1}{128} \cdot (SNR_{fta} - 30) + 0.2} = 1.2 \cdot \frac{SNR_{fta} - 30}{SNR_{fta} - 4.4} \tag{2.7}$$

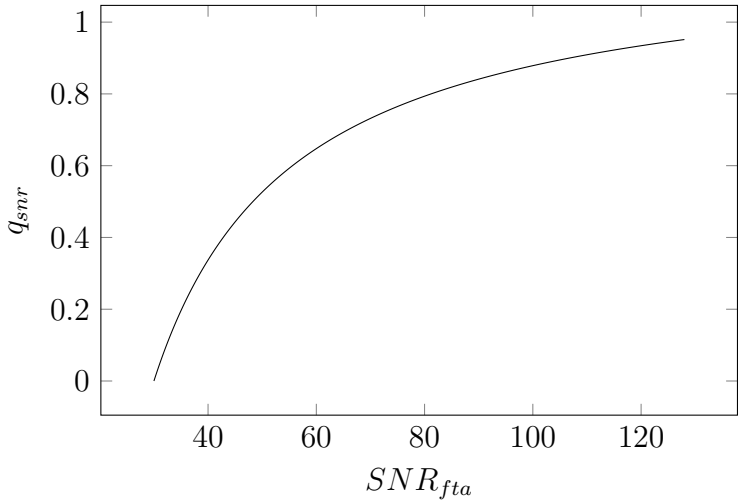


Figure 2.39: q_{snr} for $30\,dBm \leq SNR_{fta} \leq 158\,dBm$.

In situations where internal nodes are too far away from the sensing node or during the bootstrapping phase of the network, it is possible to receive foreign frames for one or more moving windows during which no internal frames are received, which means there are no measurements to derive $P^{\star}_{signal,int}$ from. For this case, we can adopt a conservative estimate such as $P^{\star}_{signal,int} = -95\,dBm$. The same estimate can also be used for $P^{\star}_{signal,ext}$ when no foreign frames are received.

Upon each frame reception, P_{signal} can be derived from the received signal strength indicator (RSSI) or the channel state information (CSI) and many NICs also deliver P_{noise}.

2.8.2 Received Signal Strength Indicator - RSSI

The energy received by a node during the transmission of the PLCP header is mapped to a Received Signal Strength Indicator [FVR07]. To be exact, RSSI is measured from the start of frame delimiter (SFD) until the end of the PCLP header error check (HEC) [80205]. This means RSSI is not computed for the full frame, just for its preamble.

RSSI is often delivered in a negative dBm scale or as dimensionless values that can be converted to this scale. [HHSW10] points out that RSSI alone can be an unreliable indicator of transmission performance. RSSI computation may be miscalibrated (hardware imprecision) and its values may be corrupted due to interference. Another argument against RSSI is that 802.11n OFDM and MIMO use multiple independent sub-channels to send different bits of the same data payload, each one with its own SNR. The RSSI-inferred SNR is thus an average for all received bits. Moreover, computing SNR values requires noise measurements (derived from frame receptions) and these values are not available to all network cards, e.g. Ralink network cards used for some of our tests do not support it. Even so, RSSI has been used throughout the literature for both performance estimation and distance gauging between nodes, as done in [ARW07].

RSSI can be retrieved through the *radiotap* interface. Device drivers that implement this interface, make both RSSI and antenna noise measurements available after frame reception through an appended *virtual* header with following fields:

- `IEEE80211_RADIOTAP_DBM_ANTSIGNAL`: an 8-bit field containing the received signal strength in decibels difference from 1mW.
- `IEEE80211_RADIOTAP_DBM_ANTNOISE`: an 8-bit field indicating the noise power, also in decibels difference from 1mW.

2.8.3 Channel State Information - CSI

An alternative to RSSI is the Channel State Information (CSI). This metric offers a higher level of granularity in OFDM transmissions by capturing the received signal strength for *each* sub-carrier of a given channel and also provides the phase information of these carriers. In OFDM, multiple narrow-band sub-carriers are used to transmit data in parallel instead of using a single wide-band channel. One can imagine these OFDM sub-carriers as sub-channels, i.e. components of a given channel. 802.11n/ac 20 MHz channels, for instance, are composed of 64 sub-carriers, of which 52 can be used for data transmission. Similarly to RSSI, every 802.11n/ac network card measures the channel

state information for each received frame only during its preamble. Even though, 802.11 standards name this property Channel State Information, strictly speaking, it is gained on a link by link basis and hence a Link State Information.

In [HHSW10], CSI measurements obtained from commodity 802.11n NICs were used to accurately predict the successful delivery of packets. Using power and noise levels measured at the receiver, they computed SNR values from which the highest usable rate in the channel as well as error rates for different modulations were predicted.

Even though CSI seems promising for link state estimation, its implementation is still experimental. [HHSW10], for instance, used a customized version of Intel's closed source Wi-Fi firmware for the Intel 5300 NIC. Firmware modifications allowed them to record CSI data, pass it to the network driver and later access it through a user-space program. Another work [XLL15] implemented a `Atheros-CSI-Tool` for CSI derivation on the Atheros ath9k driver. However, their code requires nodes to run a customized Linux Kernel 4.1.10 and one of the communicating nodes to be in 802.11n Access Point(AP) mode which is not practical in an ad-hoc scenario. In addition, for CSI to be derived, both communicating nodes have to use HT packets, since this information is only available under OFDM. Hence, even though CSI delivers a more detailed picture of SNR values when compared to RSSI, all its current restrictions make it difficult to use it as a flexible channel quality metric. Furthermore, since the SNR values must be aggregated over multiple time slots, it is rather unclear what real benefits are to be gained by having finer granularity with SNR values of sub-channels of a given channel, since this higher accuracy might lead to higher volatility as well.

2.8.4 Node degree

Another metric that strongly correlates with the performance of the communication in a network is the node degree. The node degree of a node n is the number of links from n to other nodes. In our case, even though it would be ideal to identify interference and sensing links, we can only detect communication links through frame decoding, and therefore only consider node degrees w.r.t. communication links.

Here we make a distinction between internal and external node degrees, depending respectively on whether the detected nodes are internal nodes or external ones. Whereas a higher internal node degree improves the connectivity and consequently the communication within a network, in special in an ad-hoc network, where each neighbor is a potential router, a higher external node degree increases the probability of a higher level of interference on the afflicted node. If these external nodes are 802.11 devices in

managed mode we have an automatic increase in baseline interference by adding more nodes, since APs send beacons and possibly other management frames periodically (independently of stations activity) and client devices perform active scanning, broadcasting probe requests which prompts receiving APs to reply with probe responses. Moreover, other competing technologies also foresee special packets for discoverability, such as Bluetooth. A Bluetooth base station in *scan mode* periodically broadcasts *discovery* packets on all available channels. Upon reception of these packets, all *discoverable* devices reply to identify themselves. Each scan inquiry runs for at least 10.24s seconds in order to discover all devices in communication range [HH09].

In general, a higher external node degree implies a higher level of *future* foreign spectrum occupation. This is a direct consequence of the fact that more nodes on a channel increase the probability of this channel being occupied at any given moment as well as the probability of simultaneous transmissions taking place on this channel. The amount of simultaneous transmissions from external nodes on the same channel as a node n influences the SNR (signal to noise ratio) observed by n and the SNR observed by other nodes on neighbor channels.

The authors in [HMCV07] have evaluated node degree distributions of Wi-Fi-based networks placed in densely populated areas and observed that they follow an exponential decay. The authors evaluated seven urban areas in the United States, one of which has more than 48,000 routers and have observed similar results throughout all analyzed areas. The observed node degrees all lie between 10 and 24, with the maximal node degree going as high as 154 for a maximum signal range of 45 meters. It is therefore realistic to expect a lower node degree for less densely populated scenarios.

In an ad-hoc network with dynamic membership, i.e. nodes leave or join the network at non-scheduled points in time, one can at best try to estimate the node's degree on a given channel. One property that is easy to obtain through frame decoding is the number of nodes that were seen to be communicating on a channel in a given time interval. For this, we count the number of seen MAC addresses in the last d_{seen} seconds.

2.8.4.1 Channel quality metric

If we assume nodes operating on $|C|$ available channels under random channel hopping sequences and continuously transmitting, we can derive the probability P_{hop} of more than one node picking the same channel at the same time. Given n_c channels and n_{seen} nodes we have:

$$P_{hop} = 1 - \left(1 - \frac{1}{|C|}\right)^{n_{seen}}$$

For instance, if we have $|C| = 13$ and say $n_{seen} = 20$ we have $P_{hop} = 0,798$.

On the other hand, we can formulate the probability P_{send} that any of the n_{seen} nodes transmits on any given slot:

$$P_{send} = 1 - (1 - \tau)^{n_{seen}}$$

where τ is the channel access probability. Considering both events (channel hopping and transmission) as independent, we can then compute the probability of the channel being *idle* as:

$$q_{degree} = (1 - P_{hop}) \cdot (1 - P_{send}) = \left(\left(1 - \frac{1}{|C|}\right) \cdot (1 - \tau)\right)^{n_{seen}} \tag{2.8}$$

We can use this probability as a channel quality metric, whose best value $q_{degree} = 1$ happens for $n_{seen} = 0$. For instance, if we have $|C| = 13$, $\tau = 2.5\%$ and $n_{seen} = 1$, we have $q_{degree} = \left(\frac{11.7}{13}\right)^1 = 0.9$. In Fig. 2.40, we can see the overall behavior of q_{degree} under the mentioned conditions with increasing values for n_{seen}.

Moreover, instead of assuming a fixed channel usage probability for all nodes, we can further optimize q_{degree} by estimating $\tau(n)$ as the average channel airtime of every neighbor node n after n_{slot} slots, i.e.

$$\tau(n) = \frac{\sum_{i=1}^{n_{slot}} d_{send}(n, i)}{n_{slot} \cdot d_{slot}}$$

With this we arrive at the following formula for q_{degree}:

$$q_{degree} = \left(1 - \frac{1}{|C|}\right)^{n_{seen}} \cdot \prod_{n=1}^{n_{seen}} (1 - \tau(n)) \tag{2.9}$$

It is worth noting that many 802.11 nodes in managed mode rarely switch their operating channel. Another possible optimization to q_{degree} would be to compare the received MAC addresses across all channels and to lower the probability of channel switching for those nodes we never see on a different channel. In the worst case, when no nodes are ever observed to perform channel hopping, we would have $q_{degree} = 1 - P_{send}$. In the general case, we propose using an exponential decay for P_{hop} such that it can be written as:

$$P_{hop}(t) = P_{hop}(0) \cdot e^{\frac{-t}{\tau}} = \left(1 - \frac{1}{|C|}\right) \cdot e^{\frac{-t}{\tau}}$$

where t is the number of slots since we last saw any channel switching from node n and τ the time point at which the probability of channel switching decays to $1/e \cdot P_{hop}(0) \approx 0.3679 \cdot \left(1 - \frac{1}{|C|}\right)$. In addition, each time we observe node n on a different channel than its last observed operation channel, we bump up its P_{hop} back to $1 - \frac{1}{|C|}$.

2.8.4.2 Protection against MAC spoofing

A valid concern in using MAC addresses to detect unique neighbors is the real possibility of MAC spoofing, i.e. nodes might use MAC addresses of other nodes in their vicinity. One relatively simple way of detecting such spoofing cases is to look into the sequence numbers in the MAC header of the received 802.11 frames. This MAC field is a value modulo 4096 that should be monotonically increasing for a sequence of frames. For instance, if a frame f_1 with sequence number s_1 is received after frame f_2 with sequence number s_2 such that $s_1 < s_2$ and another frame was already received before f_2 with s_1 then f_1 must be a duplicate frame or has originated at a different node with the same MAC address. Such a method is proposed by [GcC05], where detecting inconsistencies in the sequence number advance leads to detection of duplicated MAC addresses. Note that this type of detection involves keeping track of payloads of previously received frames on a node by node basis which incurs some overhead. A possible optimization here would be to use a probabilistic membership data structure such as Bloom filters where only the hashes of the received payloads have to be stored instead of the payloads themselves. This increases however the chance of false positives. An alternative approach to detect MAC spoofing that can also be combined with the sequence number-based method is identifying the distribution of RSSI values of received frames, since nodes at different distances to the sensing node will produce distinct distributions of received signal strength. In [STC+08], authors were able to detect MAC spoofing with help of only received signal strength and obtained a false positive rate of only 3% and a detection rate of 73.4% with RSSI statistics computed locally. By combining the statistics measured by multiple nodes they were able to detect 97.8% of duplicated MAC addresses.

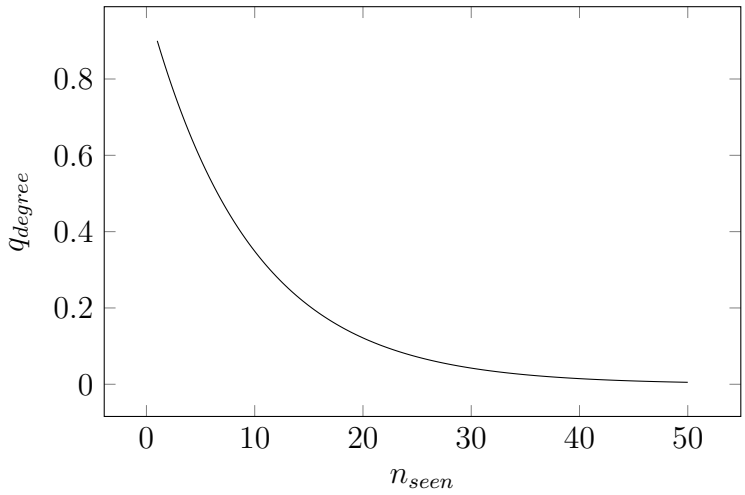


Figure 2.40: q_{degree} for $|C| = 13$ and $\tau = 2.5\%$ with varying n_{seen}.

2.8.5 Optimization of frame overhearing

Besides only detecting link properties of nodes using the same technology, deriving channel quality from frame overhearing has one significant weakness: processing overhead. One of the most promising technologies to minimize this overhead is the eXpress Data Path (XDP). This Linux kernel feature was introduced in 2014 and merged into the Linux kernel in 2016 and has already been adopted by big companies such as Facebook and Cloudflare [VCP+20]. XDP is the lowest layer of the Linux network stack (see Fig. 2.41) and allows network monitoring and packet processing to happen close to the NIC (in the driver of a network device), even before any memory allocation is done by the operating system. For this, eXpress Data Path uses eBPF, another Linux kernel feature, composed of a set of instructions and an in-kernel virtual machine (VM). XDP exposes network hooks to which eBPF programs can be attached and these are run in the eBPF VM. For instance, a node counting external neighbors by MAC address with XDP is able to retrieve only the needed information and then dropping this frame (the eBPF program returns with action `XDP_DROP`) without it ever touching any of the higher layers. Furthermore, XDP does not require any special hardware, but requires driver support for achieving best performance.

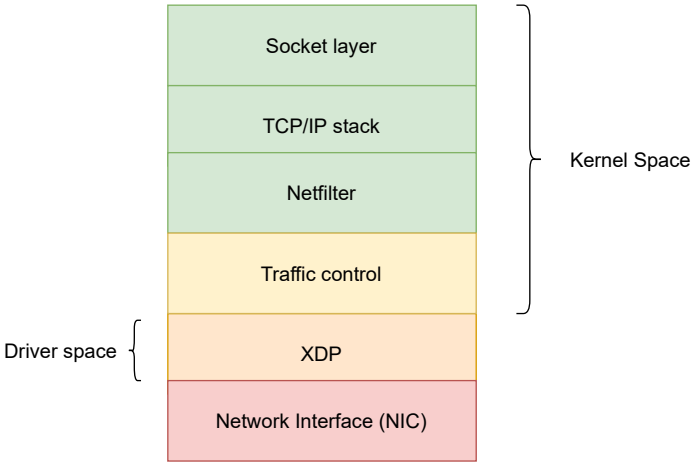


Figure 2.41: Layers of the Linux network stack.

2.9 Combining channel quality metrics

In this section, we motivate and formalize the combination of different channel quality metrics into a single value. As pointed out by the British Statistician George Box in his famous article *Science and Statistics* [Box76] *"all models are wrong"*, because by nature they not only simplify things but also usually focus on certain properties in lieu of others. This does not mean however that they cannot be useful for different applications. The same applies to our proposed channel quality metrics.

Wireless communications involve many different working components and are subject to multiple complex phenomena, such as random interactions due to back-off mechanisms, the near-far effect, the capture effect and fading. Small-scale fading, for instance, is the phenomenon of quick variations of both signal amplitude and phase for short distances (on the order of a few wavelengths) and short periods of time (within a few seconds). These variations can amount to 30 to 40 dB by changing the spacing between transmitter and receiver by a fraction of the signal's wavelength, which in 802.11 networks is on the order of a few meters. This type of fading, which defines how amplitude and phase of the radio signal varies, depends on the relations between a plethora of variables such as the symbol period, the bandwidth, delay and Doppler spread and the coherence time [Gra16], which is further dependent on the chosen 802.11 standard and its modulation scheme. It is therefore unrealistic to expect one channel quality metric alone to model and predict the behavior of all these channel properties. However, to benefit the most from a *combined* channel quality metric, its components should display

a minimum degree of separability by focusing on different properties, causal threads or different scales of similar properties.

Take q_{cbt}, for example, which strongly correlates with achievable throughput and is a solid metric to evaluate channel quality, but still completely ignores the floor noise and the amplitude of received signals, which as already mentioned in Sec. 2.8.1, do correlate with the successful delivery of frames and BER. Therefore, even though q_{cbt} can be seen as our main quality metric, it would be interesting to also take the mentioned frame-based metrics into consideration to achieve a holistic view of the channel. Moreover, in order to compute a communication schedule following the approach of [Eng20], where the number of slots assigned to each channel is proportional to its quality, we need a *single* value to represent the channel quality of each evaluated channel. Finally, we are less interested in the values themselves yielded by each channel quality metric, than concerned with the relation between these values and the ability of ranking channels based on multiple relevant channel properties.

Example 2.9.1. Let us imagine, for instance, two channels c_1 and c_2 with the same $q_{cbt} = 0.6$, but with differing q_{snr} values, respectively $q_{snr}^{c_1} = 0.8$ and $q_{snr}^{c_2} = 0.3$. It is clear in this case that we should prefer channel c_1 over channel c_2, since even though both channels potentially provide the same achievable throughput, it is more probable for a node on c_1 to have successful transmissions with the available bandwidth, leading to less end-to-end delay and fewer re-transmissions. Hence, combining q_{cbt} and q_{snr} assures that we choose c_1 over c_2 by yielding a higher channel quality for c_1.

With this in mind, we propose combining our proposed channel quality metrics q_{cbt}, q_{snr}, q_{degree} (or a subset thereof) while leaving the door open for adding any further proposed metrics. This combination is to be achieved with help of a normalized exponential transformation, namely the *softmax* function S_{max}. The softmax operation determines how much of every metric goes into the final combined metric with weights proportional to the value yielded by each metric.

Let $\vec{z}$ be a vector, the *softmax* $S_{max}(\vec{z})$ function generates a weight for every value z_i in $\vec{z}$ such that $z_i \in [0, 1]$, and the sum of all weights is always 1.

$$S_{max}(\vec{z}) = \begin{bmatrix} \omega_1 & \omega_2 & \omega_3 \dots \omega_n \end{bmatrix} \tag{2.10}$$

where

$$\omega_i = \frac{e^{\frac{z_i}{\tau}}}{\sum_{k=1}^{|z|} e^{\frac{z_k}{\tau}}}$$

τ is a temperature parameter and allows us to define how much of the greatest of the mixed values goes into the resulting metric. For a low temperature $\tau \to 0^+$, the weight of the highest values tends to 1.

Given channel quality metrics q_{m_1}, q_{m_2}, ..., q_{m_n}, we then define the combined channel quality metric $q_{m_1,m_2,...,m_n}$ as the following matrix product:

$$q_{m_1,m_2,...,q_{m_n}} = S_{max} \left(\begin{bmatrix} q_{m_1} \\ q_{m_2} \\ q_{m_3} \\ \dots \\ q_{m_n} \end{bmatrix} \right) \cdot \begin{bmatrix} q_{m_1} \\ q_{m_2} \\ q_{m_3} \\ \dots \\ q_{m_n} \end{bmatrix} = \sum_{i=1}^{n} \omega_i \cdot q_{m_i}$$

It is worth noticing that by definition the combined metric is also in the range $[0, 1]$, since:

$$\begin{aligned} \forall i \in [0,1] : 0 \leq w_i \leq 1 \wedge 0 \leq q_{m_i} \leq 1 \implies & 0 \leq w_i \cdot q_{m_i} \leq w_i \\ \implies & 0 \leq \sum_{i=1}^{n} \omega_i \cdot q_{m_i} \leq \sum_{i=1}^{n} w_i \end{aligned}$$

$$\sum_{i=1}^{n} \omega_i = 1 \implies 0 \leq \sum_{i=1}^{n} \omega_i \cdot q_{m_i} \leq 1$$

This means, any chosen combination of channel quality metrics yields a new combined channel quality metric. We can, for instance, combine q_{cbt}, q_{snr} and q_{degree} yielding $q_{cbt,snr,degree}$:

$$q_{cbt,snr,degree} = S_{max} \left(\begin{bmatrix} q_{cbt} \\ q_{snr} \\ q_{degree} \end{bmatrix} \right) \cdot \begin{bmatrix} q_{cbt} \\ q_{snr} \\ q_{degree} \end{bmatrix}$$

Example 2.9.2. Given $q_{cbt} = 0.9$, $q_{snr} = 0.65$ and $q_{degree} = 0.5$ and $\tau = 0.9$, we compute $q_{cbt,snr,degree}$ as:

$$S_{max} \left(\begin{bmatrix} 0.9 \\ 0.65 \\ 0.5 \end{bmatrix} \right) = \begin{bmatrix} 0.4 & 0.32 & 0.28 \end{bmatrix}$$

$$q_{cbt,snr,degree} = \begin{bmatrix} 0.4 & 0.32 & 0.28 \end{bmatrix} \cdot \begin{bmatrix} 0.9 \\ 0.65 \\ 0.5 \end{bmatrix} = 0.708$$

Example 2.9.3. Now, let us take our previous example and keep the same quality values for q_{cbt} and q_{degree}, but let us assume an improvement in q_{snr}, because e.g. a foreign node moved further away from our sensing node. Given $q_{cbt} = 0.9$, $q_{snr} = 0.8$ and $q_{degree} = 0.5$ and $\tau = 0.9$, we compute $q_{cbt,snr,degree}$ as:

$$S_{max} \left(\begin{bmatrix} 0.9 \\ 0.8 \\ 0.5 \end{bmatrix} \right) = \begin{bmatrix} 0.38 & 0.35 & 0.27 \end{bmatrix}$$

$$q_{cbt,snr,degree} = \begin{bmatrix} 0.38 & 0.35 & 0.27 \end{bmatrix} \cdot \begin{bmatrix} 0.9 \\ 0.8 \\ 0.5 \end{bmatrix} = 0.758$$

As expected the improvement in q_{snr} led to an improvement in $q_{cbt,snr,degree}$.

In Fig. 2.42, we show q_{cbt}, q_{snr} and $q_{cbt,snr}$ computed for a duration of more than 25 hours and in Fig. 2.43 we show the first 3 hours of the same experiment. As seen in both Fig. 2.42 and Fig. 2.43, the combined quality has values lying between the values of the original qualities and incorporates the oscillations of both q_{cbt} and q_{snr}, such as in the first 1.5 hour (strongly following q_{cbt}) and after 25 hours with a drop in quality due to a worsening in q_{snr}.

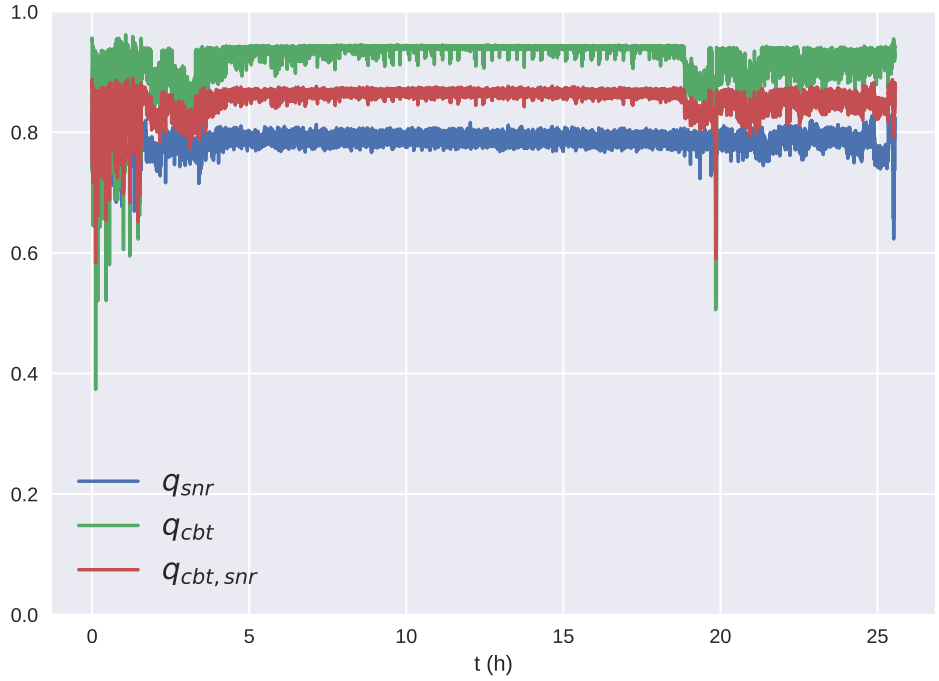


Figure 2.42: Comparison of q_{cbt}, q_{snr} and $q_{cbt,snr}$ computed for more than 25 hours.

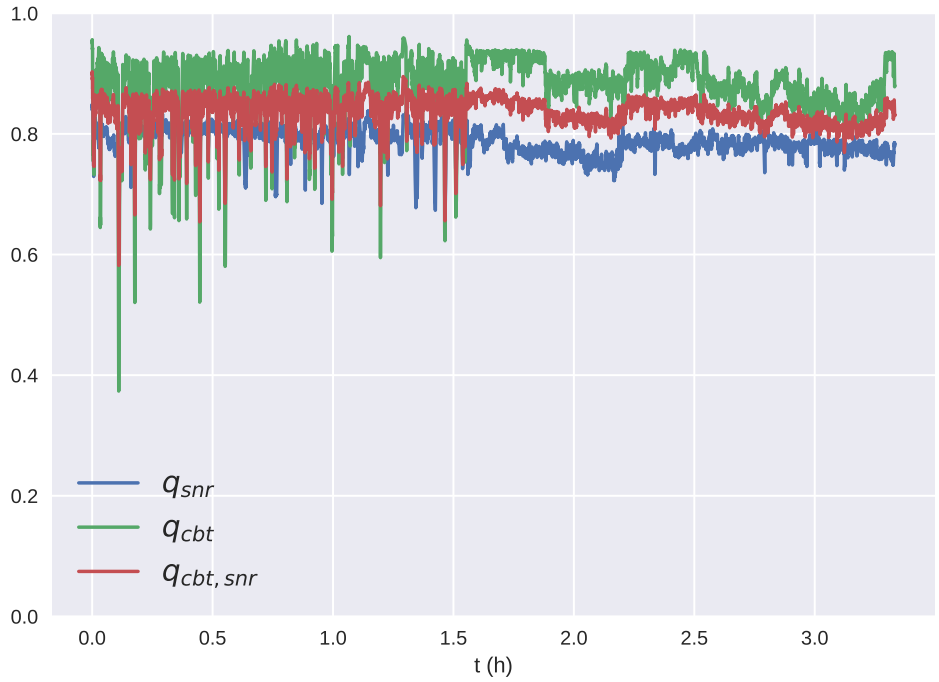


Figure 2.43: Comparison of q_{cbt}, q_{snr} and $q_{cbt,snr}$ computed for the first 3 hours of the experiment showing in Fig. 2.42

2.10 Summary

In this chapter, we have dealt with different aspects regarding the formalization, implementation and evaluation of different channel quality metrics in 802.11 based networks.

In Sec. 2.1, we formalized the concept of channel quality and have introduced our main channel quality metric, q_{cbt}. In Sec. 2.2, we then discussed multiple aspects of implementing q_{cbt} on a testbed equipped with commodity 802.11 hardware.

As a motivation for our research and to better contextualize our design decisions, we have shown in Sec. 2.3.3 typical channel selections both around the world and locally, using the campus of the TUK as an example. These selections are usually constrained to channels 1, 6, 11 in the 2.4 GHz band or channels 36, 40 and 44 in the 5 GHz band, which leads to an even sharper overcrowding of the ISM spectrum. Later in Sec. 2.3, we have discussed the results of testbed-based experiments carried on multiple channels in both 802.11 frequency bands with help of an in-house traffic generator and an external tool (iPerf3). In Sec. 2.3.4, these experiments have shown that q_{cbt} strongly correlates with achievable throughput and is a solid choice as a channel quality metric. Afterwards, in Sec. 2.3.5, we also showed how q_{cbt} is able to detect transmissions on adjacent or even non-adjacent channels due to the near-far effect. However, using raw values of q_{cbt} has some downsides, which we have improved upon. First, in Sec. 2.4, we have introduced appropriate corrections to q_{cbt} to be used by every active node in order to deduct the effects of the node's transmission on the metric, not only on the current channel but also on all further affected channels. Second, in Sec. 2.5, we have introduced an aggregation method based on an EWMA combined with a moving average to keep our channel quality metric both stable and adaptive to significant changes in channel conditions. Finally, in Sec. 2.6, we have made q_{cbt} volatility-aware by penalizing the qualities of channels with higher levels of downward volatility. In Sec. 2.7, these downward volatility levels were gauged by three proposed metrics of which the downward standard deviation showed the best performance.

In Sec. 2.8, we have introduced two additional quality metrics, q_{snr} and q_{degree}. These quality metrics rely on frame overhearing to derive metrics to assess channel conditions, such as SNR and the node degree, i.e. number of neighbor nodes currently active on a channel. In order to account for the near-far effect and the capture effect, we have defined q_{snr} based on the introduced notion of a foreign-traffic aware SNR (SNR_{fta}), in which the aggregated signal strength of foreign nodes is considered a further noise component.

Finally, in Sec. 2.9, we have argued the benefits of combining different channel quality metrics to assess the quality of a channel based on multiple properties that have a certain degree of separability, e.g. the spectrum occupation and the foreign traffic aware SNR. The proposed combination uses the softmax operation and is quite flexible, allowing the addition of further quality metrics. To better illustrate the behavior of one such combination, we have carried out experiments and compared the combination $q_{cbt,snr}$ to the individual metrics q_{cbt} and q_{snr} over a period of more than 24 hours.

What makes a problem a problem is not that a large amount of search is required for its solution, but that a large amount would be required if a requisite level of intelligence were not applied.

— Allen Newell and Herbert A. Simon

3

Sensing schedules

Contents

In Chap. 2, we have introduced a holistic approach for gathering channel quality information in ad-hoc 802.11 networks, which is then converted into a communication schedule. This schedule tells nodes at the beginning of each slot which channel to switch to, in order to communicate with peers that comply with the same communication schedule. In addition, nodes autonomously derive a sensing schedule based on their current communication schedule, which determines in which order the available channels should be sensed.

In this chapter, we will go into depth on the motivation behind using sensing schedules and describe our solution to synthesize high-quality sensing schedules based on different criteria. In Section 3.1, we discuss different approaches to determining channel sensing order, compare these to our proposed approach and informally state our main objectives when constructing sensing schedules. In Section 3.2, a geometrical model for channel overlap is introduced and in Section 3.3 different optimization metrics are defined accompanied by the description of the basic algorithm for optimizing these metrics, yielding high-quality channel sensing schedules. Section 3.4 analyses the performance of the introduced heuristics by comparing the quality of delivered sensing schedules to global optima. Section 3.5 introduces different variants of our basic heuristics using an iterated local search approach coupled with randomness to improve local minima and compares the performance of these variants against global optima. In Section 3.6, we introduce some relevant properties and constraints associated with our solution to the optimization problem at hand. Section 3.7 describes the problem of constructing conflict-minimal sensing schedules from a graph theoretical standpoint, refers to an exact method from the literature to solve it, the Hungarian Method, and compares the performance of our iterated-local-search-based heuristic against it. Finally, Section 3.8 summarizes the chapter.

3.1 Channel sensing order

As already hinted at, sensing schedules dictate the channel sensing order, i.e. the order in which channels are sensed. Most techniques proposed in the literature for computing optimal or sub-optimal channel sensing orders are devised for cognitive radio networks, and use approaches based on a single transceiver. These approaches divide the time into slots with duration T, and at each slot nodes sense multiple channels following a given hopping sequence. Each channel $c \in C$ appears once in the sensing sequence and is sensed for a fixed duration τ such that $\tau \cdot |C| < T$. Moreover, the transceiver used for

sensing is the same transceiver used for communication, which means that at any given moment only one of these activities can take place. In fact, the sensing phase always takes place at the beginning of each slot, and is followed by a transmission phase where the node can become active on a channel which it detected as idle (see Fig. 3.1). If no channel was deemed idle, nodes stay silent for the remaining duration of the slot.

In [CZ11], for instance, channels are sensed in descending order of achievable transmission rate and nodes stop sensing as soon as a an idle channel is detected. This channel can then be used for communication. Initially, a sensing order might be randomly chosen from the space of all possible permutations of $\{c_1, c_2, \ldots, c_{|C|}\}$ and the first time a channel is sensed as idle a communicating pair or nodes exchange probe frames with a training sequence in order to estimate the channel gain of the given link (directly associated with the signal attenuation of the link) and with it estimate the achievable transmission rate on the channel.

This approach has three main disadvantages:

1. By having a single transceiver, channel sensing and communication compete for transceiver time, yielding an overall loss in airtime, since all slots have a minimum silent period τ, regardless of channel conditions.

2. In addition, the view delivered by the channel sensing is incomplete and potentially outdated, since channels are measured with different frequencies, leading to a biased use of the channels, which in an extreme case results in sensing starvation for some channels, i.e. multiple channels are never sensed and their channel conditions are never updated, potentially wasting bandwidth.

3. After detecting channel c_k as idle, the approach used in [CZ11] assumes a channel stays unused by other nodes for the remaining duration $T - k \cdot \tau$ of the slot. However, this quality assessment is quite naive and easily disrupted by e.g. periodic foreign traffic whose period of transmission does not align with the channel sensing order.

Other works have proposed alternative sensing order construction schemes focusing on sensing order dispersion [KLDLa13, XWW+15, LLG+17], i.e. trying to minimize the conflicts in sensing order among peers to minimize the number of frame collisions on any given channel. These collisions happen when nodes sense the same channel as idle at the same time and try to simultaneously transmit on this channel. [KLDLa13], for instance, proposes a γ-persistent strategy that attempts to create collision-free sensing orders, i.e. neighbor nodes do not sense the same channels at the same time and consequently

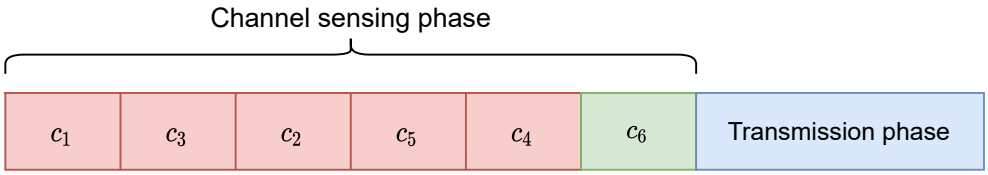


Figure 3.1: Example of sequential channel sensing with 6 channels, where channels $c_1 \dots c_5$ were sensed as busy, but channel c_6 was sensed as idle and is hence used in the transmission phase for communication.

never use the same channels for communication. The proposed algorithm takes into consideration the history of collisions on the sensed channels in order to update the probability of sticking with the current sensing order or picking a new one. Different from our approach though, γ-persistent sensing orders are not constructed at run-time, but rather pre-computed and randomly chosen from a Latin Square, i.e. a $|C|$ by $|C|$ matrix in which every channel $c \in C$ appears exactly once in each column and each row. This γ-persistent strategy does not consider minimizing adjacent channel interference and only tries to eliminate conflicts where the same channels are used. In addition, it relies on missing ACKs to detect sensing order conflicts among peers and eventually change its sensing sequence. This is a somewhat pessimistic approach and can lead to unnecessary sensing order changes, since ACK losses might include a large number of false positives w.r.t. sensing order conflicts. In fact, due to the lossy nature of the medium and depending on the used wireless technology, frame losses might occur due to multiple factors such as interference by foreign traffic.

Despite some clear similarities with the described sensing order schemes, our problem and solution for it present some fundamental differences. Different from the previously shown cognitive radio approaches, our implementation decouples channel sensing from communication by using two radio transceivers and separate schedules, such that a sensing schedule does not compete for transceiver time with a communication schedule (see Fig. 3.2). Furthermore, in our approach, high-quality sensing schedules should deliver an unbiased broader view of channel conditions, and minimize overlaps with the communication schedules. In summary, our main objectives are:

1. To construct balanced sensing schedules, in which channels are sensed equally often to minimize channel sensing bias.

2. To minimize channel overlap between sensing and communication schedules such that traffic from peers will produce the least possible disturbance to channel quality measurements.

3. If possible, distribute remaining channel overlap evenly in order to maximize channel overlap fairness. This intends to give each channel similar chances of being correctly sensed.

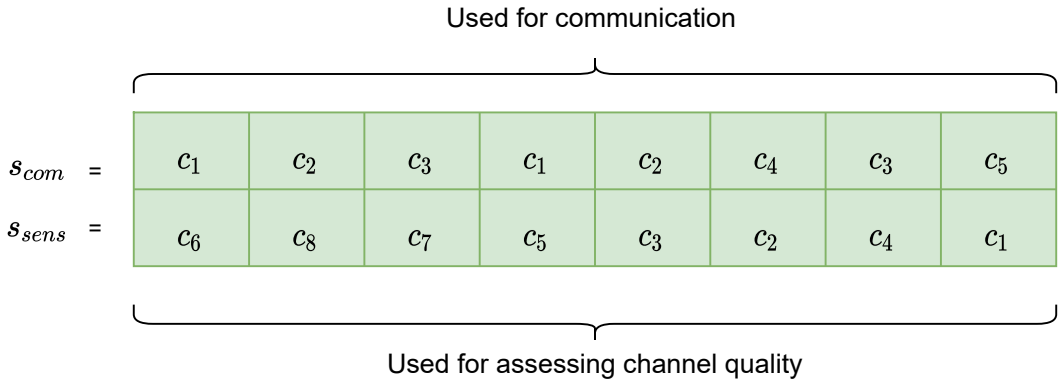


Figure 3.2: In our approach, a communication schedule s_{com} is used for hopping to communication channels (with one transceiver) and a sensing schedule s_{sens} (with another transceiver) is used for hopping to channels to be sensed.

3.2 Geometrical model for channel overlap

In order to minimize the channel overlap between the communication and the sensing schedule, we first need to define how this channel overlap is modeled and calculated. In this section, we introduce a geometrical model that allows us to calculate the amount of channel overlap between two channel hopping sequences in an efficient manner. Wireless communication technologies in general divide the wireless spectrum in frequency ranges called channels. These channels have some fundamental properties associated with them such as the bandwidth (the extension of the frequency range) and the center frequency (usually calculated as the arithmetic or as the geometric mean of the upper and lower passband cutoff frequencies). Typically, the physical layer of a wireless network protocol has a bandplan that defines how each frequency band is to be used for communication w.r.t. transmit power distribution across the frequency space. This distribution of power is defined by a transmit spectrum mask (also called a channel mask), which sets signal attenuation requirements (upperbounds) at given frequency offsets (see Fig. 3.3). Similar to the nominal channel bandwidth in 802.11 standards, FCC regulations define a channel bandwidth as the 20-dB bandwidth, i.e. the bandwidth at which the signal is attenuated by 20 dB relative to the power peak (measured at the center frequency) [LSW+20].

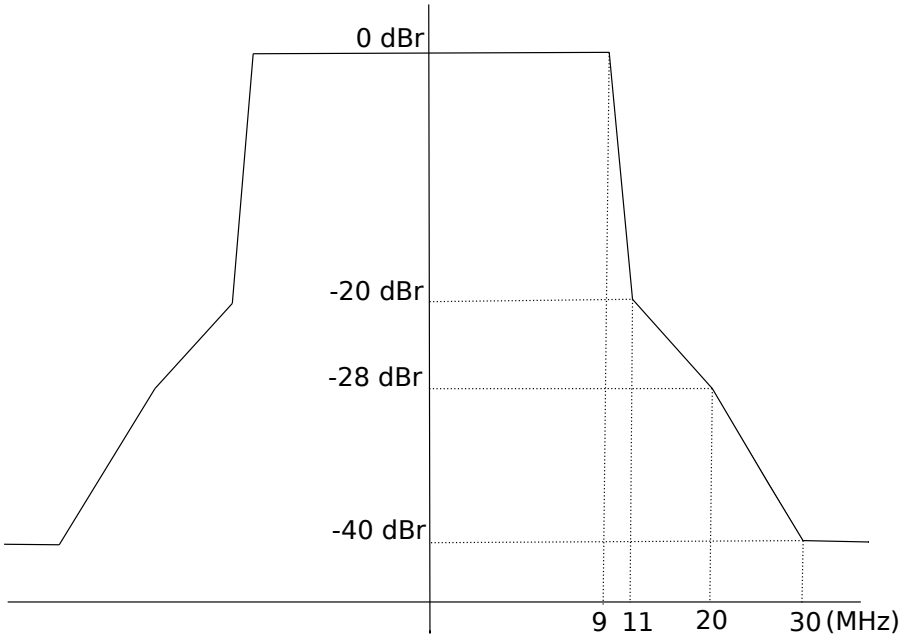


Figure 3.3: Illustration of a channel mask in 802.11 for the 2.4 GHz band. Attenuation requirements are given in dBr (decibel relative to the power spectral density peak in the signal)

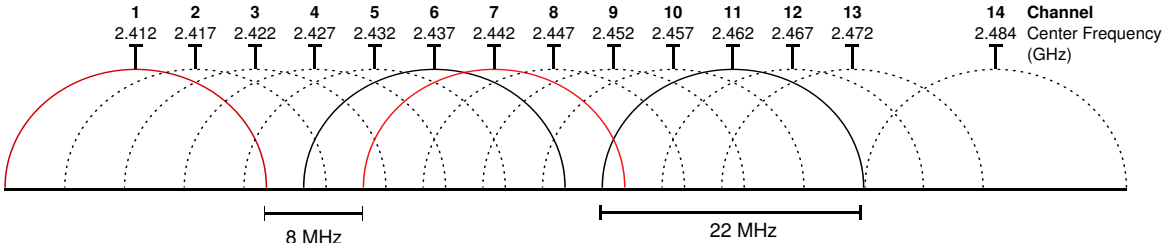


Figure 3.4: Illustraton of 802.11 channels in 2.4 GHz band.

It is clear from this definition that even though most of the power of the modulated wireless signal is contained within the channel bandwidth, there is still a portion of the total power that bleeds over to neighboring frequencies causing additional overlap between channels that nominally should not have any. If an overlap occurs between channels c_i and c_j, radiation transmitted on c_i lies in the passband of c_j and will be passed on mostly unattenuated to the demodulator of a node listening on channel c_j. Figure 3.4 shows the channels in the 2.4 GHz band and makes it evident that there is a lot of overlap in this frequency band (which is even worse if we take the whole transmit mask into consideration). It is interesting to note that consecutive channels

do not always have the same frequency spacing between them, e.g. channels 13 and 14 in the 2.4 GHz band have 12 MHz instead of 5 MHz between them. However, even though the frequency separation between consecutive center frequencies may vary, we assume that all channels that are within the same band and can be used at the same time have the same bandwidth (which is the case in 802.11 networks). In Figure 3.5, we

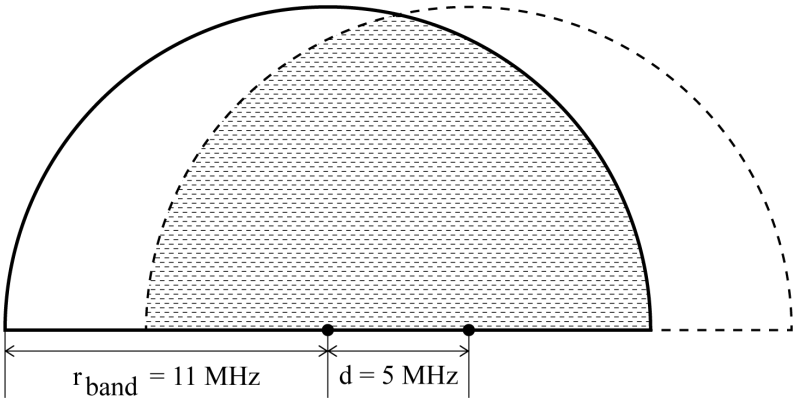


Figure 3.5: Channel overlap between two consecutive channels with 22 MHz of channel bandwidth.

display two consecutive channels c_i and c_j overlapping in the 2.4 GHz band, both with a bandwidth of 22 MHz and center frequency separation of 5 MHz. In our geometrical model we approximate the shape of a 802.11 channel (enveloped by the transmit mask) to a semicircle centered at the channel's center frequency. The 802.11b standard, for instance, considers its modulated signals to follow the shape of $\frac{\sin(x)}{x}$, which shape-wise is not far from a semicircle (see Fig. 3.6).

For the purpose of constructing high-quality sensing schedules, we consider that more than 99% of the energy of the modulated signal is *received* within this approximated shape, and disregard the additional components of the transmit mask. In fact, we have already introduced a correction model in Chap. 2 that corrects for the interference effect resulting from these additional spectrum mask components and from close proximity.

The approximation to a semicircle allows us to calculate the overlap between two channels by computing the overlapping area between two semicircles, which is a well-known closed formula.

Definition 3.2.1. Given a channel c_i and an overlapping channel c_j, both modeled as semicircles with a radius r_{band}, the overlap area $A_{overlap}(c_i, c_j)$ between c_i and c_j and the area $A_{band}(r_{band})$ of a semicircle with radius r_{band}, we define the overlap ratio $r_{overlap}(c_i, c_j)$ as

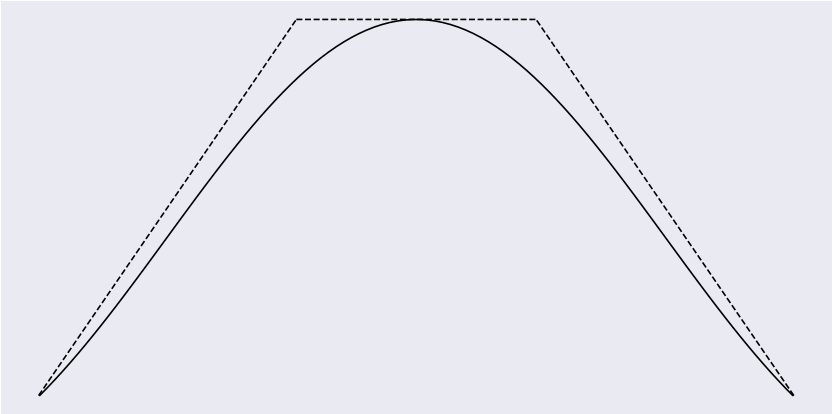


Figure 3.6: Illustration of $\frac{\sin(x)}{x}$ shape involved by a channel mask.

$$r_{overlap}(c_i, c_j) = \frac{A_{overlap}(c_i, c_j)}{A_{band}(r_{band})} \tag{3.1}$$

Given the center frequencies f_{c_i} and f_{c_j} of respectively c_i and c_j, the radius r_{band} of the overlapping semicircles, and the frequency spacing between the center frequencies $d = |f_{c_i} - f_{c_j}|$, we have

$$A_{band}(r_{band}) = \frac{\pi}{2} \cdot r_{band}^2 \tag{3.2}$$

$$A_{overlap}(c_i, c_j) = \begin{cases} 0 & d \geq 2 \cdot r_{band} \\ r_{band}^2 \cdot cos^{-1}\left(\frac{d}{2 \cdot r_{band}}\right) - \frac{d}{4} \cdot \sqrt{4 \cdot r_{band}^2 - d^2} & \text{otherwise} \end{cases} \tag{3.3}$$

Example 3.2.1. Given channel 13 and channel 14 under 802.11b the spacing between the center frequencies is 12 MHz and the radius $r_{band} = 11\ MHz$, yielding

$$A_{overlap}(c_i, c_j) = 121 \cdot 0.993\ MHz^2 - 3 \cdot \sqrt{484 - 144}\ MHz^2 \approx 64.94\ MHz^2$$

$$A_{band}(r_{band}) = \frac{\pi}{2} \cdot 121\ MHz^2 \approx 190.07\ MHz^2$$

$$r_{overlap}(c_i, c_j) = \frac{64.94\, MHz^2}{190.07\, MHz^2} \approx 0.34$$

Example 3.2.2. In 802.11b, between channels 2 and 6 we have $d = |f_2 - f_6| = 20\, MHz$ and $A_{band}(r_{band}) \approx 190.07\, MHz^2$ (computed in the previous example) leading to

$$A_{overlap}(c_i, c_j) = 121 \cdot 0.429\, MHz^2 - 5 \cdot \sqrt{484 - 400}\, MHz^2 \approx 6.17\, MHz^2$$

$$r_{overlap}(c_i, c_j) = \frac{6.17\, MHz^2}{190.07\, MHz^2} \approx 0.032$$

As shown in the previous examples, increasing the frequency separation between the center frequencies of the overlapping channels by a factor of 1.6 led to a tenfold reduction in the overlap ratio. Our geometrical model even though simplified approximates the nonlinear relation between frequency spacing and overlap ratio and delivers results cohesive with the underlying physical model. Moreover, it is worth noting that our algorithm for synthesizing high-quality sensing schedules works independently of the underlying channel overlap model. This means that in a scenario with another wireless technology where our main assumptions proved unsatisfactory, our geometrical model could be easily swapped for an alternative model.

A channel overlap model with some similarities to our geometrical overlap model was introduced in [MSBA06]. In their model, the overlap between two channels is computed by an interference factor, defined as the overlapping area between the transmitter's channel mask and the receiver's passband filter. This area is computed through the convolution of the mask and filter functions. This model while potentially more precise than our model requires not only the transmit spectral mask but also the bandpass filter function on the receiver side, which makes both the calculation itself and the implementation of this approach considerably more complex.

3.3 Construction of high-quality sensing schedules

In this section, we formalize the concept of balanced sensing schedules and define our main optimization metrics when constructing sensing schedules, i.e. the conflict metric and the overlap fairness. Both metrics take a pair of sensing and communication schedules as input. Moreover, we describe *opt_conflict*, a heuristic for synthesizing sensing schedules that are balanced, conflict-minimal and if possible that maximize overlap fairness. Also, we introduce opt_fairness, a variant heuristic that prioritizes overlap fairness

over conflict-minimality. We first introduced the basic ideas behind the construction algorithms for both heuristics in [AG20]. In this chapter, we will also introduce an extension to these basic construction algorithms that is able to improve local minima, achieving higher quality in the generated sensing schedules.

As already mentioned, we have a set of channels $C = \{c_1, c_2, \ldots, c_{n_c}\}$ that is fixed and known to every node, in which each channel has a dynamic channel quality. This quality is assessed in an order defined by a sensing schedule s_{sens} and each channel is used for communication in an order defined by a communication schedule s_{com}. The placement and the frequency of occurrence of these channels in the communication schedule is dependent on the channel qualities at the time the communication schedule is synthesized. In contrast, the placement of channels in s_{sens} depends on s_{com}.

3.3.1 Balanced sensing schedules

Here, we formalize the first of our main objectives when constructing sensing schedules, i.e. that they are balanced.

Definition 3.3.1. Given C a non-empty set of channels, s_{sens} a sensing schedule, $n(c, s_{sens})$ the number of occurrences of channel $c \in C$ in s_{sens}, and s_{sens} satisfying $\forall c \in C.\ n(c, s_{sens}) > 0$. The sensing schedule s_{sens} is *balanced w.r.t.* C iff $\forall c_1, c_2 \in C.\ n(c_1, s_{sens}) = n(c_2, s_{sens})$.

This means, a sensing schedule is balanced if and only if all channels are sensed equally often. This intends to minimize channel sensing bias by avoiding that some channels are sensed more often than others. Note that we still impose no constraints on the placement of channels in the sensing schedule, and that the frequency of occurrence of these channels in s_{sens} is independent of the channel qualities.

To derive a balanced sensing schedule we start with a balanced *seed sensing schedule* $s^{(0)}_{sens} = [c_1, \ldots, c_n]$, where every channel $c \in C$ occurs exactly once, i.e. $\forall c \in C.\ n(c, s^{(0)}_{sens}) = 1$ and $|s^{(0)}_{sens}| = |C|$.

3.3.2 Schedule conflicts

As already mentioned, any channel overlap between sensing and communication schedules is detrimental to the performance of our channel sensing scheme. Any time slots in which channel overlap occurs between both schedules are deemed as *conflicts*. These may occur as primary or secondary conflicts.

Definition 3.3.2. (Primary conflict). Given a communication schedule s_{com} and a sensing schedule s_{sens}, a primary conflict occurs in a slot i iff $s_{com}[i] = s_{sens}[i]$.

Channel hopping sequences map channels to sequences of successive time slots. Even though schedules are finite, this mapping repeats infinitely for an infinite sequence of slots (in practice for as long as nodes are in operation). Given a communication schedule s_{com} and a seed sensing schedule $s^{(0)}_{sens}$, we assume without loss of generality that the applications of these schedules are synchronized and start in the same time slot. If the communication schedule and the seed sensing schedule have the same number of slots, they will hence stay aligned in time for the whole duration of operation. The upside of this is that a reordering of $s^{(0)}_{sens}$ such that $\forall i \in [1\,,\, |s^{(0)}_{sens}|].\ s^{(0)}_{sens}[i] \neq s_{com}[i]$ eliminates all primary conflicts, i.e. all aligned slots in s_{com} and the reordered $s^{(0)}_{sens}$ have distinct channels.

Example 3.3.1. Given $C = \{\, c_1, c_2, \ldots, c_5 \,\}$, a communication schedule $s_{com} = [c_1, c_3, c_2, c_1, c_2]$ and $s^{(0)}_{sens} = [c_1, c_2, \ldots, c_5]$, we can reorder $s^{(0)}_{sens}$ obtaining a primary-conflict-free s_{sens} as shown below

$$
\begin{aligned}
s_{com} &= [c_1, \quad c_3, \quad c_2, \quad c_1, \quad c_2] \\
s_{sens} &= [c_4, \quad c_5, \quad c_3, \quad c_2, \quad c_1]
\end{aligned}
$$

However, if the sizes of the communication and seed sensing schedules differ, the alignment in time of s_{com} and $s^{(0)}_{sens}$ is lost, and hence permutation alone cannot deliver a primary-conflict-free sensing schedule w.r.t. s_{com}. Our solution to this dilemma is to modify the seed sensing schedule obtaining a new schedule that is still balanced, but that now aligns with s_{com} in time. For this purpose, we derive an auxiliary extended form of s_{com}, namely s'_{com} and an extended version of the seed sensing schedule, namely $s^{(0)'}_{sens}$ both with the same length. These extended forms are obtained by concatenating the original schedules with themselves until the least common multiple $lcm(|s_{com}|, |s^{(0)}_{sens}|)$ is reached. As $s^{(0)}_{sens}$ is balanced, if follows, by induction, that its concatenation with itself is still balanced.

Definition 3.3.3. (Schedule alignment). Given schedules s_1 and s_2, we obtain the *aligned schedules* s'_1 and s'_2 by concatenating s_1 and s_2 with themselves, such that $|s'_1| = |s'_2| = lcm(|s_1|, |s_2|)$.

Example 3.3.2. Given $C = \{ c_1, c_2, \ldots, c_8 \}$, $s_{com} = [c_1, c_2, c_3, c_1, c_4, c_2, c_1, c_3]$, and $s^{(0)}_{sens} = [c_1, c_2, c_3, c_4]$, we have $lcm(|s_{com}|, |s^{(0)}_{sens}|) = lcm(8, 4) = 8$, which yields the following aligned schedules s'_{com} and $s^{(0)'}_{sens}$:

$$s'_{com} = [c_1, \quad c_2, \quad c_3, \quad c_1, \quad c_4, \quad c_2, \quad c_1, \quad c_3]$$
$$s^{(0)'}_{sens} = [c_1, \quad c_2, \quad c_3, \quad c_4, \quad c_1, \quad c_2, \quad c_3, \quad c_4]$$

The construction of a sensing schedule free of primary conflicts is closely related to the k-coloring problem. Given a graph G, a k-coloring colors each vertex of G with one out of k colors in such a manner that no two adjacent vertices are colored the same. In our case, half of the graph (the communication schedule) is already colored and the remaining vertices (the sensing schedule) must receive each an appropriate color (a channel).

Definition 3.3.4. (Secondary conflict). Given a communication schedule s_{com} and a sensing schedule s_{sens}, a secondary conflict occurs in a slot i iff $s_{com}[i] \neq s_{sens}[i]$ and $r_{overlap}(s_{com}[i], s_{sens}[i]) > 0$.

Ideally, we would like to be even more strict and not only eliminate primary conflicts, but also secondary ones. And if this is not possible, we want to minimize the impact of the remaining conflicts on the q_{cbt}-based quality assessment.

This way, finding a conflict-free sensing schedule w.r.t. a given communication schedule can be better described by a generalization of the k-coloring problem, namely the *list* k-coloring problem. In this version of the problem, the coloring of the graph has to respect for every vertex v a list of allowed colors $L(v)$. As one might suspect, for a general graph, the list k-coloring problem is NP-complete for any integer $k \geq 3$. A considerable research effort has been put into solving this problem for restricted graph classes (most of them without cycles or paths) [BCM+18].

Moreover, the heuristics to construct high-quality sensing schedules we later introduce in this chapter are examples of local search heuristics. This class of algorithms has shown good performance when used for coloring graphs with fewer than 500 vertices [GHHP13]. Similarly to our algorithm, most heuristics that try to create a conflict-free coloring (or a partial sub-optimal coloring) start with an incomplete k-coloring where some of the adjacent vertices have color conflicts and in an iterative manner move the solution towards optimality through a sequence of improving color exchanges.

3.3.3 Conflict metric and overlap fairness

In this section, we formalize our last main objectives, i.e. that every sensing schedule should be conflict-minimal w.r.t. a communication schedule, and that it should try to maximize channel overlap fairness, without leading to the deterioration of the conflict-minimality.

Definition 3.3.5. (Conflict metric). Given aligned schedules s'_1 and s'_2, we define the *conflict metric* $\Sigma_{conflict}(s'_1, s'_2)$ as the sum of the overlap ratios for all channel pairs placed in the same position i, $1 \leq i \leq N$, where $N = |s'_1| = |s'_2|$, normalized to the interval $[0, 1]$:

$$\Sigma_{conflict}(s'_1, s'_2) = \frac{1}{N} \sum_{i=1}^{N} r_{overlap}(s'_1[i], s'_2[i]) \tag{3.4}$$

Definition 3.3.6. (Conflict-minimality). Given a sensing schedule s_{sens} aligned with a communication schedule s_{com} and $\mathbb{P}(s_{sens})$ the set of all permutations of s_{sens}, s_{sens} is conflict-minimal w.r.t. s_{com} iff

$$s_{sens} \in \underset{s \in \mathbb{P}(s_{sens})}{\operatorname{argmin}} \Sigma_{conflict}(s_{com}, s)$$

Definition 3.3.7. (Conflict-freedom). Given a sensing schedule s_{sens} and a communication schedule s_{com}, s_{sens} is *conflict-free* w.r.t. s_{com} iff $\Sigma_{conflict}(s_{com}, s_{sens}) = 0$.

With regards to the conflict metric, the worst possible sensing schedule has matching channels (with those of the communication schedule) aligned in all slots, i.e. all time slots exhibit primary conflicts, yielding $\Sigma_{conflict}(s_{com}, s_{sens}) = 1$. However, if secondary conflicts are present we have $0 < \Sigma_{conflict}(s_{com}, s_{sens}) < 1$.

Definition 3.3.8. (Total overlap). Given aligned schedules s'_1 and s'_2, we define the *total overlap* $L(s'_1, s'_2, c)$ *of channel* $c \in C$ as the sum of the overlap ratios of all slots in which c occurs in s'_2:

$$L(s'_1, s'_2, c) = \sum_{i \in \{\, k \in [1, |s'_2|] \,|\, s'_2[k] = c \,\}} r_{overlap}(s'_1[i], s'_2[i])$$

Definition 3.3.9. (Overlap fairness). Given aligned schedules s'_1 and s'_2 and the set of channels $C = \{ c_1, c_2, \ldots, c_{n_c} \}$, we define the *overlap fairness* $\phi(s'_1, s'_2, C)$, based on Jain's fairness index [JCH98]:

$$\phi(s'_1, s'_2, C) = \frac{(\sum_{i=1}^{n_c} L(s'_1, s'_2, c_i))^2}{n_c \cdot \sum_{i=1}^{n_c} L(s'_1, s'_2, c_i)^2} \tag{3.5}$$

This way, $\phi(s'_1, s'_2, C)$ values lie in the range $[0, 1]$ and measure how fair the distribution of the total overlap across all channels is, where $\phi(s'_1, s'_2, C) = 1$ indicates the fairest configuration possible.

3.3.4 Basic local search heuristics

In this section, we introduce two local search heuristics for constructing high-quality sensing schedules. Given a communication schedule s_{com} and given a set of channels C, our main heuristic *opt_ conflict* derives aligned communication and sensing schedules s'_{com} and s'_{sens}, such that the resulting sensing schedule is balanced w.r.t. C, conflict-minimal w.r.t. s'_{com}, and, if possible under these constraints, maximizes overlap fairness w.r.t. s'_{com}. In addition, we describe a variant heuristic, *opt_ fairness*, that also delivers balanced sensing schedules, but primarily optimizes for overlap fairness while secondarily attempting to minimize the conflict metric.

As already mentioned, our basic construction algorithm is a *local search-based heuristic*, in which an optimal or sub-optimal sensing schedule is constructed through permutations that progressively improve the chosen optimization criteria. In the case of opt_conflict this means primarily lowering the value of $\Sigma_{conflict}(s'_{com}, s'_{sens})$ while secondarily trying to increase $\phi(s'_{com}, s'_{sens}, C)$ until a local optimum is reached.

Definition 3.3.10. Given a schedule s and slots i and j with $i \neq j$, we define $f_{swap}(s, i, j)$ as the function that returns schedule $s^\star$, a permutation of s, in which the channels in positions i and j are swapped, i.e.

$$\forall k \in [1, |s|] : \; s^\star[k] = \begin{cases} s[k] & \text{if } k \neq i \wedge k \neq j \\ s[i] & \text{if } k = j \\ s[j] & \text{if } k = i \end{cases}$$

The main steps of opt_conflict are described in Alg. 1. As shown, a channel is only swapped with another if the resulting permutation either yields a lower conflict metric

than the current one or if the conflict metric stays the same but there is an improvement in overlap fairness.

Algorithm 1 Local search heuristic opt_conflict

function OPT_CONFLICT(s_{com}, C)
  $s^{(0)}_{sens} \leftarrow ConstructSeedSolution(C)$ ▷ construct balanced seed
  $s'_{com}, s^{(0)\prime}_{sens} \leftarrow AlignSchedules(s_{com}, s^{(0)}_{sens})$ ▷ ensure $|s'_{com}| = |s^{(0)\prime}_{sens}|$
  $l \leftarrow 1$
  $k \leftarrow 0$
  repeat
    $S_{swap} \leftarrow \{t | t \neq l \wedge t \in [1, |s'_{com}|]\}$
    Find slots k to swap with l such that we have *highest loss* in $\Sigma_{conflict}(s'_{com}, s^{(0)\prime}_{sens})$, i.e.

$$k \in \arg\min_{k \in S_{swap}} \Sigma_{conflict}(s'_{com}, f_{swap}(s^{(0)\prime}_{sens}, k, l))$$

    if there is one or more swap positions k **then**
      Pick k such that we have the highest gain in overlap fairness $\phi(s'_{com}, s^{(0)\prime}_{sens}, C)$:

$$k \in \arg\max_{k \in S_{swap}} \phi(s'_{com}, f_{swap}(s^{(0)\prime}_{sens}, k, l)), C)$$

    else
      Find the swap position k that yields the highest gain in the overlap fairness $\phi(s'_{com}, s'_{sens}, C)$, *without increasing* the value of $\Sigma_{conflict}(s'_{com}, s^{(0)\prime}_{sens})$.
    end if
    if no swap position k was found **then**
      $k \leftarrow l$
    else
      $s^{(0)\prime}_{sens} \leftarrow f_{swap}(s^{(0)\prime}_{sens}, k, l)$ ▷ Swap channels in slots l and k
      $l \leftarrow (l + 1) \mod |s'_{com}|$
    end if
  until $k = l$
  $s_{sens} \leftarrow s^{(0)\prime}_{sens}$ ▷ We found our sensing schedule
end function

Example 3.3.3. Given $C = \{1, 3, 5, 6, 7, 9, 10, 11, 13\}$, a subset of the channels available in 802.11 2.4 GHz band, a communication schedule $s_{com} = [10, 3, 5, 7, 3, 5]$ and a seed sensing schedule $s^{(0)}_{sens} = [1, 3, 5, 6, 7, 9, 10, 11, 13]$, the steps needed to construct a sensing schedule s'_{sens} balanced w.r.t. C and conflict-minimal w.r.t. s_{com} are shown below:

1. First the given schedules are aligned:

$$s'_{com} = [10,\ 3,\ 5, 7, 3, 5,\ 10,\ 3,\ \ 5,\ 7,\ 3, 5, 10,\ 3,\ 5,\ 7,\ \ 3,\ \ 5\ ]$$
$$s^{(0)'}_{sens} = [1,\ \ 3,\ \ 5,\ \ 6, 7, 9, 10, 11, 13, 1, 3, 5, 6,\ \ \ 7,\ 9, 10,\ 11,\ \ 13]$$

2. Then we perform a series of swaps to minimize the conflict metric (see Tab. 3.1). Swapped channels are shown in bold and following abbreviations are used:
   a) $result = 1$ when a primary conflict was solved
   b) $result = 2$ when a secondary conflict was solved
   c) $result = 3$ when $\Sigma_{conflict}(s'_{com}, s^{(0)'}_{sens})$ was lowered, but no conflicts were solved.

s'_{com}			
$[10, 3, 5, 7, 3, 5, 10, 3, 5, 7, 3, 5, 10, 3, 5, 7, 3, 5]$			
$s^{(0)'}_{sens}$	$\Sigma_{conflict}$	ϕ	result
$[1, 3, 5, 6, 7, 9, 10, 11, 13, 1, 3, 5, 6, 7, 9, 10, 11, 13]$	0.34	0.41	start
$[\mathbf{5}, 3, \mathbf{1}, 6, 7, 9, 10, 11, 13, 1, 3, 5, 6, 7, 9, 10, 11, 13]$	0.29	0.42	1
$[5, \mathbf{10}, 1, 6, 7, 9, \mathbf{3}, 11, 13, 1, 3, 5, 6, 7, 9, 10, 11, 13]$	0.18	0.43	1
$[5, 10, \mathbf{10}, 6, 7, 9, 3, 11, 13, 1, 3, 5, 6, 7, 9, \mathbf{1}, 11, 13]$	0.16	0.36	2
$[5, 10, 10, \mathbf{3}, 7, 9, 3, 11, 13, 1, \mathbf{6}, 5, 6, 7, 9, 1, 11, 13]$	0.081	0.22	3
$[5, 10, 10, 3, \mathbf{5}, 9, 3, 11, 13, 1, 6, \mathbf{7}, 6, 7, 9, 1, 11, 13]$	0.078	0.36	3
$[5, \mathbf{9}, 10, 3, 5, \mathbf{10}, 3, 11, 13, 1, 6, 7, 6, 7, 9, 1, 11, 13]$	0.077	0.35	2
$[5, 9, 10, 3, 5, 10, 3, \mathbf{7}, 13, 1, 6, \mathbf{11}, 6, 7, 9, 1, 11, 13]$	0.051	0.28	3
$[5, 9, 10, 3, \mathbf{6}, 10, 3, 7, 13, 1, 6, 11, \mathbf{5}, 7, 9, 1, 11, 13]$	0.037	0.17	3
$[5, 9, 10, 3, 6, 10, 3, 7, 13, 1, 6, 11, 5, 7, \mathbf{11}, 1, \mathbf{9}, 13]$	0.035	0.15	2

Table 3.1: Construction of a balanced conflict-minimal sensing schedule for $C = \{1, 3, 5, 6, 7, 9, 10, 11, 13\}$ and $s_{com} = [10, 3, 5, 7, 3, 5]$.

Note that in this example each channel swap lowered $\Sigma_{conflict}(s'_{com}, s^{(0)'}_{sens})$ and that we solved nearly all conflicts, reaching a very low conflict metric, i.e. $\Sigma_{conflict}(s'_{com}, s'_{sens}) = 0.035$. However, as this example shows, lowering the conflict metric is not always beneficial to the overlap fairness. In fact, it is very challenging to find short examples in which it is possible to increase the overlap fairness after reaching a local minimum for the conflict metric.

According to the algorithm described in Alg. 1, any pair of slots (l, k) whose channels $s^{(0)'}_{sens}[l]$ and $s^{(0)'}_{sens}[k]$ are to be swapped, complies with conditions (a), (b), (c) or (d):

(a) $r_{overlap}(s^{(0)'}_{sens}[l], s'_{com}[k]) = 0$ and $r_{overlap}(s^{(0)'}_{sens}[k], s'_{com}[l]) = 0$. This means, swapping $s^{(0)'}_{sens}[l]$ and $s^{(0)'}_{sens}[k]$ solves any existing conflict (primary or secondary) on both positions l and k.

(b) $r_{overlap}(s^{(0)'}_{sens}[k], s'_{com}[l]) = 0$ and $0 < r_{overlap}(s^{(0)'}_{sens}[l], s'_{com}[k]) < 1$. With this swap, we solve any conflict (primary or secondary) on position l while replacing an eventual primary conflict at position k by a secondary conflict. Alternatively, we might just improve a secondary conflict in k while l has no conflict and stays without any.

(c) $0 < r_{overlap}(s^{(0)'}_{sens}[l], s'_{com}[k]) < 1$, $0 < r_{overlap}(s^{(0)'}_{sens}[k], s'_{com}[l]) < 1$.
and $\Sigma_{conflict}(s'_{com}, f_{swap}(s^{(0)'}_{sens}, k, l)) < \Sigma_{conflict}(s'_{com}, s^{(0)'}_{sens})$, we improve the conflict metric, but no secondary conflict is eliminated.

(d) $\Sigma_{conflict}(s'_{com}, s^{(0)'}_{sens}) = \Sigma_{conflict}(s'_{com}, f_{swap}(s^{(0)'}_{sens}, k, l))$ and
$\phi(s'_{com}, f_{swap}(s^{(0)'}_{sens}, k, l), C) > \phi(s'_{com}, s^{(0)'}_{sens}, C)$. With this swap, we maintain the current value of the conflict metric (no conflict is improved), but increase the overlap fairness.

3.3.5 Prioritizing overlap fairness over conflict metric

As the previous example has already hinted at and as our next sections will show, conflict metric and overlap fairness seem to be conflicting objectives and only in few cases do we have a sensing and communication schedule that lead to optimal conflict metric and overlap fairness. Therefore, to better assess the extent of possible optimization for each optimization goal, we have implemented a variant heuristic, *opt_fairness* (see Alg. 2), in which we optimize primarily for the overlap fairness, only trying to minimize the conflict metric if it does not lower the overlap fairness.

Algorithm 2 Local search heuristic opt_fairness

function OPT_FAIRNESS(s_{com}, C)
  $s^{(0)}_{sens} \leftarrow ConstructSeedSolution(C)$ ▷ construct balanced seed
  $s'_{com}, s^{(0)'}_{sens} \leftarrow AlignSchedules(s_{com}, s^{(0)}_{sens})$ ▷ ensure $|s'_{com}| = |s^{(0)'}_{sens}|$
  $l \leftarrow 1$
  $k \leftarrow 0$
  repeat
    $S_{swap} \leftarrow \{t | t \neq l \wedge t \in [1, |s'_{com}]\}$
    Find slots k to swap with l such that we have the highest gain in overlap fairness $\phi(s'_{com}, s^{(0)'}_{sens}, C)$, i.e.

$$k \in \arg\max_{k \in S_{swap}} \phi(s'_{com}, f_{swap}(s^{(0)'}_{sens}, k, l)), C)$$

    if there is one or more swap positions k **then**
      Pick k such that we have the *highest loss* in $\Sigma_{conflict}(s'_{com}, s^{(0)'}_{sens})$, i.e.

$$k \in \arg\min_{k \in S_{swap}} \Sigma_{conflict}(s'_{com}, f_{swap}(s^{(0)'}_{sens}, k, l))$$

    else
      Find the swap position k that yields the *highest loss* in $\Sigma_{conflict}(s'_{com}, s^{(0)'}_{sens})$, *without decreasing* the value of the overlap fairness.
    end if
    if no swap position k was found **then**
      $k \leftarrow l$
    else
      $s^{(0)'}_{sens} \leftarrow f_{swap}(s^{(0)'}_{sens}, k, l)$ ▷ Swap channels in slots l and k
      $l \leftarrow (l + 1) \mod |s'_{com}|$
    end if
  until $k = l$
  $s_{sens} \leftarrow s^{(0)'}_{sens}$ ▷ We found our sensing schedule
end function

3.4 Performance assessment of heuristics

In this section, we will discuss and assess the performance of our local search heuristics for the construction of high-quality sensing schedules, namely opt_conflict and opt_fairness, comparing them against a complete enumeration approach. By complete enumeration we mean computing all possible permutations of a given seed sensing schedule in order to find an optimal schedule according to the chosen optimization metrics w.r.t. a given aligned communication schedule.

A naive approach to find high-quality sensing schedules would be to attempt a complete enumeration of the permutation space of $s^{(0)\prime}_{sens}$ every time a new communication schedule is received, which in the general case is not feasible in polynomial time, since the number of possible permutations grows combinatorially with the sizes of s'_{com} and $s^{(0)\prime}_{sens}$.

Example 3.4.1. To better illustrate the large search space associated with permutations, let us look at a small case with 14 channels (the number of Wi-Fi channels in the 2.4 GHz band) and a communication schedule with $|s_{com}| = 5$, which leads to $|s^{(0)\prime}_{sens}| = lcm(5, 14) = 60$. This means we would have to inspect

$$|\mathbb{P}(s^{(0)\prime}_{sens})| = \frac{60!}{5!^{14}} \approx 6.48 \cdot 10^{52}$$

distinct permutations. To put this number into perspective, the total number of possible IPv6 addresses is $3.4 \cdot 10^{38}$ and an estimated number of atoms comprising earth's total mass falls in the order of 10^{49}.

In fact, even small cases can take up to an hour or more to be completely enumerated in a sequential manner on commodity hardware and even the use of parallelism has its limits on providing a significant speedup. This not only highlights the need of heuristics to find near-optimal solutions in an efficient manner, but also poses a challenge when evaluating our heuristics against complete enumeration, since generating global optima (even offline) is quite time-intensive.

However, the major advantage of complete enumeration over any chosen heuristic is that given sufficient time it always constructs an optimal sensing schedule, e.g. a permutation of $s^{(0)\prime}_{sens}$ with minimal conflict metric value w.r.t. the communication schedule and best possible overlap fairness without increasing the value of the conflict metric. This will help us compare the quality of solutions delivered by our heuristics when compared to the global optima for each pair of communication and sensing schedules.

3.4.1 Terminology

At this point, it might be helpful for our further discussion to clarify some terms used in further sections regarding our optimization heuristics for constructing sensing schedules.

Definition 3.4.1. (Candidate sensing schedule). For a given seed sensing schedule $s^{(0)\prime}_{sens}$ constructed from a set of channels C and aligned with communication schedule s'_{com}, a *candidate sensing schedule* solution is defined as any permutation of $s^{(0)\prime}_{sens}$.

When clear from the context we will refer to candidate sensing schedule solutions simply as candidate solutions. The same holds for constructed sensing schedules which depending on the context will be just referred to as solutions.

As already mentioned, we have two optimization goals, namely minimal conflict metric and maximal overlap fairness. Each of these objective functions is used to quantify the goodness, i.e. *quality*, of a given candidate solution. Furthermore, we make the distinction between *local* and *global* optima.

Definition 3.4.2. (Global optimum). Given an optimization goal g, a *global optimum* is the solution with the maximum solution quality according to g.

This maximum solution quality might be obtained by minimizing or maximizing given optimization goals, which is respectively the case for the conflict metric and the overlap fairness.

Definition 3.4.3. (Neighborhood). Given a candidate solution s_i and a set of allowed moves that allow us to transition to another candidate solution s_j, we define the *neighborhood* of s_i as the set of all candidate solutions that are reachable from s_i in any number of valid moves.

For opt_conflict and opt_fairness valid moves are permutations of $s_{sens}^{(0)\prime}$ that improve either of the optimization metrics.

Definition 3.4.4. (Local optimum). Given a starting solution s^0, a *local optimum* is a permutation of s^0 with the best quality within the neighborhood of s^0.

This way, local optima constructed by opt_conflict or opt_fairness are either global optima or candidate solutions that cannot be improved by greedy local search, i.e. exclusively applying improving moves.

Definition 3.4.5. (Quasi-global optimum). Given a starting solution s^0 and a global optimum s^g with quality $q(s^g) \in [0, 1]$, we define a *quasi-global optimum* to be a permutation of s^0 with quality $q(s^0)$ such that $|q(s^0) - q(s^g)| \leq 0.001$.

3.4.2 Test set generation

Before analyzing in detail the performance of both opt_conflict and opt_fairness against a complete enumeration, we first describe how our test set was built.

In order to assess the performance of our heuristics we need to generate one or more *tractable* test sets. Creating a tractable test set can be achieved by limiting the size of the

communication schedule and the size of the seed solution, thereby limiting the size of the constructed sensing schedules. For our main test set, we decided to use aligned sensing schedules with size 8. This way, we have test cases that are neither too big nor too small, i.e. they still have some wiggle room for enough conflicts to happen, allowing us to have cases that are distinct enough to be representative of the general optimization problem while still being enumerated in able time. In addition, we created a secondary test set with sensing schedules of size 10. While our main test set, with $|s'_{com}| = |s^{(0)\prime}_{sens}| = 8$, has 3178 test cases, the smaller test set where the aligned communication and sensing schedules have 10 elements instead has 937 test cases. [1]

Given a set of channels C, we created the test sets by generating all possible combinations of size $l_{comb} \in \{3, \ldots, |C| - 2\}$ where all channels in the combination are distinct. For each combination k of channels, we generated a communication schedule s'_{com} with size $|C|$ by repeating channels inside k in unique ways. This was achieved by calculating all integer partitions of l_{comb}, i.e. all possible ways to partition l_{comb} into smaller integers that sum up to it. In addition, we use a balanced seed sensing schedule for both cases, i.e. $s^{(0)\prime}_{sens} = [c_1, c_2, \ldots c_{|C|}]$.

3.4.3 Similarity measures

In order to compare the results delivered by both complete enumeration and our heuristics, we introduce three similarity measures. These metrics will allow us to quantify the differences between the different optimization approaches and will work as a measure of performance of our heuristics when running on a given test set.

The first measure of similarity we are going to use is the Mean Squared Error (ϕ_{MSE}).

Definition 3.4.6. (Mean Squared Error). Given a test set with n test cases, let f_i and $\widehat{f_i}$ be the values delivered by two different optimization methods for a chosen optimization metric for test case i, we define

$$\phi_{MSE} = \frac{1}{n} \sum_{i=1}^{n} (f_i - \widehat{f_i})^2 \tag{3.6}$$

ϕ_{MSE} is a typical metric to gauge the quality of an estimation model, which includes both the bias and the variance of an estimator. In our case, if f_i is the value of an

[1] Two additional test sets with 19669 and 1680 test cases and respectively $|s^{(0)\prime}_{sens}|$ sizes of 13 and 36 were created for comparison with the Hungarian Method in Sec.3.7.3.

optimization metric delivered by complete enumeration for a test case i, an ideal heuristic will produce $\widehat{f_i}$ which approximates f as well as possible.

By calculating the mean squared error between the results delivered by our heuristics and the results obtained with complete enumeration, we can capture the overall performance of these heuristics. An exact method, for instance, would always yield $\phi_{MSE} = 0$ when compared with complete enumeration regardless of the test set at hand.

Furthermore, we define another measure of similarity, namely ϕ_g.

Definition 3.4.7. Given a test set with n test cases, let f and $\widehat{f}$ be the values delivered by two different optimization methods for a chosen optimization metric and n_{equal} the number of values where the methods deliver the same value, i.e. $f = \widehat{f}$, we define ϕ_g as $\phi_g = \frac{n_{equal}}{n}$.

If f is produced by complete enumeration and $\widehat{f}$ by one of our heuristics, ϕ_g represents the fraction of global optima found by the heuristic.

Our third similarity measure is ϕ_{quasi}. It addresses the points where f and $\widehat{f}$ differ, but not by much.

Definition 3.4.8. Given a test set with n test cases, let f and $\widehat{f}$ be the values delivered by two different optimization methods for a chosen optimization metric and n_{quasi} the number of values where the methods deliver values that differ by less than 0.001, i.e. $|f - \widehat{f}| < 0.001$, we define ϕ_{quasi} as $\phi_{quasi} = \frac{n_{quasi}}{n}$.

In this case, if f is produced by complete enumeration and $\widehat{f}$ by one of our heuristics, ϕ_{quasi} represents the fraction of quasi-global-optima found by the heuristic, i.e. points that are as good as global optima.

3.4.4 Results

In this section, we will display and analyze the performance of both opt_conflict and opt_fairness. For better visualization, we sorted the constructed solutions from worst to best quality, which in the case of the conflict metric will be in descending order and in the case of the overlap fairness in ascending order.

3.4.5 Test Set 1 - Primary goals

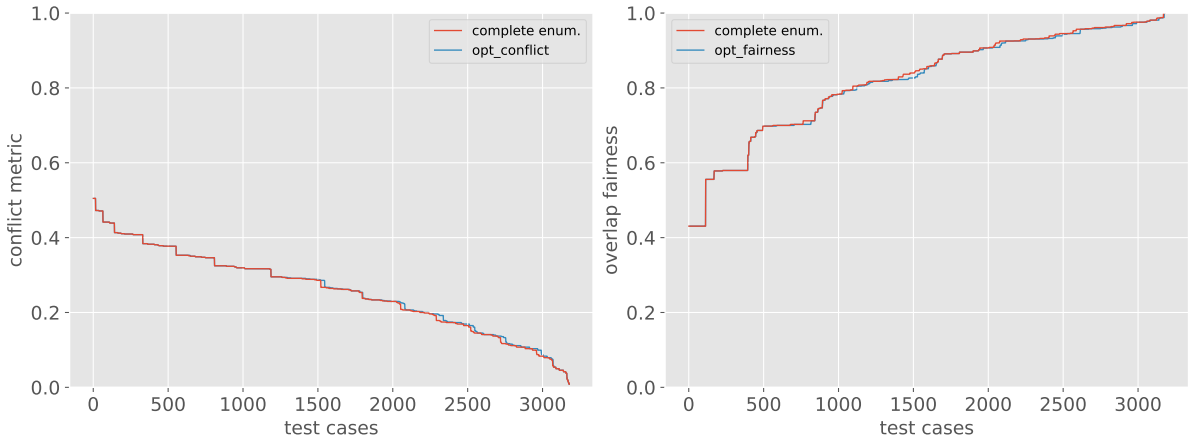
First, we will analyze the results of the optimization of the conflict metric by opt_conflict and overlap fairness by opt_fairness on test set 1. In Tab. 3.2, we can see that both

similarity	opt_conflict	opt_fairness
ϕ_{MSE}	$6.486 \cdot 10^{-5}$	$5.503 \cdot 10^{-5}$
ϕ_g	0.4292	0.8530
ϕ_{quasi}	0.915	0.8571

Table 3.2: Comparison of similarity measures of both opt_conflict and opt_fairness compared to complete enumeration.

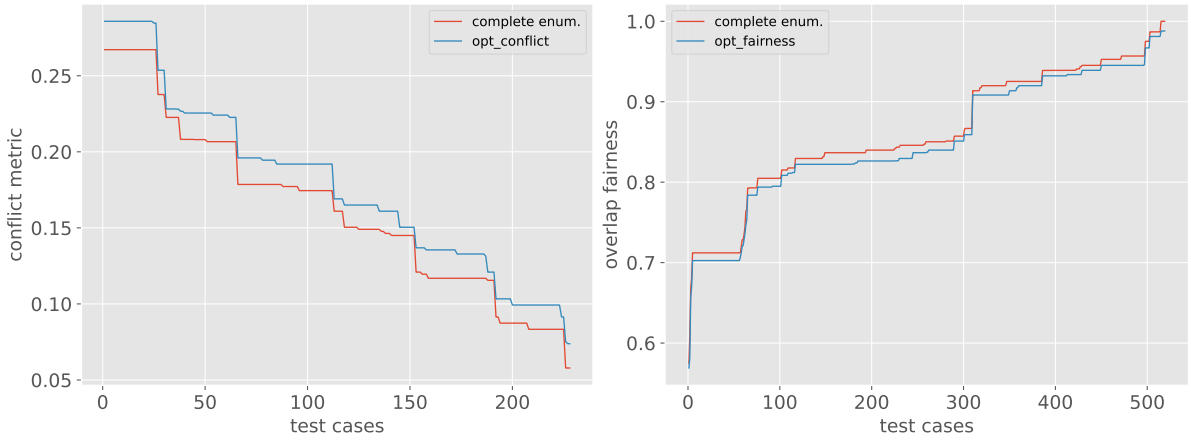
opt_conflict and opt_fairness deliver a very low ϕ_{MSE} value when optimizing for their primary optimization objectives. This is further confirmed by a quick visual inspection of both Fig. 3.7a and Fig. 3.7b.

With respect to the fraction of global optima, opt_fairness seems to have the upper hand achieving $\phi_g = 0.853$ whereas opt_conflict finds fewer than half of all global optima. On the other hand, opt_conflict seems to have a slightly better performance w.r.t. the fraction of quasi-global optima. Also, if we look at Fig. 3.8a, we can see that out of more than 3000 test cases, in only fewer than 250 cases the solutions constructed by opt_conflict have an absolute difference in conflict metric greater or equal to 0.005 w.r.t. to the global optima. In Fig. 3.8b, we can see that opt_fairness while achieving a very good performance, has slightly more than double as many cases in which the absolute difference in overlap fairness surpasses or equals 0.005. In addition, the maximum absolute difference in the values of the local and global optima was lower than 0.1 in both cases, with 0.066 for opt_fairness and 0.082 for opt_conflict. Even though not always able to find a global optimum, it is clear that both our heuristics display a very good performance in optimizing for their primary goals.



(a) opt_conflict (b) opt_fairness

Figure 3.7: Comparison of global optima of conflict metric and overlap fairness with local optima delivered by respectively opt_conflict and opt_fairness.



(a) opt_conflict (b) opt_fairness

Figure 3.8: Instances in which absolute difference between local and global optima equals 0.005 or greater.

3.4.6 Test set 1 - secondary optimization goals

In this section, we examine the results regarding the optimization of the secondary objective functions. The secondary objective function for opt_fairness is the conflict metric and for opt_conflict the overlap fairness. We compare the results of both optimization metrics with the results delivered by complete enumeration, i.e. global optima, as shown in Fig. 3.9b and Fig. 3.9a . We can clearly see a large difference when compared to the results for the primary optimization goals, as shown in Fig. 3.7b and Fig. 3.7a. Whereas

with the primary goals we saw a convergence of the heuristic results towards the global minima, we clearly see a divergence between global optima and the values delivered by the two heuristics.

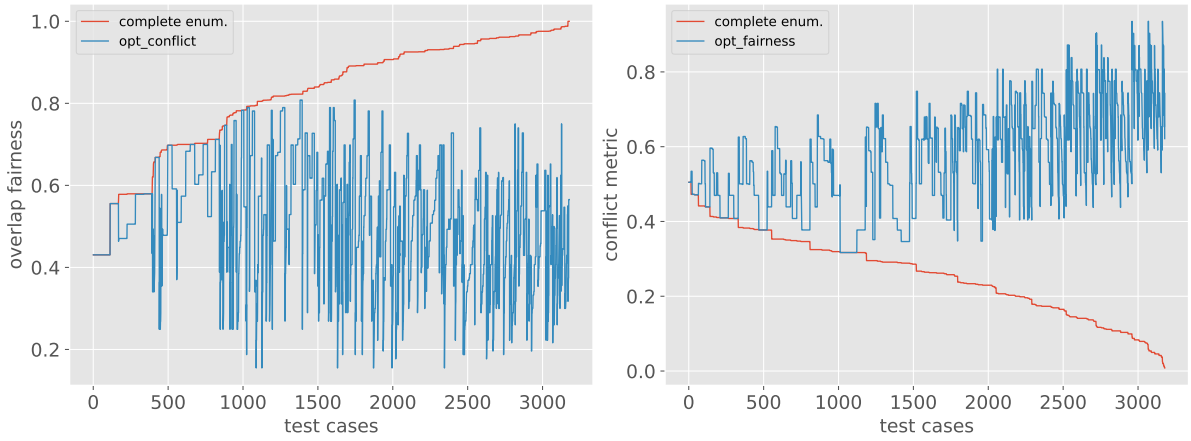
While for the primary optimization objectives the maximum difference between local and global optima for both opt_conflict and opt_fairness was below 0.1, the maximum absolute difference between the overlap fairness values delivered by opt_conflict and the global optima is 0.7799 and the maximum absolute difference for the conflict metric delivered by opt_fairness is 0.91.

If we look at the similarity measures in Tab. 3.3, we can see that ϕ_{MSE} is in the order of 10^4 times larger than in the primary case, and ϕ_g and ϕ_{quasi} are considerably lower than in the primary cases. It is interesting to note that both opt_conflict and opt_fairness still found around 10% of the quasi-global minima (in the case of opt_conflict these quasi-global optima were even real global maxima for overlap fairness).

However, despite encountering about 10% of the cases in which it is possible to optimize both metrics, the overall tendency for the remaining cases regarding the optimization of secondary goals is for the heuristic curves and the the global optima curves to diverge. This seems to indicate that neither local-search-based heuristics can reliably optimize both objectives, i.e. conflict metric and overlap fairness, at least for most cases, which indicates that the objective functions are conflicting.

similarity	opt_conflict	opt_fairness
ϕ_{MSE}	0.1365	0.14102
ϕ_g	0.1155	0
ϕ_{quasi}	0.1155	0.1073

Table 3.3: Comparison of similarity measures of secondary optimization goal results for opt_conflict and opt_fairness compared to complete enumeration.



(a) opt_conflict (b) opt_fairness

Figure 3.9: Results of optimization of secondary goal compared to global optima.

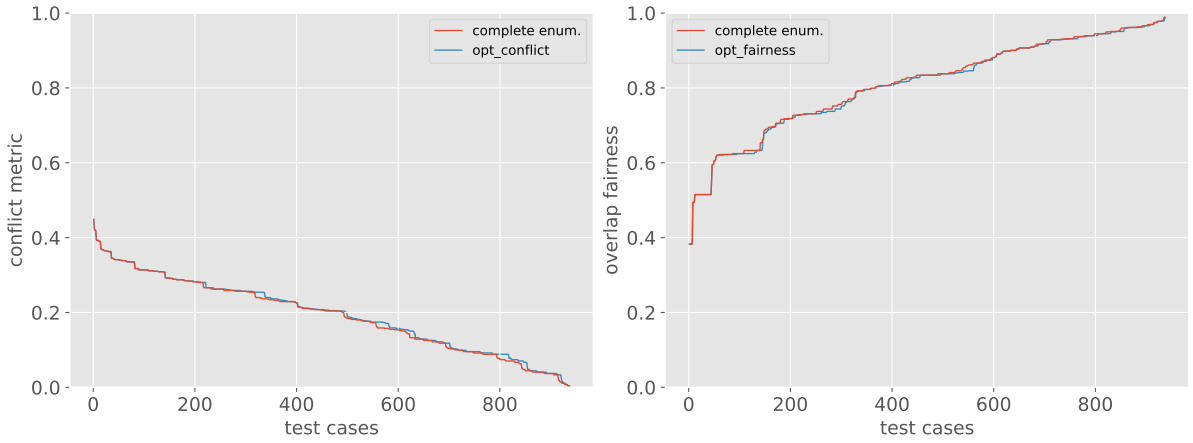
3.4.7 Test set 2 - Primary goals

In this section, we analyze the results for test set 2 and whether these are in accordance with the results regarding test set 1.

A quick visual inspection indicates a very similar performance of both opt_conflict and opt_fairness in test set 2 when compared to test set 1, see Fig. 3.10a and Fig. 3.10b. In both cases we seem to have again found high quality solutions, many of which are global optima. In Tab 3.4 we show the similarity values for test set 2, comparing both the global optima and the solutions delivered by our heuristics. As shown, both heuristics were able to find more than 82% of the quasi-global optima for their primary optimization goals with ϕ_{MSE} in the order of 10^{-4}. In addition, in only fewer than 250 test instances did opt_fairness and opt_conflict deliver a solution with a quality worse than the global one by equal to or more than 0.005, see Fig. 3.11a and 3.11b.

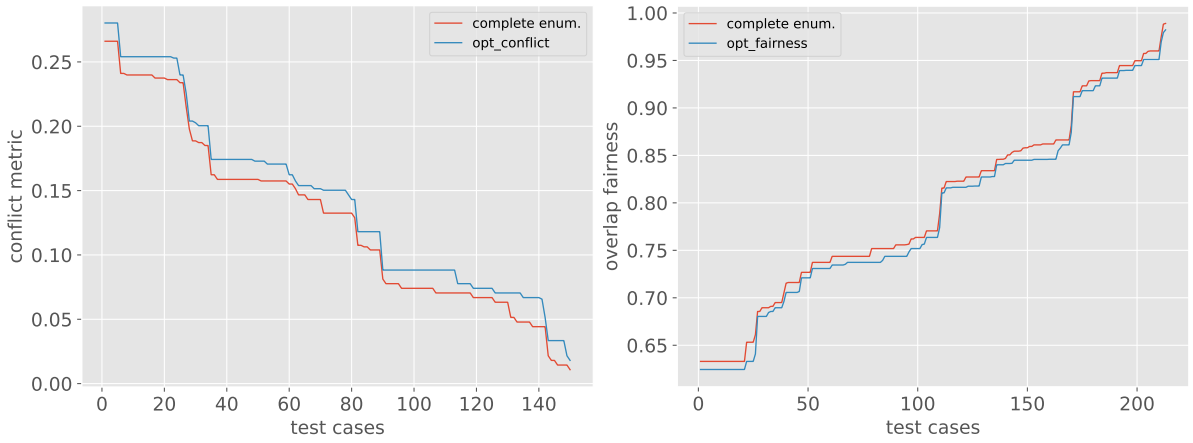
similarity	opt_conflict	opt_fairness
ϕ_{MSE}	$10.13 \cdot 10^{-5}$	$6.06 \cdot 10^{-5}$
ϕ_g	0.499	0.809
ϕ_{quasi}	0.825	0.828

Table 3.4: Comparison of similarity measures of results for opt_conflict and opt_fairness on test set 2 compared to complete enumeration.



(a) opt_conflict (b) opt_fairness

Figure 3.10: Comparison of global optima of conflict metric and overlap fairness with local optima delivered by respectively opt_conflict_ils and opt_fairness run on test set 2.



(a) opt_conflict (b) opt_fairness

Figure 3.11: Local optima delivered by opt_conflict and opt_fairness compared to global optima on test set 2 with absolute difference between optima equals 0.005 or higher.

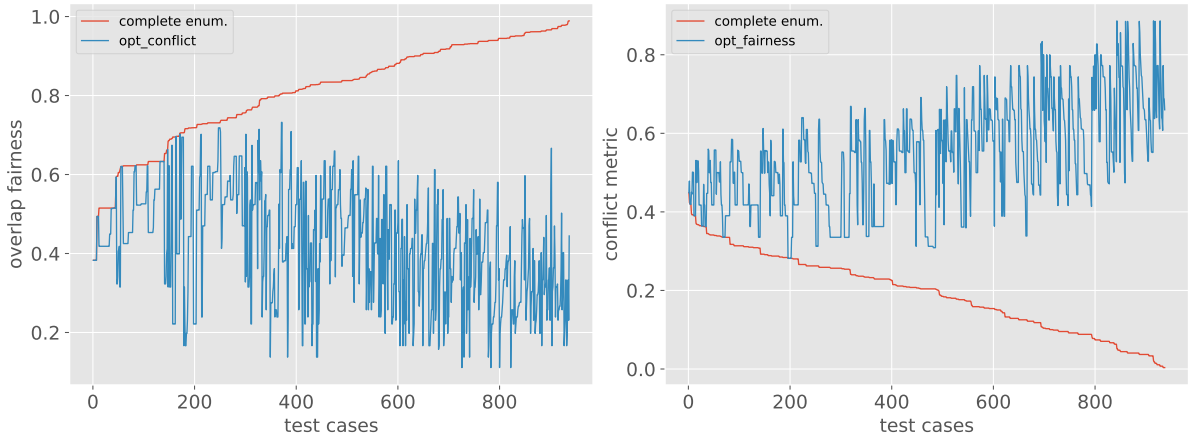
3.4.8 Test set 2 - Secondary goals

With regards to the secondary goals we do not see much difference when compared to test set 1 (see Fig. 3.12b and Fig. 3.12a). Still we see a great divergence between values delivered by the heuristics and the global optima. In fact, this seems to confirm that conflict metric and overlap fairness are for most of the cases conflicting objective

functions. In this test set, both opt_conflict and opt_fairness were able to find around 3% quasi-global optima for their secondary optimization goals.

metric	opt_conflict	opt_fairness
ϕ_{MSE}	0.1971	0.163741
ϕ_g	0.03842	0
ϕ_{quasi}	0.0384	0.0288

Table 3.5: Results for overlap fairness delivered by opt_conflict and conflict metric delivered by opt_fairness compared to global optima of test set 2.



(a) opt_conflict (b) opt_fairness

Figure 3.12: Results of optimization of secondary goal compared to global optima on test set 2.

3.4.9 Investigating divergences from optimal solutions

Even though the performance of our heuristics for their primary goals is already promising, we still have enough divergence from the optimal curve delivered by complete enumeration, since both *opt_conflict* and *opt_fairness* do not always reach the global optima.

Trying to determine whether we can identify any of the shortcomings of our heuristics, we started investigating the construction of sensing schedules performed by them. To better examine the differences between sub-optimal sensing schedules constructed by our algorithm and optimal ones delivered by a complete enumeration, we started analyzing for multiple test cases the swaps needed to move from any given sub-optimal schedule,

delivered by one of our heuristics, to one of its optimal permutations. For this purpose, we have implemented a graphical application (with help of the Qt widget toolkit), in which we are able to visualize all relevant swaps, both the swaps performed by our heuristics from a starting seed solution to a final sensing schedule and those needed to get from a local optimum to a global one (see Fig. 3.13). This helped us quickly identify some of the pain points of using our local-search-based algorithm.

As seen in Fig. 3.13, we computed a series of moves needed to move the local optimum delivered by opt_conflict to a global optimum. In order to arrive at the optimal sensing schedule

$$s_{sens} = [8, 1, 10, 2, 3, 4, 9, 11, 12, 5, 7, 6]$$

with the minimal conflict metric value of 0.15573, starting from local optimum

$$s^{\star}_{sens} = [8, 1, 12, 4, 5, 6, 7, 10, 9, 11, 2, 3]$$

with conflict metric value of 0.219312, we have to first perform a permutation that worsens the conflict metric, followed by four permutations that dot not affect the value of the conflict metric at all, and only then by applying successive improving moves we reach the global minimum, i.e.

$$s_{sens} = [8, 1, 10, 2, 3, 4, 9, 11, 12, 5, 7, 6]$$

In fact, we can see that we have at least four possible global optima w.r.t. conflict metric.

This seems to point us in the right direction. According to our investigations, the global minima we seem to be missing can be reached through what we will call *non-improving moves*. These moves are either *neutral* moves, i.e. swaps that leave the optimization metric at its current value or *worsening* moves, i.e. swaps that worsen the optimization metric.

It is hence quite clear why our heuristics cannot always reach global optima, since by using a greedy approach, in which every valid move has to either increase or decrease one of the optimization metrics, we eventually miss some needed intermediate non-improving moves. This means that in such cases our heuristics stay stuck in local minima.

s_com	s_sens	swap pos.	swap chann.	overlap fairness	conflict metric
[4, 7, 9, 9, 9, 9, 4, 7, 9, 9, 9, 9]	[8, 1, 12, 4, 5, 6, 7, 10, 9, 11, 2, 3]	(1,9)	(1, 11)	0.376347	0.219312
[4, 7, 9, 9, 9, 9, 4, 7, 9, 9, 9, 9]	[8, 1, 10, 4, 5, 6, 7, 12, 9, 11, 2, 3]	(2,7)	(10, 12)	0.345722	0.236362
[4, 7, 9, 9, 9, 9, 4, 7, 9, 9, 9, 9]	[8, 1, 10, 2, 5, 6, 7, 12, 9, 11, 4, 3]	(3,10)	(2, 4)	0.345722	0.236362
[4, 7, 9, 9, 9, 9, 4, 7, 9, 9, 9, 9]	[8, 1, 10, 2, 3, 6, 7, 12, 9, 11, 4, 5]	(4,11)	(3, 5)	0.345722	0.236362
[4, 7, 9, 9, 9, 9, 4, 7, 9, 9, 9, 9]	[8, 1, 10, 2, 3, 4, 7, 12, 9, 11, 6, 5]	(5,10)	(4, 6)	0.345722	0.236362
[4, 7, 9, 9, 9, 9, 4, 7, 9, 9, 9, 9]	[8, 1, 10, 2, 3, 4, 9, 12, 7, 11, 6, 5]	(6,8)	(9, 7)	0.318854	0.171782
[4, 7, 9, 9, 9, 9, 4, 7, 9, 9, 9, 9]	[8, 1, 10, 2, 3, 4, 9, 11, 7, 12, 6, 5]	(7,9)	(11, 12)	0.309546	0.155733
[4, 7, 9, 9, 9, 9, 4, 7, 9, 9, 9, 9]	[8, 1, 10, 2, 3, 4, 9, 11, 12, 7, 6, 5]	(8,9)	(12, 7)	0.309546	0.155733
[4, 7, 9, 9, 9, 9, 4, 7, 9, 9, 9, 9]	[8, 1, 10, 2, 3, 4, 9, 11, 12, 5, 6, 7]	(9,11)	(5, 7)	0.309546	0.155733
[4, 7, 9, 9, 9, 9, 4, 7, 9, 9, 9, 9]	[8, 1, 10, 2, 3, 4, 9, 11, 12, 5, 7, 6]	(10,11)	(7, 6)	0.309546	0.155733

Figure 3.13: Converging from solution constructed by our heuristic to an optimal solution generated by complete enumeration.

3.5 Improving local minima

Ours is a combinatorial optimization problem: we want to either maximize or minimize our chosen optimization criteria, i.e. overlap fairness and conflict metric, given a set of constraints, i.e. a fixed set of channels and a current communication schedule.

As seen in the previous section, our basic algorithm has one major drawback when compared to complete enumeration: even though it quickly converges to an optimum, this optimum might be only a local one. Escaping from a local optimum to a better one (possibly a global one) is not trivial.

3.5.1 Random restart

Randomness is a key component to solve many combinatorial optimization problems [Wal99], among which SAT and graph coloring are prominent examples.

In fact, one of the easiest ways to try to escape from a local minimum is to begin a new local search from another *random* starting point. This yields multiple local minima that are independently generated. This approach could be interpreted as a softer variant of a complete enumeration approach. While in theory multiple trials increase the probability of reaching a better local minimum [LMS10] empirical studies have shown that it is almost impossible to improve a local minimum generated by local search through random restarts even by a small percentage of the typical cost, e.g. the values of conflict metric and overlap fairness delivered by respectively opt_conflict and opt_fairness.

This happens because local search approaches tend to produce a mean that is a fixed percentage worse than the global optimum and the optimization metric values peak *arbitrarily* around the mean when the input size goes to infinity. This means, by random sampling the solution space one has a decreasing probability of finding better solutions as the size of the problem instance increases. Hence, a sensible approach to effectively reach some of the better solutions is to perform a biased sampling of the solution space.

3.5.2 Iterated local search

In our case, this biased sampling of the solution space, i.e. the set of all possible sensing schedules, will be achieved through a stochastic search following an iterated local search (ILS) metaheuristic. Metaheuristics are general-purpose solution finding frameworks and many different ones were proposed in the literature for solving combinatorial problems.

The basic idea behind ILS is to construct a chain of solutions by re-applying a core heuristic, usually in a reduced solution space, such that solutions are improved iteratively. Iterated local search has been used in many different contexts and has been re-discovered multiple times under many different names such as iterated descent [Bau86], iterated Lin-Kernighan [Joh90], chained local optimization [MO96] and large-step Markov chains [MOF91], to name a few. In fact, iterated local search is usually applied to local searches, but any underlying optimizer may be embedded within this metaheuristic [LMS19].

To improve local minima delivered by our basic heuristics we apply the ILS metaheuristic as follows (see Fig. 3.14):

1. We first compute an initial solution s'_{sens} with one of our greedy local search heuristics.
2. Perturb the constructed solution, arriving at intermediate state $s^{\star}$.
3. Re-apply our local search to the perturbed state.
4. Accept the new solution s'', if it is better than the original solution s'_{sens} with respect to the optimization criteria.
5. We then repeat this procedure until a given termination condition is met, which means a given *exploration budget* is used up.

Regarding the initial solution to be handed to the ILS procedure, we have decided to start with a local optimum delivered by one of our local-search-based heuristics, as

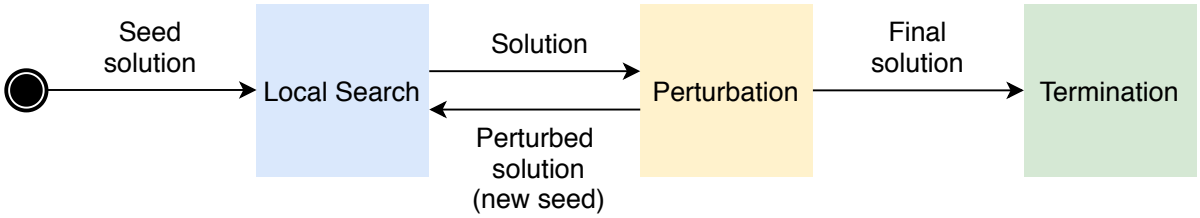


Figure 3.14: Diagram showing the basic idea of the iterated local search.

this often helps reaching better local optima [LMS10]. Nonetheless, depending on the optimized problem, starting with random solutions might gain the upper-hand compared to greedy initial solutions, as shown in [JM08] in the context of solving the Traveling Salesman Problem (TSP) with the Clarke-Wright savings heuristic where random initial solutions led to slightly better tour quality on average. As a short aside, the TSP has strong similarities to our optimization problem. Given N cities $\{c_1, c_2, \dots c_N\}$ with distances $d(c_i, c_j)$ between every pair of cities (c_i, c_j), the TSP can be defined as finding a permutation π (an order in which to visit the cities) such that the tour length, i.e. $d(c_{\pi(N)}, c_{\pi(1)}) + \sum_{i=1}^{N-1} d(c_{\pi(i)}, c_{\pi(i+1)})$, is minimized.

Another essential component in the ILS-based approach is the *perturbation*. In the context of our optimization problem, a perturbation is any move that allows us to move from the current solution to a neighbor solution that cannot be reached with $opt\_conflict$ or $opt\_fairness$. Since our local searches are greedy and only make improving moves this can be achieved by having a perturbation be a sequence of *non-improving* moves.

In addition, the exploration budget can take many forms. An imaginable termination condition would be reaching a target optimization metric value or reaching a target percentage improvement with respect to the original solution. It is not trivial though to compute such a target beforehand and due to the nature of the problem it is not possible to determine a priori whether a chosen target is reachable or not. Other possible forms of such an exploration budget are for example restrictions on the total running time or on the number of times a local search should take place.

In order to assess the performance of ILS on top of our heuristics we implemented an iterated local search variant of each of our basic heuristics: $opt\_conflict\_ils$ and $opt\_fairness\_ils$. In this implementation, we define our exploration budget through two parameters that guarantee the termination of our algorithm. These parameters are n_{swaps} and n_{pert}, respectively the number of swaps that take place within each perturbation and the total number of perturbations that take place before our algorithm finishes.

This means that for each of the ILS variants, every perturbation of a sensing schedule s'_{sens} consists of n_{swaps} channel swaps that might either worsen the chosen optimization metric or keep this optimization metric unchanged. The new perturbed schedule $s^{\star}$ is then used as a new starting point for either opt_conflict or opt_fairness, which then attempts to find a better sensing schedule s'', i.e. closer to the global optimum (see Fig. 3.15). In total, we perform n_{pert} perturbations starting with the original solution delivered by the underlying local search heuristic.

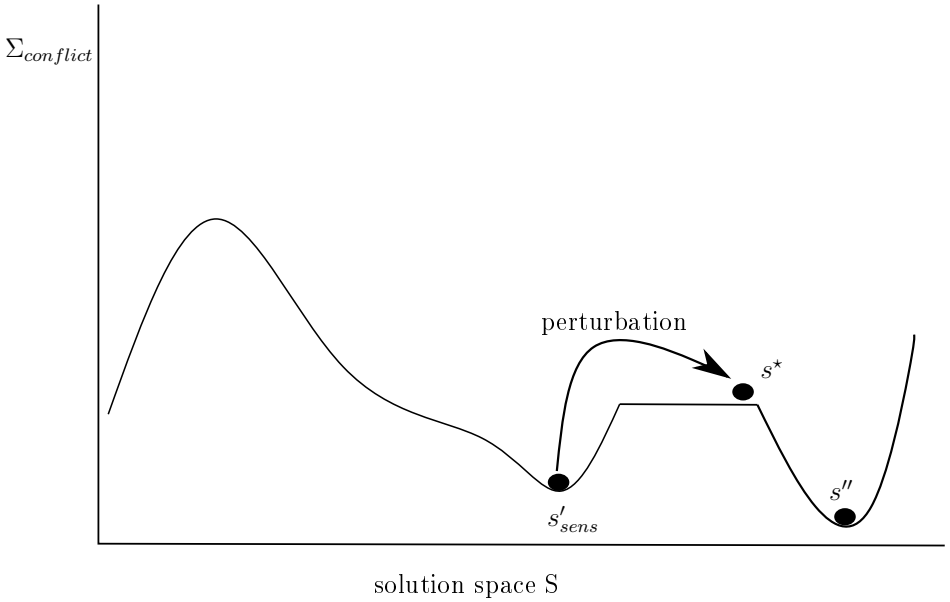


Figure 3.15: Iterated local search with embedded opt_conflict: a constructed solution s'_{sens} is perturbed to an intermediate state $s^{\star}$ from which we can search for another solution s'', such that $\Sigma_{conflict}(s'_{com}, s'') < \Sigma_{conflict}(s'_{com}, s'_{sens})$.

Ideally, perturbations should have a random component since deterministic changes to the current solution might lead to shorter cycles. This means, by adding randomness to the ILS-based heuristics, we convert the traversal of the solution space into *stochastic* local searches. To achieve this, for every swap in a perturbation, we randomly choose a slot x and make the first non-improving move (x, y) we can find. As we re-run the appropriate heuristic after each perturbation, we always keep the best solution of all attempts.

In fact, the effectiveness of ILS biased sampling of the solution space is highly dependent on the optimization problem at hand and the chosen acceptance criterion for intermediate solutions. For instance, some techniques might take into account not only

the quality of these candidate solutions but also the history of the already visited solutions. Whereas random walks accept *any* intermediate solutions, thereby prioritizing diversification over intensification of the search process, our approach uses a Markovian acceptance criterion. This means that no history is considered and that only better candidate solutions than the current best one are accepted. Alternative approaches might use acceptance criteria that fall in between ours and random walks. One such approach uses large-step Markov chains [MOF91], in which a new candidate solution s' can be accepted with probability p_{accept}.

$$p_{accept} = \begin{cases} 1 & \text{if } q(s') > q(s) \\ e^{-\frac{|q(s')-q(s)|}{T}} & \text{if } q(s') \leq q(s) \end{cases}$$

where s is the current candidate solution and $q(s')$ and $q(s)$ are respectively the qualities of s' and s w.r.t. an optimization goal. This acceptance criterion follows a simulated annealing approach and the time-varying temperature parameter T can be lowered during the iterated search, restricting the diversification of the search process as it progresses. This way, as T decreases, the search increasingly prioritizes improving solutions over non-improving ones.

Even though such a simulated annealing approach might be attractive and display good performance in different contexts, it adds complexity to the optimization problem and increases the number of parameters to be fine-tuned. We have hence opted for a simpler acceptance criterion, in which non-improving candidate solutions are discarded. However, as our results will show later in this chapter, we have obtained very impressive results, which seems to indicate that we have made a good choice w.r.t. the acceptance criterion. It is worth noting that for problems in which instances might get much larger, e.g. with hundreds or thousands of slots, more diversification might be needed during the sampling of the solution space to achieve similar results.

With Alg. 3 and its associated sub-routines Alg. 4, Alg. 5 and Alg. 6, we summarize our ILS implementation.

One of the main difficulties of the ILS approach is fine-tuning the perturbations to improve the local minima. If the perturbation is too large we lose any bias in our sampling of the solution space and end up degenerating into random restarts. On the other hand, if the perturbation is too small we may walk in cycles returning to the original solution. In addition, minimally worsening swaps might get us stuck on a plateau, i.e. a region where all neighbors have the same values for the objective function. Later in this

Algorithm 3 Iterated Local Search

function ILS(s_{com}, C, h) ▷ $h \in \{opt\_conflict, opt\_fairness\}$
  $s^{(0)}_{sens} \leftarrow ConstructSeedSolution(C)$
  $s'_{com}, s^{(0)\prime}_{sens} \leftarrow AlignSchedules(s_{com}, s^{(0)}_{sens})$
  $s^{(0)\prime}_{sens} \leftarrow LocalSearch(s'_{com}, s^{(0)\prime}_{sens}, h)$ ▷ Initial solution
  $n \leftarrow 0$
  repeat
    $s^{\star} \leftarrow Perturbation(s'_{com}, s^{(0)\prime}_{sens}, h)$ ▷ see Alg. 4
    $s'' \leftarrow LocalSearch(s'_{com}, s^{\star}, h)$
    $s^{(0)\prime}_{sens} \leftarrow Better(s'_{com}, s^{(0)\prime}_{sens}, s'', h)$
    $n \leftarrow n + 1$
  until $n = n_{perturbations}$
  $s'_{sens} \leftarrow s^{(0)\prime}_{sens}$
end function

Algorithm 4 Perform n_{swaps} swaps on $s^{(0)\prime}_{sens}$

function PERTURBATION($s'_{com}, s^{(0)\prime}_{sens}, h$)
  $s^{\star} \leftarrow s^{(0)\prime}_{sens}$
  $n \leftarrow 0$
  repeat
    $x \leftarrow GetRandomBetween(1, |s^{\star}|)$
    $y \leftarrow 1$
    repeat
      $y \leftarrow y + 1$
    until $no\_improvement(x, y, s'_{com}, s^{\star}, h)$ ▷ see Alg. 5
    $s^{\star} \leftarrow f_{switch}(s^{\star}, x, y))$ ▷ Swap slot x with y
    $n \leftarrow n + 1$
  until $n = n_{swaps}$
  return $s^{\star}$
end function

section, we will explore different parameter settings and assess the resulting performance of each of these settings w.r.t. the quality of the obtained sensing schedules.

It is also worth noting that even though picking the positions to make a swap might be random, one possible optimization would be to add a *tabu* rule in order to avoid going back to a permutation we have already visited in the current chain of swaps. This requires some bookkeeping and consequently yields a higher demand for memory but could improve the speed of convergence to a better optimum.

Algorithm 5 Check that swap (x,y) does not improve primary optimization metric

1: **function** NO_IMPROVEMENT($x, y, s'_{com}, s^{\star}, opt\_metric$)
2: $s_t \leftarrow f_{switch}(s^{\star}, x, y))$
3: old_metric = ComputeMetric($s'_{com}, s^{\star}, h$)
4: new_metric = ComputeMetric(s'_{com}, s_t, h)
5: **if** $h = opt\_conflict$ **then**
6: return $new\_metric \geq old\_metric$
7: **else**
8: return $new\_metric \leq old\_metric$
9: **end if**
10: **end function**

Algorithm 6 Decide whether s'' is better than $s^{(0)\prime}_{sens}$ w.r.t. the primary optimization goal

1: **function** BETTER($s'_{com}, s^{(0)\prime}_{sens}, s'', h$)
2: old_metric = ComputeMetric($s'_{com}, s^{(0)\prime}_{sens}, h$)
3: new_metric = ComputeMetric(s'_{com}, s'', h)
4: **if** $h = opt\_conflict$ **then**
5: return $new\_metric < old\_metric$
6: **else**
7: return $new\_metric > old\_metric$
8: **end if**
9: **end function**

3.5.3 Implementing randomness

As already mentioned, one of the key additions to our original implementation is randomness. As a matter of fact, we would like for the randomly chosen swap positions during each perturbation to be serially uncorrelated. Ideally our random number generator would have an infinite period (the series of generated numbers would never cycle) and the generated sequence of integers should be uniform and unbiased. For practical reasons we make use of a pseudo-random generator (PRNG). Even though PRNGs do not provide *real* randomness, we can pick one that provides us with cycles of integers with a large enough period. In fact, we have opted for using a Mersenne Twister (MT19937), which not only has a very large period of $2^{19937} - 1$, but also produces a statistically uniform distribution of values and has been shown in the literature to produce high-quality pseudo-random sequences [HS05]. By using the MT19937 we have an efficient random number generator that all in all should be good enough for our purposes. It is also worth

noting that there is no evidence in the literature whether *real* randomness would bring any advantage over state-of-the-art PRNGs such as the Mersenne Twister.

3.5.4 Time complexity

With the addition of ILS, we increase the run time of each generation of a sensing schedule, since by construction our ILS algorithm terminates after running $n_{pert} + 1$ local searches. Nonetheless, we are interested in finding out, whether we also increase the asymptotic time complexity class of our heuristics. Hence, we will calculate the time complexity of our algorithm with and without ILS.

We have following elements involved in our problem:

1. The set of all n_c channnels, $C = \{c_1, c_2, \ldots, c_{n_c}\}$.
2. A communication schedule s_{com}, with $|s_{com}| = n$.
3. A seed sensing schedule $s^{(0)}_{sens} = [c_1, c_2, c_3, \ldots c_{n_c}]$

This means that in the worst case, i.e. when n_c and n are co-prime, the aligned versions of the communication schedule s'_{com} and of the seed sensing schedule $s^{(0)\prime}_{sens}$ have $n_c \cdot n$ elements.

In the basic version of our heuristics (opt_conflict or opt_fairness), every element yields at least a whole scan of s'_{com} and s'_{sens} (in order to find the swap positions needed to improve the chosen optimization metrics). This means the basic version of our algorithm is $\mathcal{O}((n_c * n)^2) = \mathcal{O}(n^2)$.

By adding the ILS variant on top of it, we run our basic heuristic once and then perform n_{pert} perturbations with n_{swaps} each and re-run our basic heuristic for each perturbation, which means $opt\_conflict\_ils$ or $opt\_fairness\_ils$ are:
$\mathcal{O}(n^2) + n_{pert} \cdot (\mathcal{O}(n^2) + n_{swaps}) = \mathcal{O}(n^2)$.

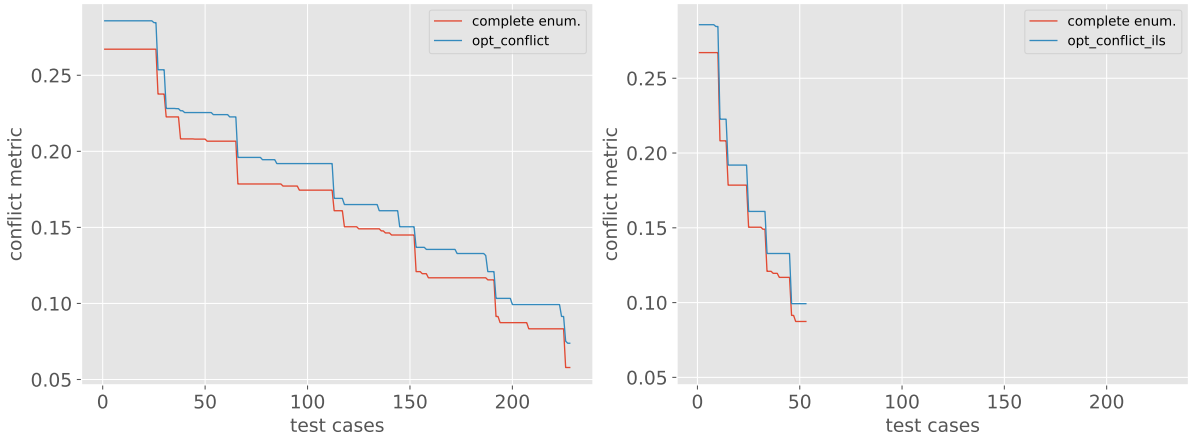
This means that even though the ILS-based variants take longer to terminate than the original heuristics, the asymptotic time complexity remains the same.

3.5.5 Experimental results

In this section, we will analyze the results of multiple conducted experiments in order to assess the performance of the ILS heuristic variants.

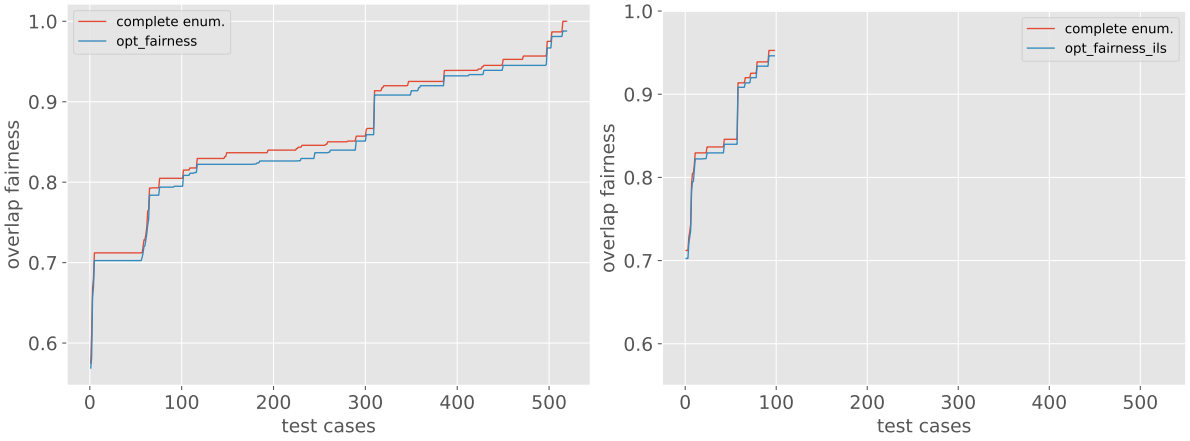
First impressions

We ran opt_conflict_ils and opt_fairness_ils on test set 1 and compared their results with our basic heuristics to assess whether any significant improvements appear. With a quick visual inspection, we can clearly see the improvements in conflict delivered by opt_conflict_ils (see Fig. 3.16b) when compared to opt_conflict (see Fig. 3.16a). Similar results were obtained for overlap fairness (see Fig. 3.17a and Fig. 3.17b). Both ILS variants were run with $n_{pert} = 5$ and $n_{swaps} = 2$. It is clear to see that using iterated local search introduced clear improvements to local minima compared to those delivered by our basic heuristics.



(a) opt_conflict (b) opt_conflict_ils

Figure 3.16: Test cases where absolute difference in conflict metric between global and local optima is 0.005 or higher. As shown, opt_conflict_ils brings a clear improvement and has far fewer such cases (about a fourth of the number of cases for opt_conflict).



(a) opt_fairness (b) opt_fairness_ils

Figure 3.17: Test cases where absolute difference in overlap fairness between global and local optima is 0.005 or higher. As we can see, opt_fairness_ils has far fewer such cases (a fifth of the number of cases for opt_fairness).

3.5.6 Parameter exploration

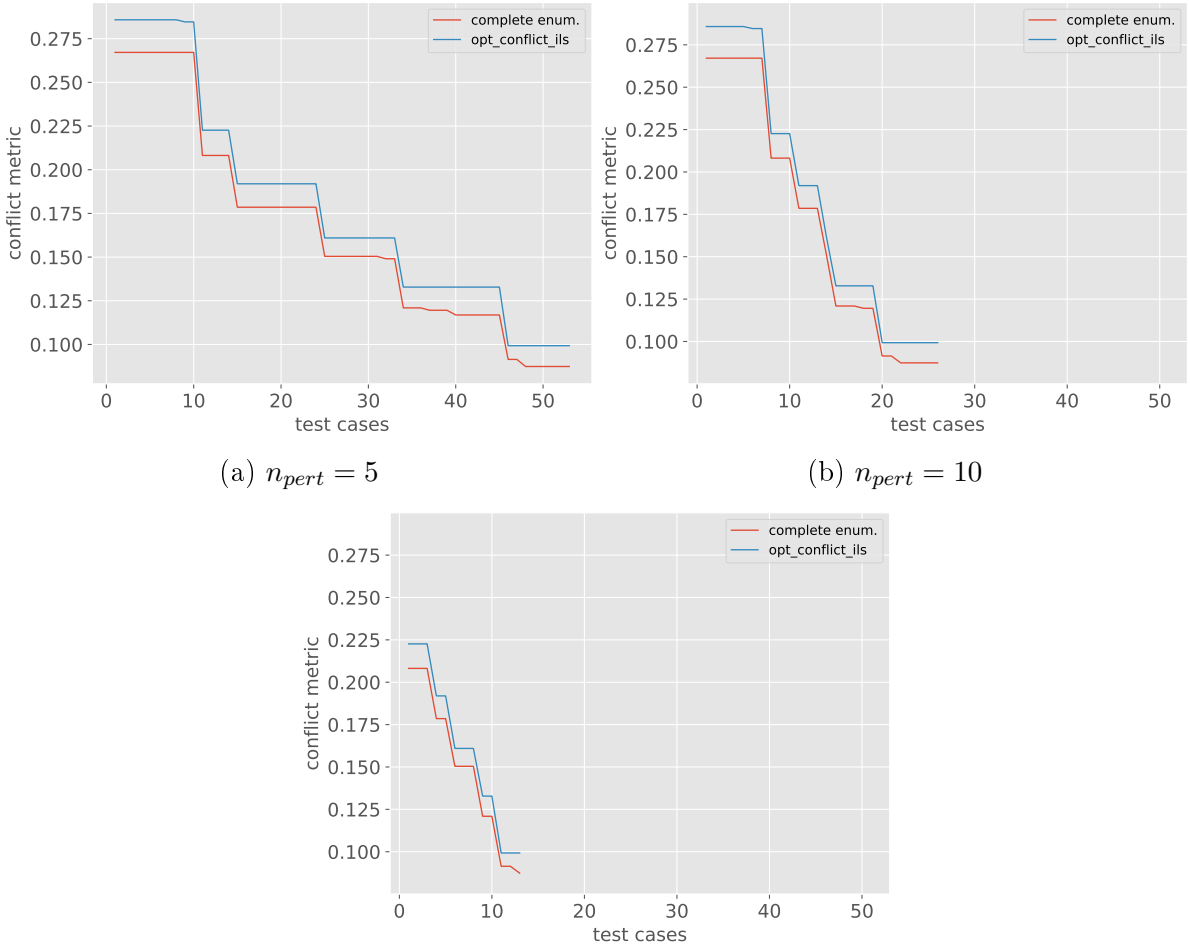
In this section, we explore different parameter configurations w.r.t. n_{swaps} and n_{pert} and analyze the resulting effects on the performance of our heuristics.

As previously mentioned, by using a greedy approach in both our local search heuristics, we are biased towards improving solutions. Introducing some randomness as well as non-improving moves with the ILS approach allows for some lateral movement within the solution space. Essentially, the farther we move within the neighborhood of a given solution, the larger the expected change in the value of the optimization metric. To determine how much lateral movement is still meaningful, it is essential to explore different parameter configurations. If our intuition is correct, there is a correlation between the quality of neighboring solutions, i.e. good solutions lie close to each other in the solution space and adding too much variation (n_{swaps} is too high) might even be detrimental to the quality of the solutions, given a fixed number of perturbations n_{pert}.

Varying n_{pert}

First, we want to investigate the effects of varying the total number of perturbations n_{pert} (while keeping n_{swaps} constant) when constructing a sensing schedule with one of the ILS variants. For this purpose, we run both opt_conflict_ils and opt_fairness_ils on test set 1 with $n_{pert} \in \{5, 10, 20\}$ and $n_{swaps} = 2$. As we can see in Figures 3.18a, 3.18b, and 3.18c there is a progressive improvement in conflict metric when increasing n_{pert}

from 5 to 10 and then to 20. If we take a look at Tab. 3.6, we can see an improvement in all similarity measurements when increasing the number of perturbations. For instance, whereas with the basic heuristic opt_conflict we only reached about 43% of the global optima, with $n_{pert} = 20$ we reached around 77% of all global optima.



(a) $n_{pert} = 5$

(b) $n_{pert} = 10$

(c) $n_{pert} = 20$

Figure 3.18: Test cases in which absolute difference between global and local optima of conflict metric is is 0.005 or higher. opt_conflict_ils is run with $n_{pert} \in \{5, 10, 20\}$ and $n_{swaps} = 2$ on test set 1. As shown, the number of test cases nearly halves when doubling n_{pert}.

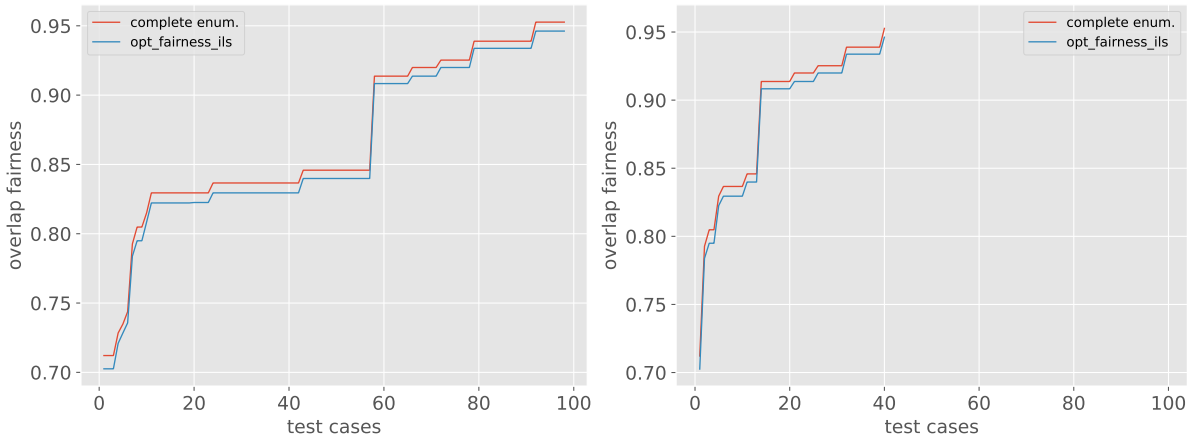
In Figures 3.19a, 3.19b, and 3.19c, we can see a similar improvement when optimizing the overlap fairness with increasing number of perturbations. Again, as we can see in Tab. 3.7, there were improvements in all similarity measurements, and the percentage of found global optima went from 85.3% with $n_{pert} = 0$ to 98.7% when $n_{pert} = 20$.

similarity	opt_conflict	$n_{pert} = 5$	$n_{pert} = 10$	$n_{pert} = 20$
ϕ_g	0.429201	0.684393	0.742920	0.773128
ϕ_{MSE}	0.000065	0.000015	0.000006	0.000004
ϕ_{quasi}	0.915041	0.976715	0.987728	0.992763
max. abs. difference	0.081693	0.052204	0.052204	0.052204

Table 3.6: Comparison of opt_conflict and opt_conflict_ils with $n_{swaps} = 2$ and $n_{pert} \in \{5, 10, 20\}$.

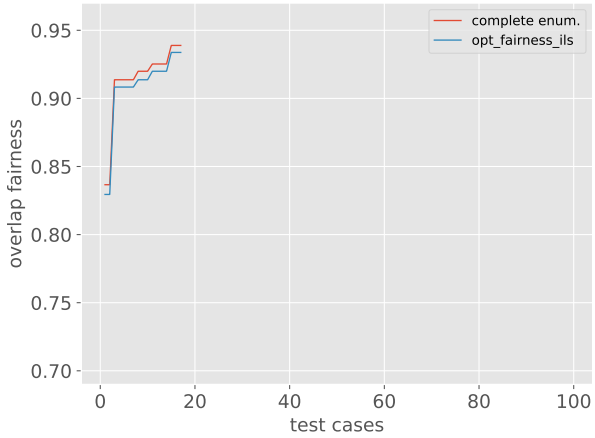
similarity	opt_fairness	$n_{pert} = 5$	$n_{pert} = 10$	$n_{pert} = 20$
ϕ_g	0.853052	0.967590	0.978918	0.986784
ϕ_{MSE}	0.000055	0.000008	0.000004	0.000002
ϕ_{quasi}	0.857143	0.970107	0.980806	0.988043
max. abs. difference	0.065840	0.028593	0.028593	0.025427

Table 3.7: Comparison of opt_fairness and opt_fairness_ils with $n_{swaps} = 2$, $n_{pert} \in \{5, 10, 20\}$.



(a) $n_{pert} = 5$

(b) $n_{pert} = 10$



(c) $n_{pert} = 20$

Figure 3.19: Comparison of overlap fairness values delivered by complete enumeration and opt_fairness_ils with $n_{pert} \in \{5, 10, 20\}$ and $n_{swaps} = 2$ on test set 1 on test cases where absolute difference between global and local optima is 0.005 or higher. As shown, the number of such cases is more than halved by doubling n_{pert}.

Varying n_{swaps}

Now, we will analyze the effects on the optimization results when keeping the number or perturbations constant while applying different *strength* values to each perturbation, i.e. different n_{swaps} values. As we can see in Figures 3.21a, 3.21b, and 3.21c, it is possible to see some improvement in the conflict metric through visual inspection alone, but it is not as clear cut as when varying n_{pert}. If we look at Tab. 3.8, we can see that even though all ILS variants display a significant improvement in comparison to the basic heuristic, increasing the strength of the perturbation does not always yield better results in the similarity measures. Even though $n_{swaps} = 6$ found the most global optima, the overall quality from the delivered near-optimal solutions suffered when compared to $n_{swaps} = 4$ (ϕ_{MSE} almost doubled). We can see in Tab. 3.9 that the results for the opt_fairness_ils were very similar, though while we see very similar results w.r.t. ϕ_g and ϕ_{quasi} (see Fig. 3.22a, 3.22b and 3.22c) there is a clear deterioration in ϕ_{MSE} when increasing n_{swaps}. This seems to indicate that there is an optimal spot to be hit with regards to the strength of the perturbation. In addition, very similar results were obtained when running opt_conflict and opt_fairness and its ILS variants with the same test instances but with less perturbations, i.e. $n_{pert} = 5$. Note that using $n_{swaps} = 6$ is equivalent to making 6 random swaps out of 8 slots (in test set 1), which amounts to modifying at most 75% of any candidate solution, which in some cases seems to be too excessive . In fact, with every swap in a perturbation we accept moves that either worsen the metric or keep it at its current value. This means that increasing the number of swaps per perturbation might worsen the current local optimum *a bit too much* having a negative effect in the total improvement of the *final* local optimum. If a better local optimum lies close to the sensing schedule being perturbed, the probability of finding it should be higher by using fewer swaps given the same number of perturbations. We illustrate this effect in Fig. 3.20.

However, it appears to be possible to compensate for this solution quality deterioration effect caused by excessively increasing n_{swaps} by also increasing the number of perturbations, as shown in Tab. 3.10. Nonetheless, this approach seems to waste more resources (run time) than it is worth for the very small increase in solution quality.

Moreover, we performed similar experiments on test set 2 as well as on test set 1 with different seeds for the random number generator to account for small effects resulting from randomness. All experiments resulted in very similar results.

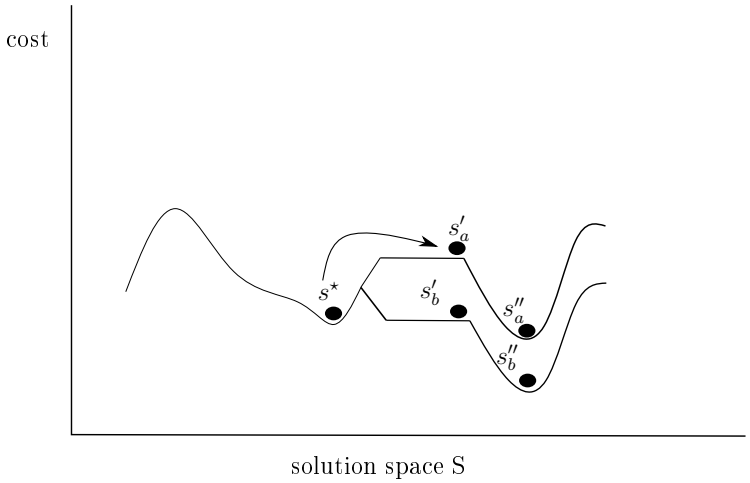


Figure 3.20: Illustration of the possible effects of too strong perturbations, i.e. *too many* swaps per perturbation, can lead to an improved local optimum s''_a but not as good as s''_b.

similarity	opt_conflict	$n_{swaps} = 2$	$n_{swaps} = 4$	$n_{swaps} = 6$
$\phi_{MSE} \cdot 10^6$	77.948	6.265	3.7456	5.9899
ϕ_g	0.4218	0.7386	0.7796	0.7848
ϕ_{quasi}	0.9085	0.9864	0.9920	0.9889

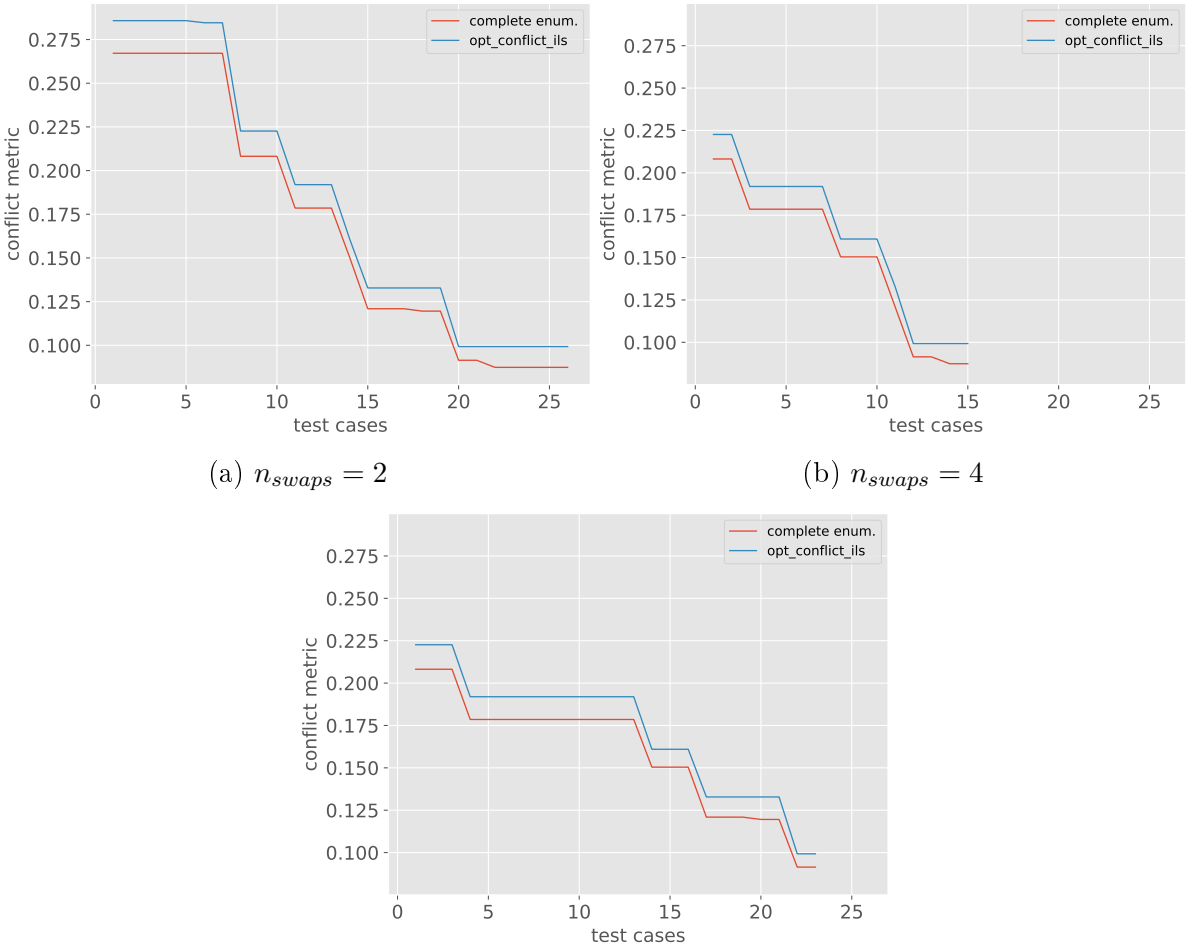
Table 3.8: Comparison of opt_conflict_ils with $n_{pert} = 10$ and $n_{swaps} \in \{2, 4, 6\}$.

similarity	opt_fairness	$n_{swaps} = 2$	$n_{swaps} = 4$	$n_{swaps} = 6$
$\phi_{MSE} \cdot 10^6$	58.598	2.8052	4.7031	5.28499
ϕ_g	0.8557	0.9840	0.9819	0.97705
ϕ_{quasi}	0.8599	0.9857	0.9840	0.9788

Table 3.9: Comparison of opt_fairness_ils with $n_{pert} = 10$ and $n_{swaps} \in \{2, 4, 6\}$.

similarity	opt_conflict	$n_{swaps} = 2$	$n_{swaps} = 4$	$n_{swaps} = 6$
$\phi_{MSE} \cdot 10^6$	77.948	6.555	3.249	2.0924
ϕ_g	0.4218	0.7688	0.8119	0.8179
ϕ_{quasi}	0.9085	0.9898	0.9920	0.9910

Table 3.10: Comparison of opt_conflict_ils with $n_{pert} = 20$ and $n_{swaps} \in \{2, 4, 6\}$.

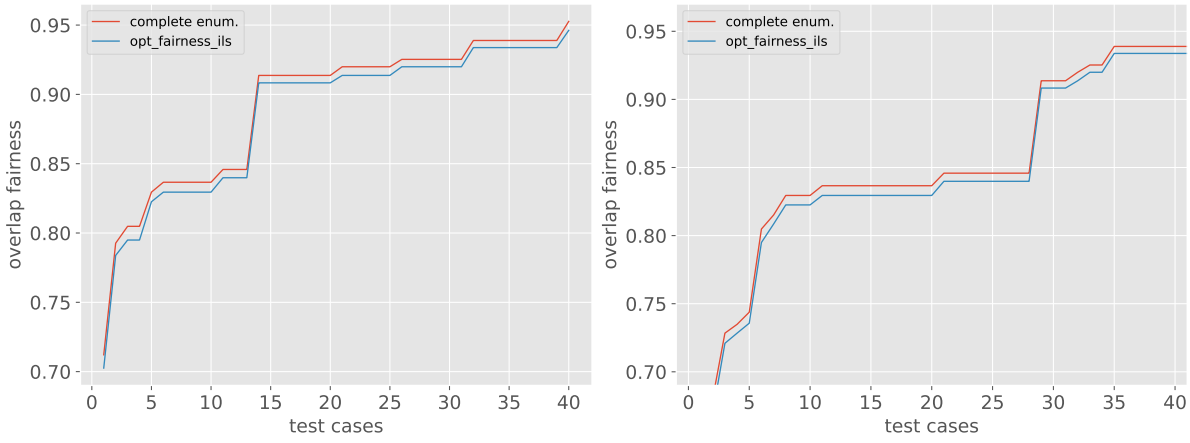


(a) $n_{swaps} = 2$

(b) $n_{swaps} = 4$

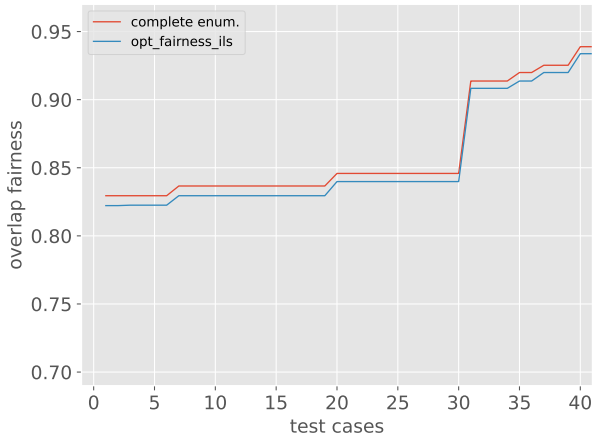
(c) $n_{swaps} = 6$

Figure 3.21: Comparison of conflict metric values delivered by opt_conflict_ils with $n_{swaps} \in \{2, 4, 6\}$ and $n_{pert} = 10$ on test set 1 with test cases where the absolute difference between global and local optima is 0.005 or higher.



(a) $n_{swaps} = 2$

(b) $n_{swaps} = 4$



(c) $n_{swaps} = 6$

Figure 3.22: Comparison of overlap fairness values delivered by opt_fairness_ils with $n_{swaps} \in \{2, 4, 6\}$ and $n_{pert} = 10$ on test set 1 where the absolute difference between global and local optima is 0.005 or higher. As shown, increasing n_{swaps} has lead either to a similar or worse performance.

Combining different n_{swaps} values

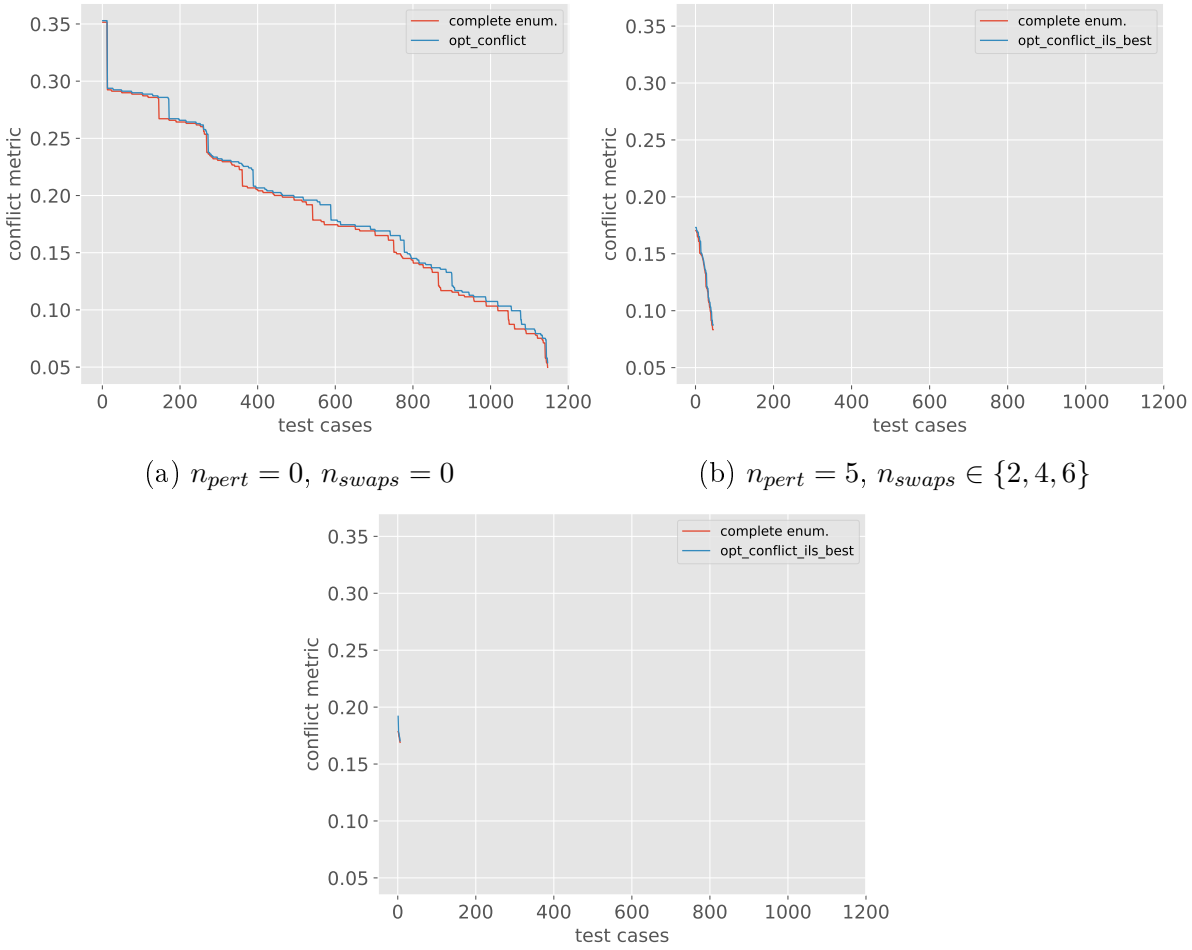
To further explore the limits of our ILS-based heuristics, we have implemented two additional ILS variants, namely opt_conflict_ils_best and opt_fairness_ils_best, where we take a list of different parameter combinations (n_{swaps}, n_{pert}), run our ILS approach for every combination and take the best result w.r.t. the chosen optimization metric, e.g. $n_{swaps} \in \{2, 4, 6\}$ and $n_{pert} = 10$. As seen in Fig. 3.23 and 3.24a, the end results are quite impressive. In Fig. 3.23c, for instance, we can barely see any divergence between the optimal curve and opt_conflict_ils_best (only 6 test cases displayed an absolute difference to the global optima that equaled or surpassed 0.001). This is further confirmed by the similarity metrics shown in Tab. 3.11. We can see, for instance, that ϕ_{MSE} got better by an order of magnitude when compared to all attempts where only a single number of swaps was used. A similar, if only not as strong, improvement was also observed with opt_fairness_ils_best, as shown in Tab. 3.12 and Fig. 3.24.

similarity	opt_conflict	$n_{swaps} = 2$	$n_{swaps} = 4$	$n_{swaps} = 6$	ils_best
$\phi_{MSE} \cdot 10^6$	77.948	6.265	3.7456	5.9899	0.4473
ϕ_g	0.4218	0.7386	0.7796	0.7848	0.8479
ϕ_{quasi}	0.9085	0.9864	0.9920	0.9889	0.9987

Table 3.11: Comparison of opt_conflict with opt_conflict_ils and opt_conflict_ils_best for $n_{pert} = 10$ and $n_{swaps} \in \{2, 4, 6\}$.

metric	opt_fairness	$n_{swaps} = 2$	$n_{swaps} = 4$	$n_{swaps} = 6$	ils_best
$\phi_{MSE} \cdot 10^6$	77.948	2.8052	4.7031	5.285	1.3375
ϕ_g	0.4218	0.9840	0.9819	0.977	0.9920
ϕ_{quasi}	0.9085	0.9857	0.9840	0.9788	0.9926

Table 3.12: Comparison of opt_fairness with opt_fairness_ils and opt_fairness_ils_best with $n_{pert} = 10$ and $n_{swaps} \in \{2, 4, 6\}$.

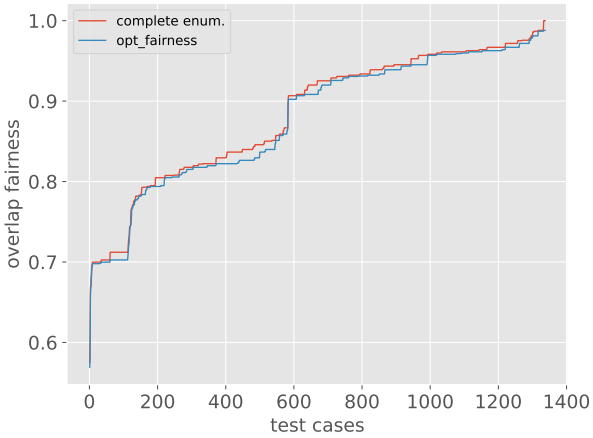


(a) $n_{pert} = 0$, $n_{swaps} = 0$

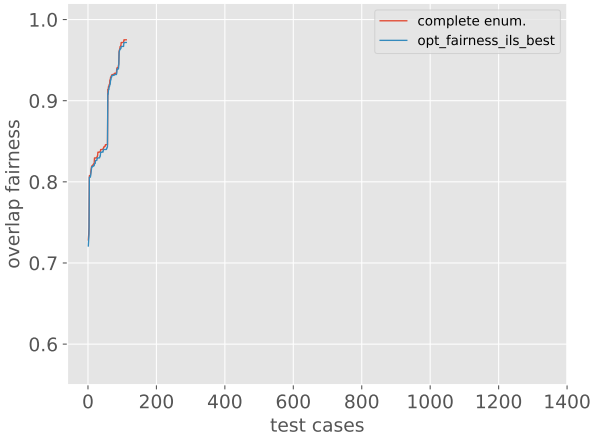
(b) $n_{pert} = 5$, $n_{swaps} \in \{2, 4, 6\}$

(c) $n_{pert} = 10$, $n_{swaps} \in \{2, 4, 6\}$

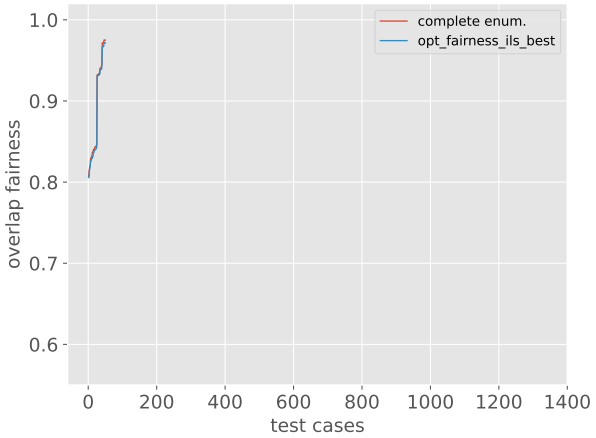
Figure 3.23: Comparison of basic opt_conflict_ils and opt_conflict_ils_best with $n_{swaps} \in \{2, 4, 6\}$ and $n_{pert} = 10$ with absolute difference between local and global optimal greater or equal to 0.001.



(a) $n_{pert} = 0$, $n_{swaps} = 0$



(b) $n_{pert} = 5$, $n_{swaps} \in \{2, 4, 6\}$



(c) $n_{pert} = 10$, $n_{swaps} \in \{2, 4, 6\}$

Figure 3.24: Comparison of basic opt_fairness_ils and opt_fairness_ils_best with $n_{swaps} \in \{2, 4, 6\}$ and $n_{pert} = 10$ with absolute difference between local and global optimal greater or equal to 0.001.

3.6 Solution constraints w.r.t. primary conflicts

In this section, we derive some constraints, delivered through a collection of theorems and associated proofs, regarding the solution of primary conflicts between communication and sensing schedules.

Theorem 3.6.1. *If a sensing schedule is constructed by our opt_conflict heuristic, any primary conflicts present in the schedule are primary conflicts of the same channel.*

Proof. For one remaining primary conflict, our property holds trivially. To prove it holds for $n > 1$ primary conflicts we will use a proof by induction with a *reductio ad absurdum* argument.

1. First, assume our theorem holds for n primary conflicts, i.e. all conflicts are of channel c_i. Then, we have to show it holds for $n + 1$.
2. If we add another conflict of the same channel, our theorem holds trivially.
3. On the other hand, if we assume the $(n + 1)$-th *unsolvable* primary conflict is of channel $c_j \neq c_i$, we can switch one of the primary conflicts of c_i with c_j. This clearly lowers the conflict metric and solves both *unsolvable* primary conflicts of c_i and c_j, which is a contradiction, ergo all unsolvable primary conflicts are of the same channel.

□

Definition 3.6.1. Let $n(c, s)$ denote the number of slots in which channel c occurs in schedule s. We define the usage $u(c, s)$ of channel c in schedule s as the fraction of the total slots in s in which channel c appears, i.e.

$$u(c, s) = \frac{n(c, s)}{|s|}$$

From the definition it follows $\sum_{c \in C} u(c, s) = 1$.

Theorem 3.6.2. *Given a a set of n channels $C = \{ c_1, \ldots, c_n \}$, a communication schedule s'_{com} and a balanced seed sensing schedule $s^{(0)\prime}_{sens}$, aligned with s'_{com}, we can derive a sensing schedule s'_{sens}, primary-conflict-free w.r.t. s'_{com}, if and only if*

$$\forall c \in C : \; u(c, s'_{com}) \leq \frac{n - 1}{n} \tag{3.7}$$

Example 3.6.1. Given a set of channels $C = \{ c_1, c_2, c_3 \}$, a communication schedule $s'_{com} = [c_1,\ c_2,\ c_1]$ aligned with a seed sensing schedule $s^{(0)\prime}_{sens} = [c_1, c_2, c_3]$, it is possible to solve all primary conflicts as shown below:

$$s'_{com} = [c_1,\ c_2,\ c_1]$$
$$s'_{sens} = [c_2,\ c_1,\ c_3]$$

In fact, we have

$$u(c_1, s'_{com}) = \frac{2}{3}$$
$$u(c_2, s'_{com})) = \frac{1}{3}$$
$$u(c_3, s'_{com}) = 0$$

This means, all channels $c \in \{c_1, c_2, c_3\}$ clearly satisfy (3.7), i.e.

$$\forall c \in C.\, u(c, s'_{com}) \leq \frac{2}{3}$$

Proof. We will start by proving that (3.7) is *necessary* for eliminating all primary conflicts. Afterwards, we will show that (3.7) is also *sufficient* to guarantee that we can solve all primary conflicts.

In Tab. 3.13, we show a summary of some relevant notation and expressions that will appear during our proof.

Expression	Description
n_{slots}	$\lvert s'_{com} \rvert$
$u(c, s'_{com}) \cdot n_{slots}$	Number of slots in s'_{com} where channel c occurs
$(1 - \frac{1}{n}) \cdot n_{slots}$	Number of slots in s'_{sens} where c does not occur

Table 3.13: Notation and expressions relevant for proving Theorem 3.6.2.

For our argument, we will use the Dirichlet's box principle, also known as the pigeonhole principle. This principle though quite simple to grasp is a very powerful tool that finds applications throughout combinatorial mathematics and number theory. It

basically states that if we want to distribute $(n+1)$ objects into n boxes then at least one box is filled by two or more objects.

Given $c \in C$ a channel with primary conflicts, we assume the number of slots in s'_{com} where c occurs is greater than the number of slots in s'_{sens} where c does not occur. This means that to eliminate all primary conflicts, we need at least one slot in s'_{sens}, where a channel $c^\star \neq c$ occurs, to be aligned with two slots in s'_{com} where c occurs, which is impossible. This means that under the assumed conditions, at least one slot in s'_{com} and s'_{sens} displays a primary conflict of channel c. Hence, for us to solve all primary conflicts of channel c it is *necessary* that the number of occurrences of c in s'_{com} to be the *at most* the total number of slots in which other channels occur in s'_{sens}, i.e.

$$\forall c \in C: \; u(c, s'_{com}) \cdot n_{slots} \leq (1 - \frac{1}{n}) \cdot n_{slots} \implies$$

$$\forall c \in C: \; u(c, s'_{com}) \leq 1 - \frac{1}{n} \implies$$

$$\forall c \in C: \; u(c, s'_{com}) \leq \frac{n-1}{n}$$

To prove that (3.7) is not only *necessary* but also *sufficient* to construct primary-conflict-free sensing schedules, we will use the famous Hall's Marriage Theorem.

For our proof, we will make use of the combinatorial formulation of the Marriage theorem that states that each channel $s'_{sens}[k]$ at slot k can happily marry a channel $s'_{com}[k]$ (in our case without a primary conflict) if and only if for any subset I of slots in s'_{sens}, the number of channels in s_{com} that can marry at least one of the channels in the slots in I is at least as large as the size of I.

Let $C^\star_{optimal}(s'_{com}, k)$ be the multiset containing all $(1-u(s'_{com}[k], s'_{sens})) \cdot n_{slots}$ instances of every channel $c \neq s'_{com}[k]$ in s'_{sens}. These are all instances of the channels that can be placed at slot k in s'_{sens} without creating a primary conflict.

The marriage theorem then states that

$$\forall I \in 2^{\{1,\ldots,n_{slots}\}}: \; |I| \leq |\bigcup_{k \in I} C^\star_{optimal}(s'_{com}, k)| \tag{3.8}$$

Put in other words, this means that if the size of the union of the lists of suitable groom candidates is at least as large as the number of brides (for any subset of the brides), then there is a perfect matching.

Therefore, we have to show that

$$\forall I \in 2^{\{1,\ldots,n_{slots}\}}\ \forall c \in C:\ u(c, s'_{com}) \leq \frac{n-1}{n} \implies |I| \leq |\bigcup_{k\in I} C^{\star}_{optimal}(s'_{com}, k)| \quad (3.9)$$

To prove (3.9), we will divide our proof into two cases:

1. At least two of the channels $s'_{com}[k_1]$ and $s'_{com}[k_2]$ for $k_1, k_2 \in I$ are distinct. This implies

   $$|\bigcup_{k\in I} C^{\star}_{optimal}(s'_{com}, k)| = n_{slots}$$

   As the subset of slots I has at most all n_{slots} slots

   $$|I| \leq n_{slots} \implies |I| \leq |\bigcup_{k\in I} C^{\star}_{optimal}(s'_{com}, k)|$$

2. All channels at slot k in s'_{com} are the same, i.e. $\forall k \in I$: $s'_{com}[k] = c \implies$

   $$\begin{aligned} |\bigcup_{k\in I} C^{\star}_{optimal}(s'_{com}, k)| &= |C^{\star}_{optimal}(s'_{com}, k)| \\ &= n_{slots} \cdot (1 - u(c, s'_{sens})) \\ &= n_{slots} \cdot (1 - \frac{1}{n}) \end{aligned}$$

   $$\begin{aligned} u(c, s'_{com}) \leq \frac{n-1}{n} &\implies u(c, s'_{com}) \cdot n_{slots} \leq (1 - \frac{1}{n}) \cdot n_{slots} \\ &\implies u(c, s'_{com}) \cdot n_{slots} \leq |\bigcup_{k\in I} C^{\star}_{optimal}(s'_{com}, k)| \end{aligned}$$

   Furthermore, since the slots in I have at most all instances of c in s'_{com}, i.e.

   $$|I| \leq u(c, s'_{com}) \cdot n_{slots} \implies |I| \leq |\bigcup_{k\in I} C^{\star}_{optimal}(s'_{com}, k)|$$

□

I addition, by using a similar argumentation it is not hard to prove that in the general case, where s'_{sens} can be unbalanced, for all primary conflicts between s'_{sens} and s'_{com} to be eliminated it is necessary and sufficient that

$$\forall c \in C: \; u(c, s'_{com}) + u(c, s'_{sens}) \leq 1$$

Theorem 3.6.3. *Given a communication schedule s'_{com} aligned with a seed sensing schedule $s^{(0)\prime}_{sens}$ and a set of channels $C = \{c_1, \ldots, c_n\}$, it is possible to construct a sensing schedule s'_{sens} primary-conflict-free w.r.t. s'_{com} if and only if*

$$\forall c \in C: \; u(c, s'_{com}) + u(c, s'_{sens}) \leq 1 \tag{3.10}$$

Proof. Similarly to the previous proof, we first prove that (3.10) is *necessary* to guarantee that all primary conflicts are solvable and then show that it is also *sufficient*. Some relevant expressions to our proof are shown in Table 3.14.

Expression	Description
n_{slots}	Number of slots in s'_{sens} and s'_{com}
$u(c, s'_{com}) \cdot n_{slots}$	Number of slots in s'_{com} that have channel c
$(1 - u(c, s'_{sens})) \cdot n_{slots}$	Number of slots in s'_{sens} that do not have c

Table 3.14: Notation and equivalences relevant for the proof of Theorem 3.6.3.

Again using Dirichlet's drawer principle, it follows that the number of slots in which channel c occurs in s'_{com} has to be at most the number of slots in which other channels appear in s'_{sens}, i.e.

$$\forall c \in C: \; u(c, s'_{com}) \cdot n_{slots} \leq (1 - u(c, s'_{sens})) \cdot n_{slots} \implies$$

$$\forall c \in C: \; u(c, s'_{com}) \leq 1 - u(c, s'_{sens}) \implies$$

$$\forall c \in C: \; u(c, s'_{com}) + u(c, s'_{sens}) \leq 1$$

To prove that (3.10) is not only *necessary* but also *sufficient*, we will again use the marriage theorem.

Again, we let $C^{\star}_{optimal}(s'_{com}, k)$ denote the multiset containing all $(1-u(s'_{com}[k], s'_{sens}))\cdot n_{slots}$ instances of channels in $s^{(0)\prime}_{sens}$ that are distinct from $s'_{com}[k]$. These are all the channel instances that could be placed at slot k without creating a primary conflict with the given communication schedule.

So this time we want to show that

$$\forall I \in 2^{\{1,\ldots,n_{slots}\}}\ \forall c \in C:\ u(c, s'_{com}) + u(c, s'_{sens}) \leq 1 \implies |I| \leq |\bigcup_{k\in I} C^{\star}_{optimal}(s'_{com}, k)| \tag{3.11}$$

We can divide our analysis in two cases:

1. In the slots $k \in I$, not all channels are the same which implies

$$|\bigcup_{k\in I} C^{\star}_{optimal}(s'_{com}, k)| = n_{slots}$$

Since I has at most all n_{slots} slots

$$|I| \leq n_{slots} \implies |I| \leq |\bigcup_{k\in I} C^{\star}_{optimal}(s'_{com}, k)|$$

2. $\forall k \in I:\ s'_{com}[k] = c \implies$

$$\begin{aligned}|\bigcup_{k\in I} C^{\star}_{optimal}(s'_{com}, k)| &= |C^{\star}_{optimal}(s'_{com}, k)| \\ &= n_{slots}\cdot(1-u(c, s'_{sens}))\end{aligned}$$

$$\begin{aligned}u(c, s'_{com}) + u(c, s'_{sens}) \leq 1 &\implies u(c, s'_{com})\cdot n_{slots} \leq (1-u(c, s'_{sens}))\cdot n_{slots} \\ &\implies u(c, s'_{com})\cdot n_{slots} \leq |\bigcup_{k\in I} C^{\star}_{optimal}(s'_{com}, k)|\end{aligned}$$

Finally, as I has at most all instances of c in s'_{com}, i.e.

$$|I| \leq u(c, s'_{com})\cdot n_{slots} \implies |I| \leq |\bigcup_{k\in I} C^{\star}_{optimal}(s'_{com}, k)|$$

□

3.7 Matching - a graph theoretical formulation

In this section, we will describe how matching theory can help describe and at least partially solve the problem of constructing high-quality sensing schedules.

Matching is a recurring problem in different disciplines such as combinatorics, operations research and mathematical computing, which basically involves some sort of pairing of objects. Usually, this pairing tries to minimize associated costs (or maximize profits).

3.7.1 Fundamentals

First, we start with some definitions that will help us formulate our optimization problem in graph-theoretical terms.

Definition 3.7.1. (Bipartite graph). A graph is bipartite if its set of vertices can be partitioned into two sets such that every edge connects one vertex in A to a vertex in B.

A and B are called the parts or the color classes of bipartite graph G, also called a 2-colorable graph or simply a bigraph.

Definition 3.7.2. (Matching). Let $G = (V, E, w)$ be an undirected weighted graph, a matching is a subset of the edges $M \subseteq E$ where no two edges in M are incident to the same vertex $v \in V$.

Definition 3.7.3. (Perfect matching). Given undirected weighted graph $G = (V, E, w)$, a *perfect* matching is a matching M where *exactly one* edge in M is incident to *every* vertex $v \in V$.

In its graph theoretical formulation, the Marriage Theorem states that a bipartite graph $G = (A \cup B, E)$ has a perfect matching, if and only if $|A| = |B|$ and for every subset $A^\star \subseteq A, |N_G(A^\star)| \geq |A^\star|$ [LP09], where $N_G(A^\star)$ denotes the the set of all vertices in B adjacent to at least one element of $A^\star$. Here adjacent means that there is an edge connecting one vertex in A to one vertex in B.

Historically, this formulation of the Marriage Theorem was introduced by Frobenius, but there were multiple theorems proved by different mathematicians that deliver equivalent results, among them Hall's theorem and König's Minimax Theorem [LP09].

Theorem 3.7.1. *The problem of deciding whether a conflict-free sensing schedule can be constructed can be reduced to the decision problem of determining whether a perfect matching exists.*

Proof. Construct a graph $G = (V_{com} \cup V_{sens}, E)$ where the vertices in V_{com} are all instances of all channels that occur in s'_{com} and V_{sens} contains all instances of all channels that occur in $s^{(0)\prime}_{sens}$. In addition, for each channel $c_i \in V_{com}$ and each channel $c_j \in V_{sens}$, add an edge (c_i, c_j) to E if and only if $r_{overlap}(c_i, c_j) = 0$, i.e. there is no overlap between the communication and sensed channel. This way, it is easy to see that if a perfect matching M exists in G, then we can trivially construct a conflict-free sensing schedule, a permutation of $s^{(0)\prime}_{sens}$, in which every edge $(c_i, c_j) \in M$ determines that $c_j \in V_{sens}$ must be placed in the same slot at which $c_i \in V_{com}$ occurs such that all aligned channels have no overlap. □

[Tut47] was the first work to demonstrate that the perfect matching decision problem is in NP ∩ co-NP, i.e. both the existence and the non-existence of a perfect matching in a graph can be verified in polynomial time. Moreover, in [Lá79] Lovász's introduced a perfect matching decision algorithm which can be used to determine with high probability whether a bipartite graph G with two equal parts has a perfect matching or not. Given a bipartite graph $G = (A \cup B, E, w)$ with $n = |A| = |B|$, consider a $n \times n$ matrix I such that

$$I = \begin{cases} x_{a,b} & \text{if } (a,b) \in E \\ 0 & \text{if } otherwise \end{cases}$$

where $x_{a,b}$ are variables associated to each edge (a, b). Lovaśz's algorithm works by substituting every variable $x_{a,b}$ in I by random numbers. If G does not have a perfect matching then the determinant $det(I)$, which is a polynomial in variables $x_{a,b}$ for $(a, b) \in E$, is identically zero. The algorithm works by making multiple independent runs, since its error probability 4^{-t} decreases exponentially with the number t of runs. The Lovász algorithm runs in randomized time $\mathcal{O}(n^\omega)$ with $\omega < 2.38$ for a sequential execution or in $\mathcal{O}((\log n)^2)$ when executed in parallel using multiple processors [Che97].

Definition 3.7.4. (Minimum weight perfect matching). Given $G = (V, E, w)$ an undirected weighted graph, where w_e denotes the weights of each edge $e \in E$, we define the the minimum weight perfect matching problem as finding a perfect matching M such that the weight of the matching, i.e. the sum of all weights of the matching $\sum_{e \in M} w_e$, is minimal.

Interestingly, the weighted perfect matching problem is closely related to the already mentioned Travelling Salesman Problem, but the latter is a NP-hard problem and significantly more difficult to solve than the former. In fact, the weighted perfect matching can be seen as a relaxation of TSP [LP09]. It is also a known fact in matching theory

that a greedy approach alone such as the one used for our basic heuristics or such as the greedy algorithm proposed by Kruskal [Kru56] to find the shortest spanning subtree of a graph do not always find optimal solutions for the minimum weight perfect matching problem, because while spanning subtrees form a *matroid* (mathematical objects used to generalize linear independence between vectors), perfect matchings do not [LP09].

Theorem 3.7.2. *Creating a conflict-minimal sensing schedule s'_{sens} aligned with communication schedule s'_{com} reduces to the minimum weight perfect matching problem.*

Proof. Construct a graph $G = (V_{com} \bigcup V_{sens}, E, w)$ where the vertices in V_{com} are all instances of all channels that occur in s'_{com} and V_{sens} contains all instances of all channels that occur in $s^{(0)\prime}_{sens}$. In addition, E contains $|s'_{com}| \cdot |s^{(0)\prime}_{sens}|$ edges that connect all vertices from V_{com} with all vertices from V_{sens} and the weight for each edge $e = (c_i, c_j) \in E$ is $w_e = r_{overlap}(c_i, c_j)$.

This way, finding a conflict-minimal sensing schedule s'_{sens}, a permutation of $s^{(0)\prime}_{sens}$, aligned with a communication schedule s'_{com} that minimizes

$$\sum_{1 \leq k \leq |s'_{com}|} r_{overlap}(s'_{sens}[k], s'_{com}[k])$$

is equivalent to finding a perfect matching M in G that minimizes

$$\sum_{e \in M} w_e = \sum_{(c_i, c_j) \in M} r_{overlap}(c_i, c_j)$$

And this is by definition the minimum weight perfect matching problem. □

Fig. 3.25 illustrates the equivalence between a conflict-minimal sensing schedule s_{sens} w.r.t. s_{com} and a minimum weight perfect matching in a bipartite graph with color classes V_{com} and V_{sens}.

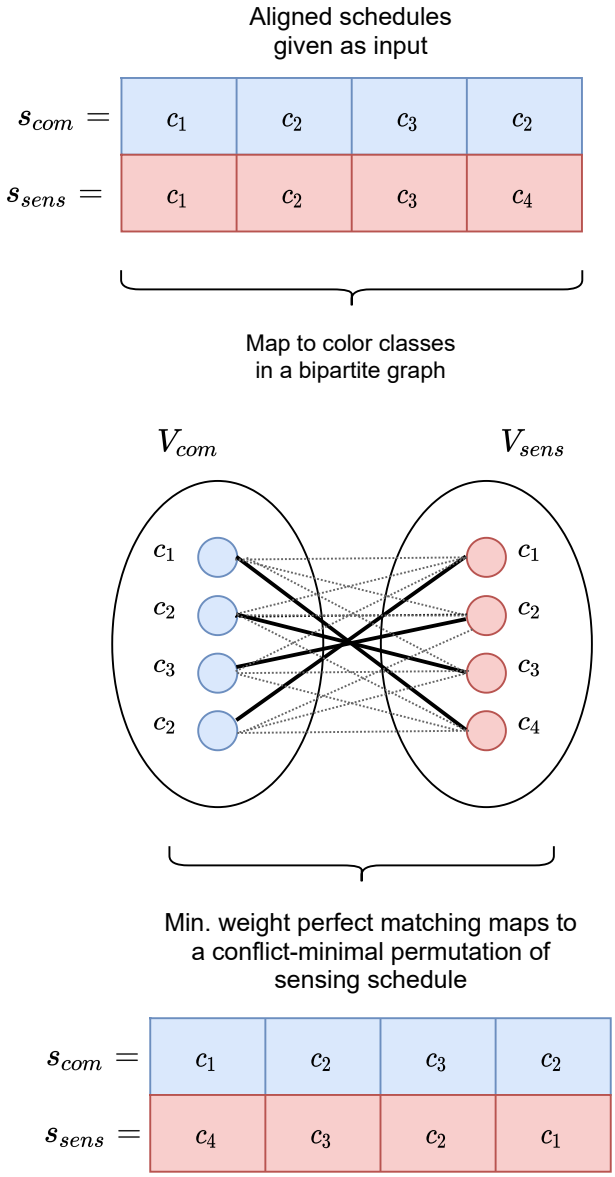


Figure 3.25: Illustration showing the mapping between aligned communication and sensing schedules and a perfect matching. Note that the weight of the edges are omitted for better visualization.

3.7.2 Finding minimum weight perfect matchings

To solve the minimum weight perfect matching problem in bipartite graphs, Kuhn [Kuh55] originally proposed the Hungarian Method, which was later proposed in different formulations and further refined by multiple research works such as [Mun57]. The Hungarian method is able to *exactly* solve the minimum weight perfect matching for bipartite graphs. For an undirected weighted bipartite graph $G = (V, E, w)$, the Hun-

garian Method can be implemented with a worst-case time complexity of $\mathcal{O}(|V|^3)$ in contrast to a time complexity of $\mathcal{O}(|V|^2)$ of our local search heuristics.

This means, if we apply this algorithm to the construction problem of sensing schedules, we can obtain global optima w.r.t. conflict metric. However, nothing comes for free. Even though, the Hungarian Method is able to always find the minimum for the conflict metric, it has a higher time complexity than our proposed heuristics (which could prove problematic w.r.t. to execution time with larger sizes of aligned schedules) and is in general more complex, both for understanding as well as implementing and testing. Furthermore, while it can easily optimize conflict metric, it is not able to optimize overlap fairness or the combination of conflict metric and overlap fairness.

The first important concept that we need to explain the Hungarian Method is that of a feasible labeling. A labeling, in general, associates labels (or weights) to every vertex in a graph.

Definition 3.7.5. (Feasible labeling). Given a bipartite graph $G = (A \cup B, E, w)$, and weights $w(a, b)$ assigned to each edge $(a, b) \in E$, the labeling of the graph is feasible if

$$l(a) + l(b) \geq w(a, b) \, \forall a \in A, \forall b \in B \tag{3.12}$$

where $l(a)$ and $l(b)$ are respectively the labels of vertices a and b.

Example 3.7.1. Given a bipartite graph $G = (A \cup B, E, w)$ (see Fig. 3.26) with parts A and B and adjacency matrix W whose entries are the weights of all edges as shown below

$$W = \begin{array}{c} \\ B_1 \\ B_2 \\ B_3 \end{array} \begin{array}{c} \begin{array}{ccc} A_1 & A_2 & A_3 \end{array} \\ \begin{pmatrix} 3 & 2 & 3 \\ 1 & 2 & 0 \\ 3 & 2 & 1 \end{pmatrix} \end{array}$$

In this scenario, a possible feasible labeling is

$$l(A_1) = 2$$
$$l(A_2) = 1$$
$$l(A_3) = 2$$
$$l(B_1) = 1$$
$$l(B_2) = 1$$
$$l(B_3) = 1$$

The chosen labeling is feasible, since it guarantees that the weight of every edge never surpasses the value of the sum of the labels of the connected vertices.

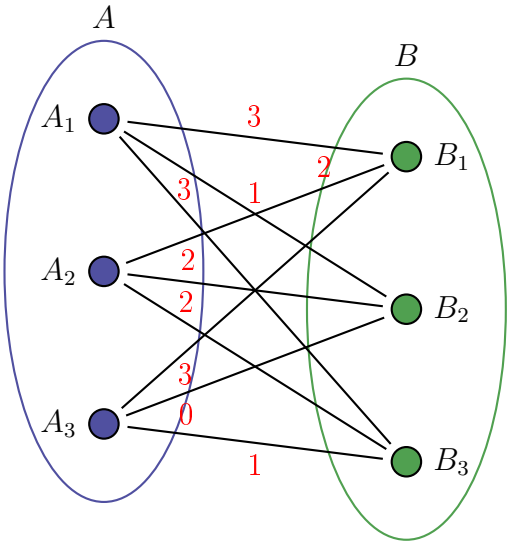


Figure 3.26: Illustration of a bipartite graph G with parts $A = \{c_1, c_2, c_3\}$ and $B = \{c_6, c_7, c_8\}$, where weight of each edge is shown in red.

In addition to the concept of a feasible labeling, we need to define what alternating and augmenting paths are.

Definition 3.7.6. (Alternating path). Given graph $G = (V, E)$ and a matching M, an alternating path $P = v_1, v_2, \ldots, v_m$ is a subset of E such that $(v_i, v_{i+1}) \in M$ and $(v_{i+1}, v_{i+2}) \notin M$, i.e. the edges in the path *alternate* between being in the matching and not being in the matching.

In Fig. 3.27 we show an example of a graph and a matching M (red edges) with an alternating path

$$P = B_1, A_1, B_2, A_2, B3, A_4, B_4, A_3, B_5, A_5$$

where vertices are connected by red and green edges in alternating fashion.

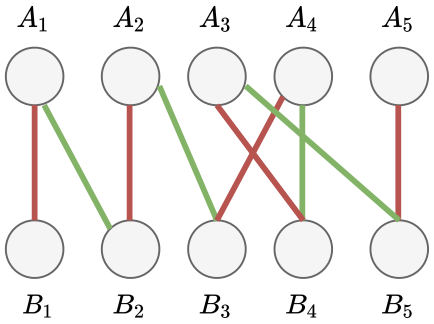


Figure 3.27: Illustration of an alternating path composed of edges in *perfect* matching M (red edges) and edges not in M (green edges) starting with B_1 and ending with A_5.

Definition 3.7.7. (Augmenting path). Given graph $G = (V, E)$ with perfect matching M, an augmenting path is an alternating path $P = v_1, v_2, \ldots, v_m$ whose endpoints v_1 and v_m are not included in M.

In Fig. 3.28 we show an example of a graph and a matching M (red edges) with an augmenting path

$$P = B_1, A_1, B_2, A_2, B3, A_3, B_5, A_5$$

where vertices are connected by red and green edges in alternating fashion. Note that P starts and ends with green edges such that both endpoints B_1 and A_5 are not in M.

Definition 3.7.8. (Equality graph). Given a graph $G = (A \cup B, E, w)$, we define the equality graph $G_l = (A \cup B, E_l, w) \subseteq G$, where $E_l \subseteq E$ is such that

$$E_l = \{(a, b) \in E \mid l(a) + l(b) = w(a, b)\}$$

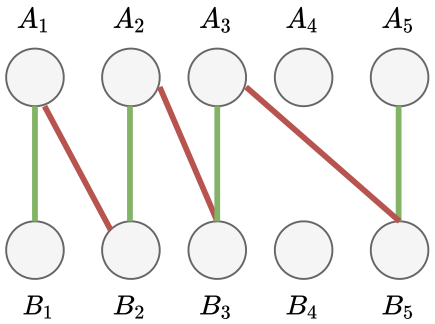


Figure 3.28: Illustration of an augmenting path composed of alternating edges in matching M (red edges) and edges not in M (green edges) starting with $B_1 \notin M$ and ending with $A_5 \notin M$.

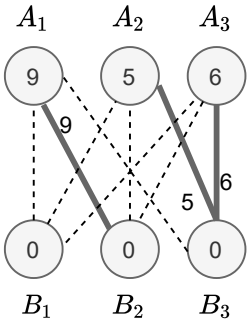


Figure 3.29: Graph $G = (V, E)$ and equality graph $G_l = (V, E_l)$. The dashed edges are the edges in E that are not in E_l.

Theorem 3.7.3. *(Kuhn-Munkres Theorem). Given a feasible labeling l, if M is a perfect matching on the equality graph G_l, then M is a minimum (or maximum) weight perfect matching in G.*

By using the Kuhn-Munkres Theorem, we can reduce the problem of finding an optimal weight perfect matching to finding the right labeling l and with it a perfect matching in the associated equality graph G_l. Fig.3.29 displays an example of an equality graph.

Before describing the Hungarian Method, we will briefly describe its two main building blocks: the augmenting procedure and the label improvement.

The augmenting procedure

Given an equality graph $G_l = (A \cup B, E_l, w)$ derived from graph $G = (A \cup B, E, w)$ and labeling l, the augmenting procedure attempts to enlarge a given matching M in G_l. Ideally, this matching will eventually be perfect (all vertices are matched), which

means we found the minimum weight perfect matching in the original graph G. This procedure begins with an empty matching M, and starts building an alternating path from any unmatched vertex u trying to make the path augmenting. Either the matching associated with this alternating path becomes perfect or the procedure runs out of edges in G_l to add to the growing alternating path. If the procedure runs out of edges, it means we cannot enlarge the current matching anymore. To get out of this impasse, we first improve the used labeling and then enlarge E_l accordingly.

Improving the labeling

We denote the sets S and T as the set of vertices in respectively A and B which were already added to the alternating path by the last run of the augmenting procedure and define

$$N_l(a) = \{b \mid (a,b) \in E_l\}$$

$$N_l(S) = \bigcup_{a \in S} N_l(a)$$

This means the label improvement procedure is always activated when $N_l(S) = T$. After improving the labeling, we enlarge E_l such that $N_l(S) \neq T$ and the augmenting procedure can continue.

Given $\delta(l)$ the minimum value of $l(a) + l(b) - w(a,b)$ for all $a \in S$ and $b \in B$ but $b \notin T$, we improve the labeling l to l' (also a feasible labeling) as follows:

$$l'(v) = \begin{cases} l(v) - \delta(l) & \text{if } v \in S \\ l(v) + \delta(l) & \text{if } v \in T \\ l(v) & \text{if } c \notin S \wedge c \notin T \end{cases} \tag{3.13}$$

We then update the equality graph by modifying its set of edges:

$$E_{l'} = E_l \cup \{(a,b) \mid (a,b) \in E \wedge l'(a) + l'(b) = w(a,b)\} \tag{3.14}$$

With all the pieces in place, we finally describe the Hungarian Method.

The Hungarian Method

Input: a graph $G = (A \cup B, E, w)$, a matching $M = \{\}$.

1. First, we negate the weights, i.e. $\forall (a,b) \in E. w(a,b) = -1 \cdot w(a,b)$. This way, finding the perfect matching that maximizes the negated weights of the matching

is equivalent to finding the minimum weight perfect matching with the original weights.

2. Apply a a feasible labeling l to G: label each vertice on one part of the bipartite graph with the maximum weight of all edges incident with it. Label the vertices on the other part of G with 0. Formally,

$$\forall a \in A \forall b \in B. l(a) = \max_{b \in B} w(a, b), l(b) = 0$$

This labeling is feasible by construction since $l(a) + l(b) \geq w(a, b)$ for all edges (a, b).

3. Construct an equality graph $G = (A \cup B, E_l, w)$ for the given feasible labeling l.

4. Start the augmenting procedure on G_l and improve the current matching M.

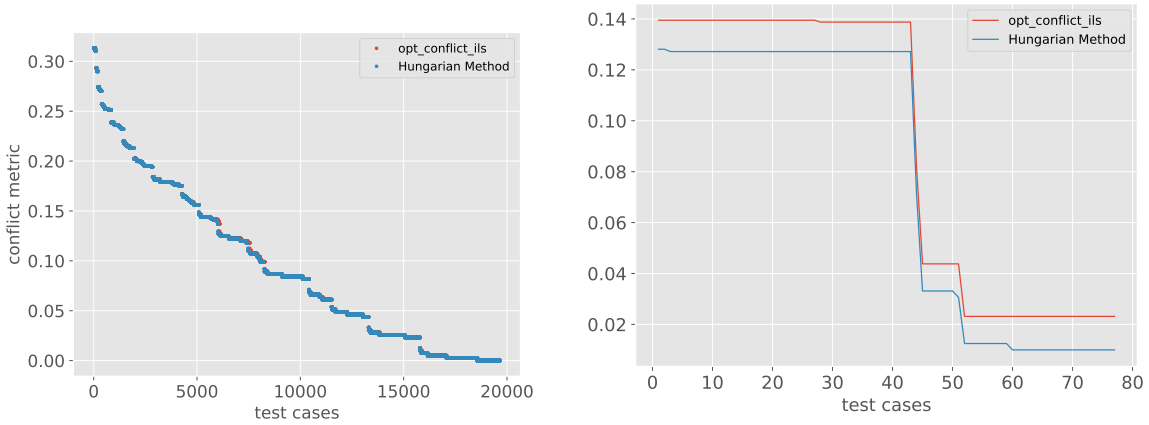
5. If M is perfect, then we are done and this matching is the perfect matching with maximum negated weights, i.e. the minimum weight perfect matching.

6. If M is not perfect, improve the used labeling and go back to step 4.

For brevity, we will leave out implementation details regarding the augmenting procedure, but it suffices to say that naive implementations of the Hungarian Method deliver a time complexity of $\mathcal{O}(|A \cup B|^4))$, and proper use of techniques such as breadth-first search and appropriate data structures to avoid unnecessarily re-visiting vertices during the augmenting procedure can lower the time complexity down to $\mathcal{O}(|A \cup B|^3))$.

3.7.3 Comparison with our heuristics

In a similar fashion to the test sets constructed for comparison with a complete enumeration, we have created a test set with 19669 test cases where $|s'_{com}| = |s^{(0)\prime}_{sens}| = 13$. In Fig. 3.30a, we can see that the performance of opt_conflict_ils is really good when compared to the conflict-minimal solutions delivered by the Hungarian Method. For a better visualization of the divergences between both methods, we show in Fig. 3.30b only the cases in which the absolute difference in conflict metric value was greater than 0.01. There are 77 such cases and as we can see all absolute differences stay below 0.02, which further demonstrates the high quality of the sensing schedules delivered by our heuristic.

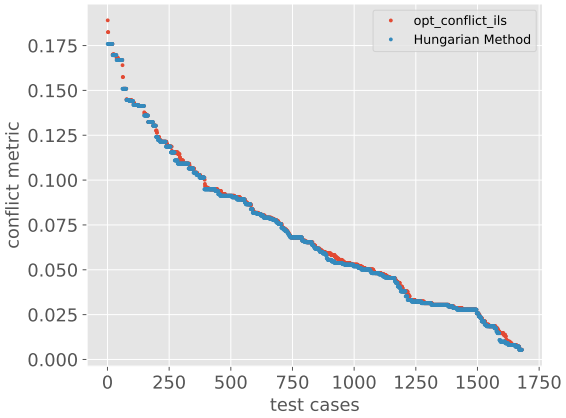


(a) 19669 test cases.

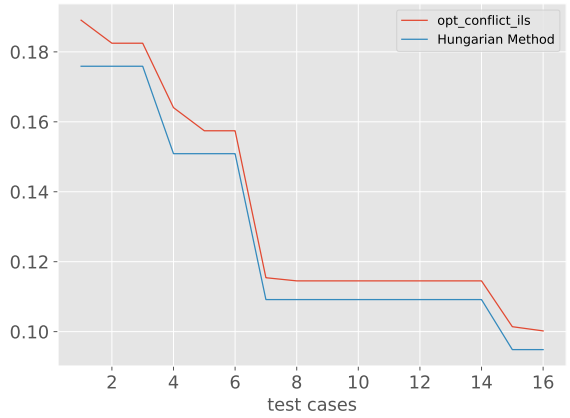
(b) Absolute difference between global and local optima equals 0.01 or higher.

Figure 3.30: Comparison of opt_conflict_ils with the Hungarian Method on test set with 19669 cases where $|s_{com}| = |s^{(0)\prime}_{sens}| = 13$. On the right, we show only the 77 test cases where the absolute difference between global and local optima is greater than or equals 0.01. This means 99.6% of all test cases display a difference of less than 0.01 in conflict metric.

In addition, we have generated another testset using $s^{(0)}_{sens} = [c_1, c_2, c_3, \ldots, c_{12}]$ and communication schedules s_{com} as permutations of all combinations of 4 channels $c_i \in [c_3, c_4, c_5, \ldots, c_{10}]$ with utilizations $\vec{u} = (2, 1, 3, 3)$. Since $|s_{com}| = 9$ and $|s^{(0)}_{sens}| = 12$, the size of the aligned schedules is $|s'_{com}| = |s^{(0)\prime}_{sens}| = lcm(9, 12) = 36$. As seen in Fig. 3.31a, in all 1680 test cases opt_conflict_ils is again capable of constructing sensing schedules that are either conflict-minimal or very close to the global optima. In Fig. 3.31b, we see that only 16 test cases had an absolute difference greater than 0.005 in conflict metric. This means that more than 99% of all test cases display a difference of less than 0.005 in conflict metric.



(a) 1680 test cases.



(b) Test cases where absolute different between global and local optima equals 0.005 or higher.

Figure 3.31: Comparison of opt_conflict_ils with the Hungarian Method on test set with 1680 cases where $|s_{com}| = |s^{(0)\prime}_{sens}| = 36$. On the right, we show only the 16 test cases where the absolute difference between global and local optima is greater than or equals 0.005.

3.8 Summary

In this chapter, we proposed and implemented a method for the construction of high-quality local sensing schedules in which nodes compute their sensing order based on a communication schedule. The resulting sensing schedule is balanced, and if constructed by the opt_conflict heuristic tries to achieve conflict-minimality as well as maximize overlap fairness when possible through local-search-based heuristics. An alternative heuristic, namely opt_fairness, also constructs balances schedules but prioritizes maximizing the overlap fairness over minimizing the conflict metric. These heuristics, while not able to always deliver optimal solutions, accomplished impressive results, especially with the fine-tuned iterated local search variants. Our ILS approach embeds our basic opt_conflict and opt_fairness heuristics in a stochastic local search-based heuristic that uses perturbations and randomness to improve local optima. Minimizing channel overlap between the sensing and communication schedules tries to minimize the effect of internal transmissions in the q_{cbt} channel quality metric, allowing nodes to derive channel quality primarily based on foreign traffic. Moreover, making sensing schedules balanced and trying to improve overlap fairness when possible, minimizes bias in the sensing process. Finally, we have shown that the problem of constructing conflict-minimal sensing schedules can solved by the Hungarian Method by reducing it to the problem of finding

a weighted perfect matching in a bipartite graph (whose parts are derived from our sensing and communication schedules). This method, however, while exact displays a worse asymptotic time complexity than our opt_conflict_ils_best heuristic, which is able to approximate the optimal results remarkably well.

Long term consistency trumps short term intensity.

— Bruce Lee

4

Three-Dimensional Stabilization

Contents

As described in Chapter 1, alongside channel sensing the other main component we need for achieving a robust utilization of the available radio spectrum is channel hopping. As described in Chapter 2, time is divided into time slots in our network and in each slot a single channel is used.

We keep the operation of our network stable by using the volatility-aware techniques introduced in Chapter 2 and by using a subset of all available channels in which a minimum acceptable quality is reached. Moreover, nodes proactively hop to different channels at the end of each slot in order to minimize interference with external nodes. The choice of which channel should be used for communication is dictated by a common *communication* schedule. This schedule allocates each channel a number of slots proportional to its quality and tries to keep channel usages from the same channel as far apart as possible [EG18a].

Channel qualities and with it the set of *usable* channels, i.e. channels with sufficient quality for proper communication, may change in space and time. This requires dynamic adjustments to the communication schedule in order to adapt to these new conditions and a consensus mechanism through which all nodes use the same communication schedule. Furthermore, to keep the channel usage in different nodes not only in the same order (schedule consistent) but also aligned in time, we need network-wide synchronization.

First, we will introduce a master-based three-dimensional stabilization protocol in which a master node is responsible for time-giving and for the computation of communication schedules. In this motivating scenario we have a network where all nodes communicate on a common channel and where the master node is an essential component to keep all nodes in the network both synchronized and schedule-consistent, i.e. all nodes eventually possess the same communication schedule. The stabilization protocol can handle multiple perturbations, such as clock skew (dimension time), significant changes in channel quality (dimension channel) and topology alterations (dimension space), hence three-dimensional, with little deterioration in network performance.

Since the master node should produce a channel hopping sequence that optimizes the overall operation of the network it should aggregate the views of all nodes within it. With this purpose, all non-master nodes report the channel qualities measured locally through channel quality reports.

Later, we will relax the constraint that a master node is *always* present and will sketch a leader election algorithm in which a new master node is elected.

Even though we originally proposed this stabilization protocol in [JEG18] for cognitive radio networks in the presence of primary users, i.e. license holders of a given frequency

band, in this chapter we will refine and expand the proposed protocol without making any distinction between primary and secondary users. Note that primary users are still possible in 802.11 networks when using certain channels in the 5 GHz band, but they will be treated here purely as foreign traffic. 802.11a nodes are secondary users in a portion of the 5 GHz band, in which primary channel usages are satellite communications and radars. To avoid interfering with ongoing radar transmissions, 802.11a AP nodes use both transmission power control (limit their radiated energy) and Dynamic Frequency Selection (DFS), i.e. hop to another channel upon detecting radar transmissions or stay silent for a certain time. DFS uses physical layer characteristics of radar transmissions, such as pulse width and the number of pulses per burst, to detect them. However, 802.11 DFS implementations are far from perfect and many deployed nodes at 5 GHz implement DFS in a manner non-compliant with regulatory requirements [SCT+16]. Indeed, detecting certain types of radar transmissions is very challenging. For instance, pulses in weather transmissions are very short (0.5 to 2 μs), radar beam direction varies with time, typically at every 1 to 10 min, and waveforms used for matching radar transmissions during DFS certification tests while comprehensive do not cover all possible types of weather radar communication [SCT+16]. Moreover, in contrast to our protocol, DFS hopping sequences are static and built into the implementation of the protocol. Furthermore, it is not only in the 5 GHz band that primary users can occur: with the advent of the 6 GHz band for 802.11 networks, approved for use in Germany since July 2021 [Bun21], coexistence between secondary and primary users (e.g. 6 GHz microwave links) remains a very relevant topic.

A natural extension of the techniques developed in Chapter 2 in the presence of primary users would be to give the traffic resulting from primary users more weight than the traffic generated by secondary foreign nodes when computing channel qualities. This way channels with equivalent foreign spectrum occupation could have different channel qualities based on primary user activity, i.e. channels where primary nodes are more active would have lower channel quality. This however presupposes a mechanism for 802.11 devices to distinguish primary users, either decoding a primary user's frames or through signal detection (as in the case of 5 GHz). In addition, supporting multiple primary user's protocols requires special re-programmable radio such as a software defined radio (SDR). Even though SDR is a promising technology that can bring more flexibility to the table, this flexibility comes at a cost: SDR tunable frequency ranges can be quite limited (most boards do not support 5 GHz off-the-shelf), SDR boards are considerably more expensive than commodity 802.11 hardware (more robust boards with larger

frequency ranges can cost upwards of 500 dollars as of 2021) and most mobile boards are quite young, still under active development and lack long proven records regarding stability, accuracy and performance.

This chapter is structured as follows. In Section 4.1, we outline the foundations for our stabilization protocol. In Section 4.2, we discuss how to achieve fast re-stabilization having already reached an initial stabilization. In Section 4.3, we sketch out a leader election mechanism to cope with master node failures and in Section 4.4, we discuss the steps needed for achieving initial stabilization. After that, Section 4.5 presents simulation results regarding the performance of our stabilization protocol and Section 4.6 sketches out a further application of the techniques described in this chapter and in Chapter 3 through conflict-minimal channel orderings for communication schedules. Finally, a summary of the chapter is delivered by Section 4.7.

4.1 Foundations

In this section, we briefly review and extend our used system model, formalize and discuss the computation and the quality assessment of communication schedules and review its relation to sensing schedules.

4.1.1 Channels

As mentioned in previous chapters, we have a non-empty finite set of channels C. This set of all available channels is known beforehand to all nodes and is also fixed at runtime. In fact, the regulatory constraints of which frequencies are available to each node in a specific country is preconfigured before deployment in the form of a Central Regulatory Domain Agent (CRDA) database.

Each channel has a center frequency and the nodes use all channels with the same characteristic bandwidth. A typical channel bandwidth value for 802.11 is 20 Mhz, but smaller channels can be bonded into larger channels with 40 MHz or more. Channels might be chosen from different frequency bands available to the nodes.

4.1.2 Graph model and topology

Due to the nature of the wireless medium, the network has lossy and possibly asymmetric links. Moreover, since nodes might temporarily use different channels at the same time and different environmental conditions will inevitably produce distinct channel qualities

for different channels in different regions of the network, we will have distinct network topologies for different channels. We can hence model our network as the union of all directed graphs for every channel $c \in C$, $G = \{G_c = (V, E_c)\}$, where an edge $(v_1, v_2) \in E_c$ indicates that node v_1 can communicate with node v_2 on channel c.

In addition, there is a distinct node $v_{master} \in V$ that acts as the master node in the network and every node v stores the minimal known number of hops $v.hops$ from itself to v_{master}. This hop count is added by each node to all its outgoing management messages. With this, the neighbors of v can update their own hop counts as well as store $v.hops$. The importance of this step will be explained later on.

Furthermore, nodes have no pre-configured knowledge of other nodes or of the topology of the network at deployment.

4.1.3 Time-slotted channel hopping

Much as in Time Division Multiple Access (TDMA) protocols, time in our network is divided into time slots of fixed duration d_{slot}. Every node in the network adopts the same slot duration d_{slot} and the same number n_{slot} of time slots in a hopping sequence.

Some protocols in the literature also make use of a time-slotted operation coupled with schedule-based channel hopping, the most promiment example being TSCH [WPG15], one of the MAC layer protocols defined by IEEE 802.15.4e [tsc12]. TSCH was designed for low-power and lossy networks and has several similarities with our stabilization protocol: time is divided into slots and a fixed number of slots compose a slotframe, nodes are time-synchronized to a coordinator node, and each node follows a communication schedule. Nonetheless, different from our protocol, the schedule tells TSCH nodes not only which channel to use for communication in which slot but also which neighbor it should communicate with. A further distinction from our protocol is that TSCH nodes can either transmit or receive in a slot (links are half duplex) and can only receive from a single node during a slot. Moreover, schedules tell nodes when to sleep, receive or transmit data, whereas in our approach nodes only need to stay passive during the initial stabilization phase. Another difference is that our schedule (unless re-computed) assigns the same channel to the same slot for each iteration of the schedule, while in TSCH the same slot in a schedule is assigned a different channel for every iteration of the slotframe [DANLW15].

Furthermore, the TSCH protocol defines more of a framework with different mechanisms needed to operate with a communication schedule, but it does not define how to compute it or if needed how to distribute this schedule. Hence, different scheduling

approaches can be found in the literature, such as Orchestra [DANLW15], a distributed approach designed for RPL [WTB+12] networks, and TASA [SZQ+19], a centralized scheduling algorithm. While Orchestra, for example, lets nodes compute schedules autonomously, TASA relies on an omniscient master node, which not only has complete knowledge of the topology of the network, but also knows the traffic load generated by each node.

4.1.4 Channel quality metric

Every channel $c \in C$ has a dynamic channel quality $q_c \in [0,1]$ that is derived through passive channel sensing. As described in Chap. 2, every node computes the volatility-aware quality of each channel at the end of each time slot and aggregates it locally. The computed qualities of node v are listed in vector $v.\vec{q} \in [0,1]^{|C|}$. The main used channel quality metric, q_{cbt}, is based on energy detection and correlates strongly with achievable throughput on any given channel. Our implementation on 802.11 commodity hardware measures the spectrum occupation needed to compute q_{cbt} with help of 802.11 CCA capabilities.

Furthermore, the main channel quality metric may be enhanced by combining it with frame-based metrics to incorporate additional information such as the signal-to-noise ratio. In general, we assume that the channel quality also strongly correlates with the probability of frame delivery.

4.1.5 Communication schedules

The master node is responsible for synthesizing the global channel hopping sequence, i.e. communication schedule, to be adopted by the whole network. Whereas in Chap. 2 we formalized the computation of channel qualities based on passive observation of the medium, in this section we will briefly describe how communication schedules are synthesized. First we start with some definitions.

Definition 4.1.1. (Schedule). A schedule s maps every time slot in the interval $m \in \{1,\ldots,n_{slot}\}$ to a channel $c \in C$.

$$\begin{aligned} s\colon \{1,\ldots,n_{slot}\} &\to C \\ i \in \{1,\ldots,n_{slot}\} &\mapsto s(i) \in C \end{aligned} \tag{4.1}$$

Hence, a schedule is a channel hopping sequence, i.e. a list of channels a node hops to in a given order repeatedly. As the name indicates, a communication schedule is therefore a schedule that defines which channels should be used for transmissions and in which slots.

4.1.6 Schedule computation

The algorithm we apply to compute communication schedules follows the technique developed by Engel [EG18a]. This method solves the apportionment problem by trying to allot each channel a fair share of the schedule w.r.t its channel quality while keeping channel reuses within the schedule as far apart as possible. The main insight of Engel's method is ensuring that channels with better quality are used more often while avoiding grouping channel reuses such that significant changes in the quality of one channel will not have such a great impact on the overall performance of the network. This helps keep the network more robust in special against channel failures since the slots before a channel reuse function as a time buffer during which a new schedule can be synthesized and disseminated into the network.

Definition 4.1.2. (Utilization). We define a vector $\vec{u}$ over all channels such that the *utilization* $u_c \in \mathbb{N}$ of a channel c denotes the number of slots in which c appears in the communication schedule.

This way, the utilization of a channel determines how often this channel is used every n_{slot} slots, which means that the sum of all utilizations in $\vec{u}$ must add up to n_{slot}. As we will later describe, $\vec{u}$ is derived from $\vec{q}$, the vector containing the channel quality of each channel.

Definition 4.1.3. (Relative quality). We define the relative quality τ_c of a channel c as the relation between its measured quality q_c and the sum of the qualities of all channels:

$$\tau_c := \frac{q_c}{\sum\limits_{c \in C} q_c} \tag{4.2}$$

The relative quality can be interpreted as a normalized quality such that $\sum_{c \in C} \tau_c = 1$.

Example 4.1.1. Let $C = \{c_1, c_2, c_3\}$ be a set of channels with qualities $\vec{q} = (1, 0.4, 0.6)$. We can the compute $\vec{\tau} = \frac{\vec{q}}{\sum\limits_{c \in C} q_c} = (0.5, 0.2, 0.3)$.

Furthermore, we need to formalize the concept of a fair share.

Definition 4.1.4. (Fair share). Given n_{slot} the number of slots in a schedule, C the set of all channels and $\vec{\tau}$ the relative qualities, we define the fair share as the following utilization:

$$\vec{u^\star} := n_{slot} \cdot \vec{\tau} \tag{4.3}$$

Example 4.1.2. Let $C = c_1, c_2, c_3$, $n_{slot} = 8$ be a set of channels with relative qualities $\vec{\tau} = (0.5, 0.2, 0.3)$ the fair share is then $\vec{u^\star} := n_{slot} \cdot \vec{\tau} = (4, 1.6, 2.4)$.

Having defined the concept of relative qualities and fair share, we can finally formally define the apportionment problem as follows.

Definition 4.1.5. (Apportionment Problem). Given the set of all channels C, a set of channel qualities $\vec{q}$ and the number of slots in a schedule n_{slot}, find a utilization $\vec{u}$ that approximates the fair share $\vec{u^\star}$ such that $\forall u_c \in \vec{u}.\ u_c \in \mathbb{N}_{>0}$ and $\sum_{u_c \in \vec{u}} u_c = n_{slot}$.

As illustrated in Example 4.1.2, while utilizations are natural numbers, fair share values are not always whole numbers. Since we are dealing with real numbers, in the general case, we can only approximate the fair share of each channel. The apportionment problem is a well studied problem and finds its original motivation in the distribution of seats in electoral systems characterized by proportional representation such as the European Parliament. In such a system, the support of $\tau\%$ of the electorate for a given party leads to an allotment of approximately $\tau\%$ of the available seats to the party. In our application of the apportionment problem, this allotment is kept until a significant change in channel qualities and consequently in utilizations takes place. Hence, slot assignment repeats every n_{slot} slots, until fair shares and slot assignments are re-computed, a process akin to holding new elections. Nonetheless, while in politics old election results are immediately discarded, in our scenario, we change assigments progressively by applying an *incremental convergence* algorithm, which will be described later in this chapter.

Engel [EG18a] surveyed different apportionment methods and identified the Hamilton method as a good choice for our requirements. The Hamilton method (also known as the Vinton method) [BY75] is a quota method and works by initially giving each channel its lower quota $\lfloor u_c^\star \rfloor$, then ordering the channels based on the fractional remainders of their fair shares $r_c = u_c^\star - \lfloor u_c^\star \rfloor$. The remaining $n_{slot} - \sum_{c \in C} \lfloor u_c^\star \rfloor$ slots are then distributed such that each channel receives one additional slot following a priority ordering where channels with higher remainders have a higher priority. Since we might have ties (channels with the same remainder) we can generate distinct equal-valued solutions following this method.

Having derived an optimal utilization that approximates the fair share of each channel, we already know how often every channel should appear in our communication schedule, but we still do not have an order of usage for these channels. Defining this order on top of a given utilization is the last step to synthesizing a communication schedule. As already mentioned, this order should be chosen such that we try to maximize the distances between channel reuses.

Definition 4.1.6. (Schedule compliance). A schedule $\vec{s}$ is compliant with a utilization $\vec{u}$ if the number of distinct channels that appear in $\vec{s}$ equals $|\vec{u}|$ and every channel occurs exactly u_c times in $\vec{s}$.

Example 4.1.3. Given utilization $\vec{u} = (3, 2, 1)$ and schedule $\vec{s} = (c_1, c_2, c_3, c_1, c_2, c_1)$, then $\vec{s}$ is compliant with $\vec{u}$, since $\vec{s}$ has three distinct channels ($|\vec{u}| = 3$), channel c_1 occurs 3 times ($u_1 = 3$), c_2 two times ($u_2 = 2$) and c_3 occurs one time ($u_3 = 1$) in $\vec{s}$ as mandated by $\vec{u}$.

Definition 4.1.7. (Reuse optimization goal). Given a set of channels C, $\vec{s}$ a schedule compliant with a given utilization $\vec{u}$ and the set $S(c)$ of all slots in which channel $c \in C$ is used in $\vec{s}$, we have following optimization goal: for any slots $i, j \in S(c)$ we want to maximize the distance $|i - j|$.

Note that there can be no general solution that always constructs a compliant schedule that also maximizes the channel reuse for all channels, since permuting a channel within a schedule affects other channels. Moreover, it is also not possible to always avoid adjacent channel reuse. In fact, Engel demonstrated that at most one channel c will have to suffer adjacent reuse and that this channel has a utilization $u_c > 0.5 \cdot n_{slot}$. Therefore, in the absence of a general solution, Engel proposed a heuristic to construct sub-optimal high-quality communication schedules, where ideally the channel reuses are distributed equally among all channels. The theoretical optimum $\delta_c^\star$ for the number of slots between channel reuses of a channel $c \in C$ is:

$$\delta_c^\star = \frac{n_{slot}}{u_c} \tag{4.4}$$

4.1.7 Schedule quality metric

In order to qualify a compliant schedule w.r.t. the channel reuses we use the schedule quality metric $\Omega(\vec{s})$ defined by [EG18b]:

$$\Omega(\vec{s}) = \begin{cases} 1 & \text{if } \Psi_{min} = \Psi_{max} \\ 1 - \frac{\Psi(\vec{s}) - \Psi_{min}}{\Psi_{max} - \Psi_{min}} & \text{otherwise} \end{cases} \tag{4.5}$$

where given $\delta(c, i)$ the number of slots between the i-th and (i+1)-th use of channel $c \in C$, we have

$$\Psi(\vec{s}) = \sum_{c \in C} \sum_{i=1}^{u_c} \frac{(\delta(c, i) - \delta_c^\star)^2}{\delta_c^\star} \tag{4.6}$$

In order to compute $\Omega(\vec{s})$, we need both Ψ_{min} and Ψ_{max}, respectively the squared error sum of the best and worst possible schedule w.r.t. channel reuses. Ψ_{max} is easy to obtain by computing the squared error sum of a schedule where all channels are used in consecutive slots, i.e. $\forall c \in C. \forall i \in [1, u_c - 1]$. $\delta(c, i) = 1$ and $\delta(c, u_c) = n_{slot} - (u_c - 1)$. This way, we have:

$$\Psi_{max} = \frac{1}{n_{slot}} \sum_{c \in C} ((u_c - 1) * (n_{slot} - u_c)^2) \tag{4.7}$$

Given a utilization $\vec{u}$, to compute Ψ_{min} we would have to generate all schedules compliant with $\vec{u}$ and determine the minimum squared error sum. However, the number of possible schedules grows extremely fast with n_{slot} and $|C|$. In fact, the number of all possible compliant schedules is $|S(\vec{u}, n_{slot}, C)|$:

$$|S(\vec{u}, n_{slot}, C)| = \frac{n_{slot}!}{\prod_{c \in C} u_c!} \tag{4.8}$$

So, instead of taking the true Ψ_{min}, we can calculate its lower bound $\Psi_{min}^\star$ as:

$$\Psi_{min}^\star = \sum_{c \in C} E_{min}(c) \tag{4.9}$$

where $E_{min}(c)$ is the minimum squared error sum for channel c and can be computed as (see [EG18b]):

$$E_{min}(c) = \frac{1}{\delta_c^\star} \cdot (n_{slot} \bmod u_c) \cdot (1 - \delta_c^\star + \lfloor \delta_c^\star \rfloor) \tag{4.10}$$

By using this lower bound instead of the true Ψ_{min} we have to keep in mind that the metric becomes a bit more pessimistic since schedules where $\Psi_{min}^\star$ cannot be reached will have a quality below 1 even if the schedule is the best possible one.

4.1.8 Heuristic Computation

Finally, given a utilization $\vec{u}$, we use the heuristic H_1, introduced by [EG18b] to compute a schedule $\vec{s}$ compliant with $\vec{u}$ that optimizes $\Omega(\vec{s})$. For this, we first define the local cost of using channel $c \in C$ at some slot i in $\vec{s}$:

$$L(\vec{s}, c, i) = \frac{((i - last(\vec{s}, c, i) - \delta_c^\star)^2}{\delta_c^\star} \tag{4.11}$$

where $last(\vec{s}, c, i)$ is the last slot in $\vec{s}$ before slot i at which channel c occurs.

Given a utilization $\vec{u}$, for every slot $1 \leq i \leq n_{slot}$, H_1 decides which channel $c \in C$ to place into s_i as follows:

1. Compute the set $C_{cand} \subseteq C$ of candidate channels, i.e. those channels which have not yet achieved their utilization u_c. All other channels are ignored. This guarantees the compliance of $\vec{s}$ with the desired utilization $\vec{u}$.

2. Compute the local cost $last(\vec{s}, c, i)$ of all channels $c \in C_{cand}$. If the channel was never used in the schedule its local cost is 0.

3. Compute the set C_{inc} of channels whose local error is increasing and pick the channel $c_{max} \in C_{inc}$ with the maximum local error for s_i. We know due to quadratic nature of the local cost function that for any given channel its local cost will decrease up to some slot and will start increasing from then on.

4. If $C_{up} = \{\}$, pick the channel $c_{min} \in C_{cand}$ that has the minimum $L(\vec{s}, c, i)$.

4.1.9 Channel sensing schedules

As introduced in Chap. 3, a channel sensing schedule defines the channel sensing order, that is in which slot should the channel quality of each channel be measured. This channel hopping sequence is computed by each node locally and is indeed derived from communication schedules with help of a stochastic local search-based heuristic. This heuristic attempts to minimize the channel overlap between both schedules, which minimizes the effect of internal transmissions in the resulting channel quality metric. Having minimal overlap between schedules allows nodes to derive channel quality primarily based on foreign traffic. In addition, in order to minimize bias in the channel sensing process, the heuristic constructs balanced sensing schedules, i.e. each channel is assigned the same number of slots within a sensing schedule. Moreover, the remaining channel

overlap between the sensing and the communication schedule is distributed among the channels steering towards overlap-fairness.

4.2 Fast restabilization

In this section, we discuss the steps taken by our protocol to stabilize the network operation w.r.t. time synchronization and schedule-consistency. The discussion in this section assumes the presence of a master node v_{master} and that the network has already performed an initial stabilization, i.e. all nodes possess the same communication schedule s_{com} and are synchronized. As already mentioned, these constraints (master node and initial stabilization) will be relaxed and we will discuss in later sections the adjustments needed for dealing with the relaxed constraints.

There are three main perturbation sources that affect the stability of the network

- clock skew, making it harder for nodes to keep the same time structure.
- topology changes due to node movement or node failure, changing the current communication topology
- significant changes in channel quality, which requires the re-computation of the communication schedule.

The main mechanisms to correct these perturbations are:

- Tick synchronization to deal with clock skew perturbations.
- Periodic exchange of channel quality reports to correct for channel quality changes.
- Periodic exchange of schedule reports to disseminate the current communication schedule into the network.

Our protocol is in addition robust against topology changes and provides a fast restabilization. One of the reasons for this is that the degradation of quality on some channels has a limited impact in the stabilization protocol, since tick frames, schedule reports and channel quality reports can be further exchanged on channels that still display good quality, without having to repeat the initial stabilization process.

4.2.1 Synchronization

In synchronous digital circuits, synchronism is introduced by a *clock* signal with an associated frequency of oscillation. For binary signals this oscillation takes place between a low and a high state. Clocks count the number of oscillations of an oscillator, usually a piezoelectric resonator, such as a quartz crystal, that vibrates with a precise frequency when an electric charge is passed through it. By counting these oscillations or ticks, a clock can be used to measure the passage of time. In a perfect world with perfect clocks that always gave the same time at the same rate a synchronized network would stay synchronized forever. However, in the real world, wireless nodes rely on crystal oscillators that are fallible. A basic crystal oscillator can achieve frequency accuracy in the order of 10^{-4} and 10^{-5}, which means that they yield errors in the order of 10 to 100 microseconds per second [TA20].

Even though these oscillators are very stable, the frequency of the generated signal is still sensitive to temperature and supply voltage fluctuations. In addition, other factors such as impurities in the crystal and its geometry also influence the frequency stability. The resulting frequency variations in the network, also called clock drift, lead to recurring clock offsets among nodes, i.e. differences in the given times.

To account for this clock skew and minimize clock offset between nodes we need a mechanism to resynchronize them properly and frequently.

Most synchronization algorithms in the literature are master-based. Examples thereof are TPSN [GKS03], the Gossiping Time Protocol GTP [IvSV06], Recursive Time Synchronization Protocol (RTSP) [AS12] and the Flooding Time Synchronization Protocol (FTSP) [MKSL04].

Recent research and standardization attempts for future 802.11 standards have also proposed using IEEE 1588, the Precision Time Protocol (PTP) [CSZ+15] for achieving time-sensitive capabilities in 802.11 networks. PTP is a time synchronization protocol originally developed for wired networks, which is able to achieve sub-microsecond accuracy.The PTP master nodes synchronizes with its slave nodes using sync messages.

In fact, almost all synchronization methods in wireless networks rely on message passing between network nodes. An exception here are some protocols that make use of the Global Positioning System (GPS). Especially solutions for wireless networks in outdoor spaces have relied on GPS for synchronization, since GPS time has an offset in the nanosecond range w.r.t. the Universal Time Coordinated (UTC). However, this requires additional dedicated hardware (GPS modules) that are not only expensive, but

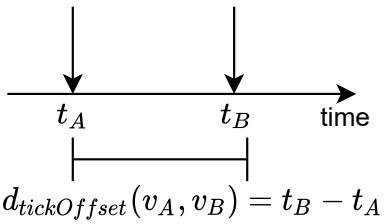


Figure 4.1: Tick offset between the tick perceived by node v_A and the tick perceived by node v_B

also energy-hungry. Moreover, since GPS depends on proper satellite coverage, using it indoors is not always ideal.

Black Burst Synchronization (BBS)

Another master-based synchronization protocol that seems to be a good candidate for our needs is Black Burst Synchronization (BBS) [GK11], a protocol devised for deterministic tick or time synchronization in wireless ad hoc networks. The main insight behind BBS is the use of *black bursts*, fixed-length energy bursts starting at pre-determined points in time, to encode synchronization messages such that collisions are non-destructive. Moreover, BBS provides low convergence delay and in the absence of foreign nodes bounded clock offsets.

Definition 4.2.1. (Clock offset). Given two nodes v_A and v_B with respectively clocks c_A and c_B and local times $c_A(t)$ and $c_B(t)$, then the clock offset of c_A relative to c_B at time t is

$$d_{offset}(c_A, c_B, t) = c_A(t) - c_B(t) \tag{4.12}$$

Synchronization comes in two flavors: tick and time synchronization. Whereas time synchronization intends to keep both reference points in time and a clock value synchronized, tick synchronization has no need for the clock value and only focuses on minimizing the tick offset (see Fig. 4.1) network-wide. One of the oldest tick synchronization protocols is the Reference-Broadcast Synchronization (RBS) protocol [EGE03].

For our purposes, network-wide tick synchronization is sufficient, since we only need to keep the time slots in all nodes aligned, and therefore do not need to exchange clock values.

In order to achieve tick synchronization with BBS, nodes broadcast special synchronization messages called tick frames. The operation of a BBS-based tick synchronization can be outlined as follows:

1. The resynchronization is started at the beginning of every slot by the master node, which sends a tick frame with a round number 1 to its downstream neighbors.

2. One-hop neighbors of the master node receive the tick frame and can then resynchronize by recording the reception time as their new local tick. They then broadcast a synchronization message with round number 2 at the beginning of the next slot.

3. Nodes on the next hops of the network repeat this process (always increasing the number of the round) until the maximum known diameter of the network is reached. Based on the time of reception t_{rx}, the duration of a round d_{round} and the round number i, each node can determine its resynchronized local tick

$$t' = t_{rx} - (i-1) \cdot d_{round}$$

By using BBS, tick frames from the same round are sent almost simultaneously without creating destructive collisions. This is achieved by encoding the bits of every synchronization message with black bursts. Overlapping black bursts resolve to a bitwise OR operation on the medium.

To implement a black burst we need to transmit energy on the medium with a fixed period. In fact, the information contained in the transmitted energy is irrelevant for the BBS encoding: a bit has value 1 if energy is transmitted on the medium and a bit has value 0 if no transmission takes place. This can be achieved with 802.11 commodity hardware by sending a 802.11 frame with no payload. A good choice in this case are control frames (see Fig.4.2), which in contrast to management and data frames have no frame body (no data payload) and therefore have minimal frame length in number of bytes.

Even though the BBS-inspired tick frame dissemination logic can be applied to 802.11, the presence of foreign traffic and false positives might nullify the benefits of using the black burst enconding itself. For instance, foreign transmissions that cannot be distinguished from logical 1s while internal nodes are sending logic 0s will corrupt these internal black bursts. One way to try to counteract this effect is by identifying the used control frames by setting the 4-bit `subtype` field to one of its reserved (unused) values such as 0000. This would help then to distinguish possible black burst logical bits from external transmissions and help avoid false positives. Nonetheless, since we have no

control over foreign nodes, we cannot guarantee, for example, that a similar encoding for control frames will not be used by such nodes.

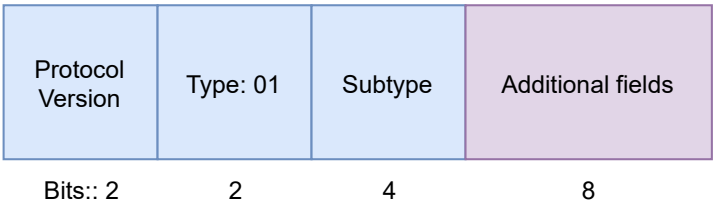


Figure 4.2: Relevant frame control fields for implementing BBS with 802.11. The type field is always 1 for control frames and the subtype varies depending on application, such as 1011 for RTS and 1101 for ACK frames.

Our approach

A probably better approach for our scenario is to discard the BBS encoding altogether and use a single regular frame per node instead of sending multiple short 802.11 frames. This makes a tradeoff, of course, of avoiding false positives, while having to deal with more tick frame collisions and losses, and less synchronization accuracy. However, not only can our scenario handle the loss in synchronization accuracy, but it is possible to minimize tick frame collisions by having nodes choose a random timepoint within a time slot for broadcasting the current tick frame.

Much as in BBS, in our approach to tick synchronization, tick frames are broadcast from the master frame and every receiving node re-transmits the tick frame, containing:

- the number of the current slot
- the number of milliseconds since the start of the current slot at the time of the tick frame transmission

These two pieces of information are all a node needs to re-sync its current time slot upon reception of a tick frame. Here we can disregard the time needed for the signal to travel from one node to the other, since it lies in a microsecond range and we are dealing with time slots with duration in a millisecond range.

The master node always broadcasts tick frames at the beginning of each slot where the channel with the highest utilization occurs. Transmitting on the channel with the most slots (which was the best channel at the time of computation of the communication schedule) increases the probability of successful delivery of each tick frame due to

the good quality of the channel and provides the highest frequency of tick frame dissemination by the master node when always using the same channel to broadcast tick frames.

Nodes downstream from the master node try to re-transmit the tick frame in the same time slot, and if it is not possible on the next available slot until the synchronization process is finished. To allow multiple nodes to broadcast in the same slot while trying to minimize tick frame collisions, we divide a time slot in 1 ms synchronization windows. Every node picks a random time point within the next available 1 ms window to broadcast the current tick frame. The nodes in the next hop repeat this procedure until a network-wide synchronization is reached.

This simple scheme should provide with high probability a convergence delay of one time slot for any network with a diameter of less than or equal to $\frac{d_{slot}}{1ms}$ hops (as long as we do not have hundreds of nodes per hop), where d_{slot} is the duration o a single time slot. For a maximum number n_{hops} of hops in the network, we should reach a network-wide synchronization with high probability in at most $\frac{n_{hops} \cdot 1ms}{d_{slot}}$. Since in our scenario we adopt $d_{slot} \geq 10$ ms, this convergence delay should be no worse than $\frac{n_{hops}}{10}$ time slots.

Moreover, by knowing its distance in hops to the master node, each node can calculate the worst case delay for it to receive a tick frame, allowing it to detect missing tick frames, which is a important signal in detecting a potential master node failure.

It is worthy noticing that it is sufficient for nodes in our network to stay *loosely* synchronized within one-hop neighborhoods, since only nodes within communication range can talk to each other. In addition, by repeating the synchronization process, we can keep the clock offset in the network bounded in the order of $\mathcal{O}(d_{slot})$, where d_{slot} is the duration of a single time slot.

4.2.2 Channel quality reports

In our master-based approach, the communication schedule is synthesized by the master node based on the qualities of all channels. However, in order to optimize the overall throughput and minimize the overall interference on the used channels network-wide, it is insufficient to only consider the channel qualities measured *locally* by the master node v_{master}. Depending on the placement of internal nodes as well as on the placement and channel usage of foreign nodes, locally measured qualities might paint a complete different picture of the spectrum. With this in mind, nodes periodically broadcast special messages, i.e. channel quality reports, responsible for carrying locally measured channel qualities up to the master node. It is then the responsibility of v_{master} to properly

aggregate these channel qualities and compute a *global* communication schedule to be distributed in the network. Channel quality reports are broadcast asynchronously in an independent manner. Furthermore, *upstream* neighbors of reporting nodes also aggregate the received channel qualities into their own channel quality reports.

This approach based on channel quality reports follows the parallel data fusion model described in [Var12]. This cooperative model is based on spatially distributed nodes that observe a given phenomenon (in our case the channel quality) and that report their observations to a central coordinator (our master node), which then combines them and carries a global decision (compute a communication schedule and disseminate it). Different schemes in the literature have also used the parallel data fusion model or variants thereof for cooperative spectrum sensing [ALB11], most of them in the context of cognitive radio networks in the presence of primary users.

Broadcasting reports

Every node v reports the *aggregation* of its own channel quality report $\vec{q}_v$ with the channel quality reports of reporting *downstream* neighbors.

Definition 4.2.2. (Downstream neighbor). Given a node v with hop count $v.hops$, i.e. the number of hops to the master node, and a node u with hop count $u.hops$ such that $v.hops < u.hops$ and u is in communication range of v then u is a *downstream neighbor* of v.

The added constraint that nodes only aggregate reports from downstream neighbors tries to avoid the generation of cycles and attempts to construct a directed convergecast tree rooted at the master node, the final destination of the reports (see Fig. 4.3).

Nodes report at maximum $n_{rep} \in \mathbb{N}$ times every n_{slot} slots. Having $n_{rep} > 1$ attempts to compensate for lost reports, e.g. due to high interference conditions, and tries to reach more upstream nodes faster such that the information contained in these reports is carried faster to the master node.

At every slot, nodes decide whether they should broadcast a channel quality report or not. This decision is based on the local quality of the current channel and on the channel qualities reported by upstream neighbors. By picking channels with higher channel qualities (both on the sending and the receiving side), nodes increase the probability of a successful report delivery.

Nodes expect channels to comply with two requirements in order for them to carry quality reports:

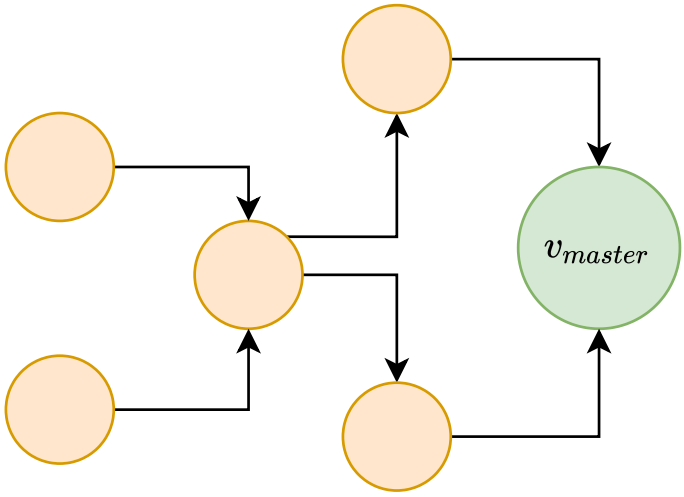


Figure 4.3: Illustration of reports being brought to the master node. Each incoming edge indicates the delivered report is aggregated at the receiving node.

1. The channel is the best local channel, i.e. the channel with the highest channel quality adapted for volatility-awareness and locally aggregated (for now ignoring reported qualities from downstream neighbors).

2. If the current channel is not the best channel, at least n_{up} upstream neighbors of the node must have used this channel for their quality reports in the past n_{slot} slots.

Note that for this scheme to work, nodes have to keep track of which channels are used by its upstream neighbors for channel quality reports.

4.2.3 Aggregation of quality reports

The next issue to be dealt with is the aggregation function. How should each node combine the incoming reports with its own measured channel qualities? One possible approach would be to simply average the incoming quality vectors. However, this simple averaging procedure can deliver quite undesired results by e.g. delivering a communication schedule that is non-optimal for a large portion of the network. This can happen because an arithmetic mean will give all downstream nodes the same impact on each aggregation performed at every upstream node along the way to the master node. One way to deal with this is to make use of a weighted average of the incoming reports, with weights derived from the current network state and topology.

In [VJP08], for instance, secondary users in a cognitive radio network perform cooperative spectrum sensing and report spectrum energy measurements that are combined by a master node with a weighted arithmetic mean where weights are derived from the local mean SNR observed at each node. Simulations showed that this weighted combination yielded an improved primary user detection performance with a lower false alarm rate than when combining all energy measurements with the same weight.

In our approach, we take topology into consideration by deriving the aggregation weights out of each reporting node's *downstream* in-degree. In fact, we want the higher-weight nodes to hold more influence in the final channel quality aggregation.

Definition 4.2.3. (Downstream in-degree). We define the node *downstream in-degree* $v.w_{in}(c)$ of node v on channel $c \in C$ as the number of downstream neighbors of v on c.

Note that the downstream in-degree is always smaller or equals the total node in-degree, since we can also have incoming transmissions from upstream neighbors on any given channel.

We estimate each node's downstream in-degree by calculating the number of unique MAC addresses seen in incoming messages from downstream neighbors in the past d_{seen} seconds. This time constraint helps avoid the set of downstream neighbors becoming stale, e.g. a node is offline, but is still considered a neighbor. Therefore, each node repeatedly computes its downstream in-degree and sends it along with its channel quality report to its upstream neighbors.

Weighing the aggregation of the collected reports based on the network topology prevents nodes with fewer connections from highly influencing the final quality vector at the master node, which will be used for synthesizing the communication schedule.

In addition to the network topology, we look at a metric based on local measures of the current channel quality and the current communication schedule to help us compute the aggregation weights: the expected achievable throughput.

Definition 4.2.4. (Expected achievable throughput). Given a communication schedule s_{com}, and u_c the utilization of channel $c \in C$ with quality q_c, we define the expected achievable throughput as:

$$U(s_{com}) = \sum_{c \in C} u_c * q_c \tag{4.13}$$

The maximum expected achievable throughput for a given communication schedule s_{com} is then

$$U_{max}(s_{com}) = \sum_{c \in C} u_c = n_{slot} \tag{4.14}$$

If a network achieves $U_{max}(s_{com})$, this means it has the full capacity of all channels at its disposal and could under ideal conditions achieve maximum throughput.

Definition 4.2.5. (Normalized expected achievable throughput). We then define the normalized version of the expected achievable throughput $\tau_{EAT} \in [0, 1]$ as

$$\tau_{EAT}(s_{com}) = \frac{U(s_{com})}{U_{max}(s_{com})} = \frac{\sum\limits_{c \in C} u_c * q_c}{\sum\limits_{c \in C} u_c} = \frac{1}{n_{slot}} \cdot \sum_{c \in C} u_c * q_c \tag{4.15}$$

This means that decreasing values of $\tau_{EAT}(s_{com})$ for a given node signal that this node is profiting less from the current hopping sequence either due to significant changes in relative qualities among channels, i.e. the fair share w.r.t. slot distribution has changed, or due to an overall worsening in channel conditions, i.e. all channels are suffering more foreign traffic and have worse qualities for it, without displaying a significant change in fair share.

Definition 4.2.6. (Aggregation weight). We then define the aggregation weight $\vec{w_r}$ reported by node v to be used *upstream* in the aggregation of the quality reports coming from nodes with similar hop counts. $\vec{w_r}$ should be proportional to the node downstream in-degree $\vec{w_{in}}$ and proportional to (1 - τ_{EAT}) such that:

$$\vec{w_r} = (1 - \tau_{EAT}) \cdot (1 + \vec{w_{in}}) \tag{4.16}$$

Note that we use $(1+v.\vec{w_{in}})$ instead of $v.\vec{w_{in}}$ because we have to account for leaf nodes with no downstream neighbors.

Our aggregation scheme works then as follows:

1. Every node v receives channel quality reports from downstream neighbors and stores these reports for later use in a set of received but not aggregrated reports $v.R$.

2. Before broadcasting a report, node v aggregates the $|R|$ received reported qualities using the reported weights:

$$q_R[c] = \frac{1}{|R|} \sum_{r \in R}^{|R|} w_r[c] \cdot q_r[c] \tag{4.17}$$

3. The locally aggregated quality $\vec{q_{aggr}}$ of node v is then combined with $\vec{q_R}$:

$$\vec{q_R^\star} = \frac{1}{2} \cdot (\vec{q_{aggr}} + \vec{q_R}) \tag{4.18}$$

4. $\vec{q_R^\star}$ is then broadcast in a channel quality report together with the weight $\vec{w_r}$ of node v.

As described above, our aggregation policy tends to prioritize over time those regions of the network that either have the higher network densities, observe the worst channel quality conditions or are *losing* the most with the current schedule w.r.t. achievable throughput due to a high discrepancy of the local fair share and the global one embedded in s_{com}.

The aggregation weights dynamically adapt to new conditions in the network e.g. τ_{EAT} changes due to a new communication schedule or when some nodes leave the network, downstream in-degree values will change for different nodes, reflecting the new topology.

4.2.4 Data dissemination methods

Different approaches were proposed in the literature trying to solve the problem of propagating data into a wireless ad-hoc network and keeping it consistent network-wide. A widely-used approach is delivered by Trickle [LPCS04], a polite gossip protocol designed for low-power multihop ad-hoc networks. The two main ideas behind Trickle are the use of an adaptive transmission rate and the suppression of redundant messages, i.e. nodes only disseminate data if they have not seen *consistent* data enough times in a chosen listen-only time interval. Trickle's adaptation of the transmission rate works as follows:

1. At the beginning of a node's operation, set the Trickle interval I to the chosen minimum interval I_{min}, i.e. $I := I_{min}$.

2. For every potential transmission, at the beginning of the current Trickle interval choose a random transmission point t in the interval $[\frac{I}{2}, I]$.

3. After interval I has passed, double the interval or after enough doublings set the interval to the maximum possible interval, i.e. $I := min(2 \cdot I, I_{max})$.

4. If at any moment inconsistent data is observed, re-start the interval I, i.e. $I := I_{min}$.

This means, Trickle starts with a high transmission rate that decays exponentially with time as long as no inconsistent data is observed. Most Trickle-based protocols resort to version numbers in order to differentiate data artefacts and detect inconsistencies in the network. As already mentioned, in addition to transmission rate adaptation, nodes try to minimize redundant transmissions by counting the number of received consistent messages in each transmission interval and only transmit the same data if a certain threshold K is not reached.

Trickle's ideas have been adopted (often with some variations) by many protocols such as Drip [Tol05], Dip [LL08] and Dhv [DBFP09]. For instance, TinyOS [LMP+05], an operating system designed for sensor networks in use by hundreds of research groups throughout the world, has implemented Dip and Dhv for lightweight data dissemination. Even though Trickle has been shown to be quite scalable, its simplicity proves to be problematic under different scenarios leading, for instance, to information partitioning.

A recent variant of Trickle, A^2-Trickle, adds enhancing mechanisms such as:

- Aligned intervals: neighboring nodes align their Trickle intervals through tick synchronization to minimize message collisions.
- Adaptive suppression mechanism: nodes change the suppression counter K for every Trickle interval by keeping track of the topology around it. In addition, if a node v detects that a certain node u only receives information from it, v always transmits new information regardless of K.

The authors showed with testbed-based experiments and simulations with more than a hundred different topologies that A^2-Trickle leads to fewer collisions than the vanilla Trickle algorithm, forwards messages faster and leaves no node in an inconsistent state. Moreover, it makes configuring the dissemination protocol easier by making the suppression threshold adaptive and is also more energy-efficient.

Even though Trickle-based data dissemination protocols have found wide adoption, there is still a dearth of algorithms concerned with multichannel data dissemination. Some works focus only on the channel selection aspect (leaving out any discussion of data consistency) such as [ACKR], where the authors measure worst-case transmission latency when using five different channel access schemes.

An interesting approach, which is comparable to our approach is SURF [RVKF13]. In a similar fashion to our stabilization protocol, SURF assigns different weights to channels. However, in contrast to our protocol, nodes are only equipped with a single transceiver and channel selection is decided on a node-by-node basis: nodes always use

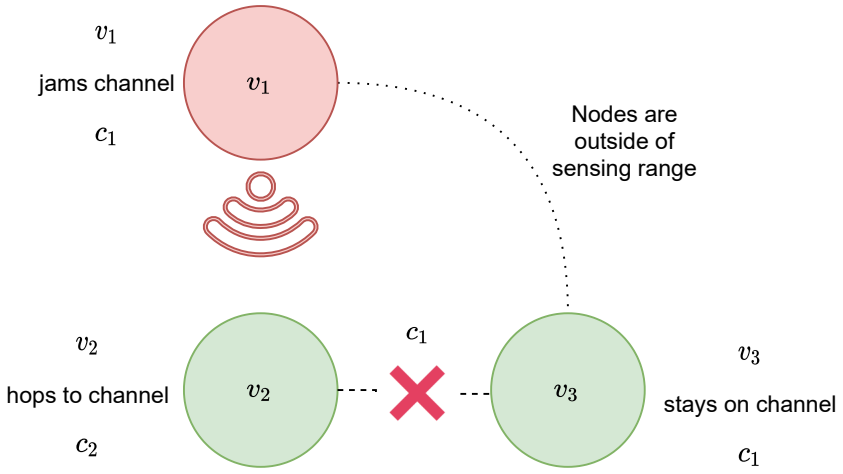


Figure 4.4: Hidden stations make it rather problematic for nodes to always use the best local channel for communication.

the best local channel for communication. Channel weights are proportional to primary user occupancy of the channel and the number of neighbors on the channel. One of the main assumptions made by SURF is that neighbors will rank channels similarly and will hence with high probability pick the same channel for communication. Different from our algorithm SURF node might overcrowd channels with good conditions instead of distributing the channel usage over multiple channels. Furthermore, even though it might minimize channel switching, always picking the best local channel is much more susceptible to scenarios where the channel quality suddenly becomes considerably worse due to hidden stations, e.g. a node v_3 has only one neighbor v_2 and they communicate on channel c, when a foreign node in communication range of v_2 and outside the sensing range of v_3 starts jamming channel c. This would make it effectively impossible for v_2 and v_3 to communicate with each other, because whereas v_3 will stay on channel c, since for it the channel is still very good, v_2 will switch to its next best channel (see Fig. 4.4).

Another two data dissemination protocols that consider multichannel access are McTorrent and McSynch [SHFS05]. McTorrent was designed to allow the end-to-end dissemination of large data objects such as over-the-air reprogramming and adopts many of the ideas introduced by Deluge [CHT04] while extending them to work in a multichannel scenario. McTorrent nodes use advertisement and request messages to negotiate the transmission of data pages (portions of the data object) on a given channel. McSynch, on the other hand, focuses on keeping data consistent within two-hop clusters. With this purpose, it uses a round-based approach and divides the time into slots. In each round, data pages are assigned to both a channel and a slot in order to produce collision-free

transmission within the cluster, i.e. only one node transmits on any given channel in any given slot. Both McTorrent and McSynch only marginally consider data consistency, McSynch a bit more than McTorrent, but even so in a simpler scenario than ours (the consistency is only kept within two hops).

4.2.5 Communication schedule dissemination

In our scenario, the master node v_{master} is responsible for computing a communication schedule s_{com} based on collected channel quality reports and on its own local channel quality measures. The synthesized s_{com} must be then disseminated into the network and be used as a global hopping sequence for communication. This means that one of the main tasks of our stabilization protocol is to keep the network schedule-consistent, i.e. all nodes possess the same hopping sequence.

Communication schedules are propagated through the network with special messages, namely schedule reports. Similarly to channel quality reports, these messages are broadcast from each node with a certain frequency.

Besides the communication schedule itself, schedule reports carry additional metadata such as the version number of the current schedule. This helps other nodes distinguish between new and old schedules and also helps nodes decide which schedule to use (the one with the higher version number) when multiple nodes advertise different schedules.

It is worth noting that nodes advertise their communication schedules even when no change took place. This is important, for instance, for nodes joining the network, which otherwise would have no communication schedule until it got re-computed and re-disseminated.

Our basic dissemination protocol works as follows:

1. For every n_{slot} slots, every node broadcasts its current commnunication schedule at maximum $n_{dis} \in \mathbb{N}$ slots.

2. Each node keeps track of how many *consistent* schedule reports it received since its last attempt at broadcasting a schedule report.

3. If a node is on a report channel and it has received less than n_{adv} schedule reports and it has broadcast fewer than n_{dis} schedule reports in the past n_{slot} slots, then it picks a random time point in the current slot and reaching that point it advertises its current communication schedule. Otherwise it stays silent, resets its report counter and starts counting anew. Choosing random time points for the broadcasts

tries to minimize collisions between nodes in communication range that select the same channel in the same slot for the schedule reports.

By using this *polite* gossip scheme inspired by Trickle [LPCS04], we lower the number of redundant schedule advertisements, since every node only advertises its current communication schedule if not enough schedule reports were broadcast in its vicinity.

In order to avoid having the same nodes advertising all the time while others stay silent, each node picks a random time point within each slot at which its schedule report should be broadcast.

At every time slot, each node has to decide whether the current channel can be used for a schedule report or not.

This channel selection relies primarily on locally measured channel qualities. Channels are ranked according to aggregated channel quality and the n_{best} local channels are selected for schedule reports in the descending order of channel quality. In addition, nodes do not re-use channels until all suitable channels have been selected. This channel selection scheme makes it easier for joining nodes to receive a new schedule by broadcasting the current communication schedule on multiple good channels. Furthermore, as an optimization, if a node v receives an inconsistent schedule report from node u, it may broadcast its next schedule report on the next channel that is also good for u, even if this channel is outside the subset of the n_{best} local channels for v, as long as this channel comes up earlier in the communication schedule. This exception to the channel selection scheme intends to speed up the elimination of schedule inconsistencies within a one-hop neighboorhod. For this scheme to work, nodes have to keep track of the channels on which their neighbors sent schedule reports in past slots. Furthermore, in order to avoid nodes keeping stale information on neighbors, channels are not considered good for neighbors anymore if they were not used for n_{slot} slots for broadcasting a schedule report.

4.2.6 Optimal and temporary schedules

Based on current locally measured channel conditions and the estimated network-wide channel conditions, derived from channel quality reports, the master node computes an optimal communication schedule. However, if this schedule is not the first communication schedule computed by the master node since bootstrapping the network, and this schedule has just been computed, the master node does not disseminate this new schedule in a single schedule report. Instead, the master reports *schedule deltas*, i.e. changes

to the previous schedule that have to be performed to converge to the new optimal schedule.

This helps keeping the variation in schedule-consistency due to new schedules bounded. This avoids having nodes who still have not received the new schedule from drifting too far apart from the new schedule. A certain level of drifting is unavoidable, especially in a larger network, where a number of nodes will most probably miss some schedule reports due to frame losses resulting from frame collisions or overall bad channel conditions. These schedule drifts should be most prominent at the border of the network, since the nodes at this region are the furthest apart from the master node. In order to keep this schedule drifiting contained, we limit the sizes of the schedule deltas that are disseminated into the network. The minimal possible schedule delta is called an *atomic substitution* and is the pair $(i, c^\star)$, where i denotes the slot i where channel $c^\star$ should be used, i.e. for a schedule s, $s[i] \leftarrow c^\star$. While having schedule reports carry atomic substitutions lead to minimal divergence it also leads to the slowest convergence from the current communication schedule to the new optimal one. This means that it might be advisable for a schedule delta to carry more than one atomic substitution, especially if the depth of the network grows, i.e. there is an increase in the maximum number of hop counts from the master node to the border of the network.

In addition, in order to steer towards the best possible network performance, also during the convergence process between two communication schedules, atomic substitutions are chosen such that the resulting schedules are the best possible *intermediate* schedules after a single substitution. This substitution is computed considering the aggregated channel qualities perceived by the master node and the resulting optimal channel utilizations. For more details on how to compute such optimal atomic substitutions, please refer to [Eng20].

After a certain time has passed since the dissemination of the new optimal schedule (see Sec. 4.2.8), nodes assume the network has been stabilized and is schedule consistent. As an optimization, in order to have joining nodes receive the current communication schedule faster, nodes might switch to broadcasting the *whole* communication schedule in a single schedule report.

4.2.7 Re-computation of communication schedules

In our scenario, the number of slots assigned to each channel is proportional to its quality at the time of computation of the current hopping sequence. In general, this channel selection remains static for multiple time slots and is only updated upon detection of

significant changes in channel quality. On top of that, we do not want to re-compute the communication schedule too often, since this could lead to unnecessary schedule inconsistencies throughout the network.

Every d_{comp} milliseconds, the master node tries to gauge whether its aggregated channel qualities (of both its measured qualities and the channel reports) signal a change in overall channel quality for it to re-compute and disseminate a new schedule.

First, we need to define what constitutes a significant change in overall channel quality. We define a significant change with help of the throughput variance.

Definition 4.2.7. (Throughput variance). Given a current communication schedule s, computed based on previously measured channel qualities, a schedule s' computed by the master node based on the current aggregated channel qualities, as well as $\tau_{EAT}(s)$ and $\tau_{EAT}(s')$ the normalized expected achievable throughput of respectively s and s', we define the throughput variance $R(s, s')$ as:

$$R(s, s') = \frac{\tau_{EAT}(s') - \tau_{EAT}(s)}{\tau_{EAT}(s)} \tag{4.19}$$

$R(s, s')$ calculates the the net gain or loss in expected achievable throughput relative to the current $\tau_{EAT}(s)$ by swapping the current communication schedule s for a new one s', computed based on the current channel state. For instance, if $R(s, s')$ is negative, it indicates that there is a loss utility-wise in disseminating a new schedule, from the perspective of the master node. This is clearly an estimate, derived from aggregated channel qualities at the master node, of the expected payoff for each node in using the new schedule and might differ from local views. However, due to how channel quality reports are aggregated, we try to align the view of the channel state at the master node with the view of the majority of the network giving more weight to better connected nodes and nodes which benefit the least from the current communication schedule.

Given a pre-configured threshold δ_{ret}, we only adopt a new communication schedule if the throughput variance rises above this threshold, i.e. $R(s, s') > \delta_{ret}$. This threshold should be chosen in a conservative manner to only reflect meaningful changes in overall channel quality, e.g. $\delta_{ret} = 0.2$.

4.2.8 Estimating d_{comp}

Regardless of channel quality fluctuations, synthesizing new communication schedules and subsequent dissemination should not happen too often, allowing for the network to enjoy the current stabilized state.

The parameter d_{comp} defines the minimum time the master node waits between schedule computations. This waiting time depends on how long it takes for the network to transition from inconsistency to network-wide schedule-consistency.
Given

- d_{slot} the duration of a slot
- n_{slot} the number of slots in a hopping sequence
- $n_{maxHops}$ the maximum distance in hops from a node to the master node (see Figure 4.5)
- n_{dis} the maximum number of times every node broadcasts its current schedule within n_{slot} slots.

we estimate the time duration $d_{stabilization}$ needed to stabilize the network:
A very optimistic estimation. Assuming that the new schedule is propagated downstream one hop for every slot, we can calculate the time for the whole network to become schedule consistent with the master node as

$$d_{stabilization,opt} = n_{maxHops} \cdot d_{slot} \tag{4.20}$$

Example 4.2.1. For $n_{maxHops} = 5$, $d_{slot} = 500ms$, we have:

$$d_{stabilization,opt} = 5 \cdot 500ms = 2.5s$$

Even though this estimate of the stabilization time is realistic enough, depending on n_{dis} and on the current channel conditions, it might be way too optimistic. That is why we will try to be a bit more conservative.

A more conservative estimation. Assuming that the new schedule advances n_{dis} hops downstream from the master node for every n_{slot} slots, we can estimate the time for the whole network to become schedule-consistent with the master node as

$$d_{stabilization,fast} = \left\lceil \frac{n_{maxHops}}{n_{dis}} \right\rceil \cdot n_{slot} \cdot d_{slot} \tag{4.21}$$

Example 4.2.2. For $n_{maxHops} = 5$, $n_{dis} = 3$, $d_{slot} = 500ms$ and $n_{slot} = 12$ we have:

$$d_{stabilization,fast} = \left\lceil \frac{5}{3} \right\rceil \cdot 12 \cdot 0.5s = 2 * 12 * 0.5 = 12s$$

A less optimistic estimate. For a more pessimistic estimate, we assume one hop propagation per n_{slot} slots of the new schedule through schedule reports. This way, we have

$$d_{stabilization,slow} = n_{maxHops} \cdot n_{slot} \cdot d_{slot} \tag{4.22}$$

Example 4.2.3. For $n_{maxHops} = 5$, $n_{dis} = 3$, $d_{slot} = 500ms$ and $n_{slot} = 12$ we have:

$$d_{stabilization,slow} = n_{maxHops} \cdot n_{slot} \cdot d_{slot} = 5 \cdot 12 \cdot 0.5s = 30s$$

Based on the estimates described above, we constrain d_{comp} to lie between our more conservative estimates:

$$d_{stabilization,fast} \leq d_{comp} \leq d_{stabilization,slow}$$

Estimating $n_{maxHops}$ Since the network topology is dynamic we need to repeatedly estimate the depth $n_{maxHops}$ of the network in order to keep d_{comp} up to date. In order to accommodate for this, channel quality reports can carry not only the hop count of the current broadcasting node, but also the maximum seen hop count, allowing for the current depth of the network to eventually propagate to the master node (see Fig. 4.6). Since changes in the network (nodes joining or leaving it) may have rendered hop counts temporarily invalid, this yields only an estimation of $n_{maxHops}$.

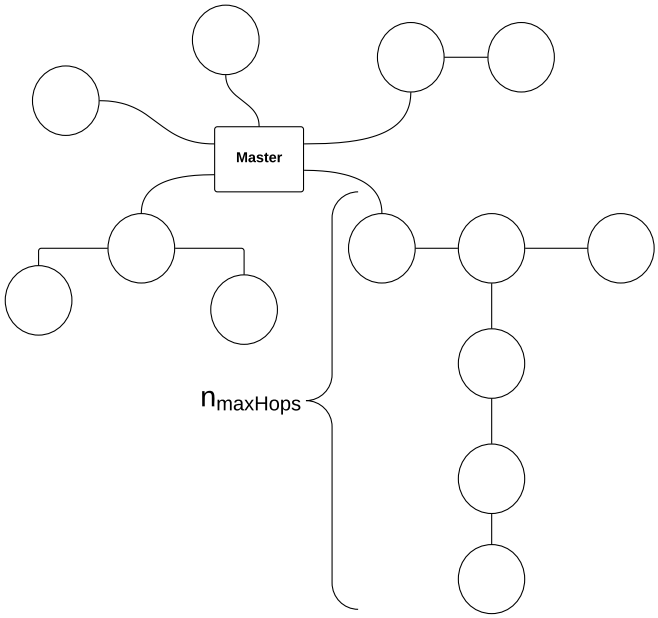


Figure 4.5: $n_{maxHops} = 5$ is the estimated maximum distance in hops from a node to the master node

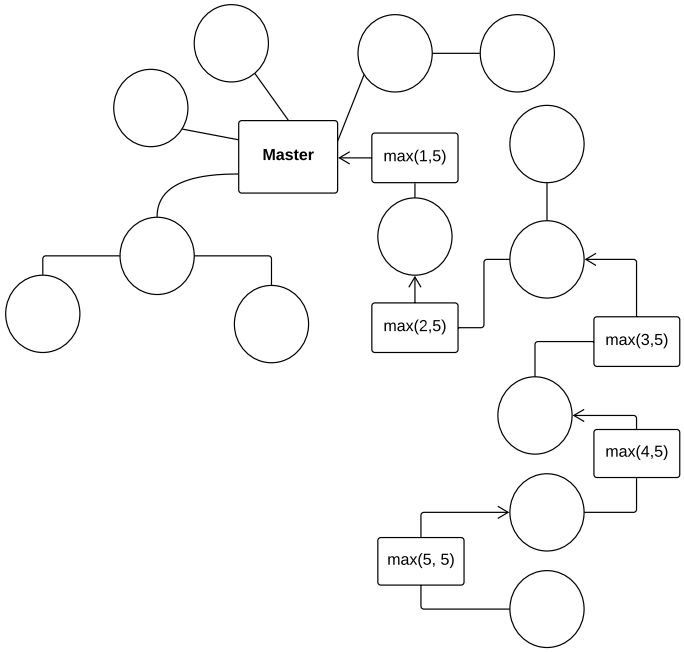


Figure 4.6: Maximum seen hop count piggy-backs on channel reports and yields an estimate of the depth of the network at the master node

4.3 Leader election

Since the computation and dissemination of communication schedules rely on a master node, a still missing primitive in our stabilization protocol is how to elect a new master node when the current one fails. This problem is referred in the literature as leader election. Hence, in this section we use the term leader and master node interchangeably. Different leader election algorithms have been proposed in the literature under different network settings such as [GH12] and [CD15].

4.3.1 Master failure

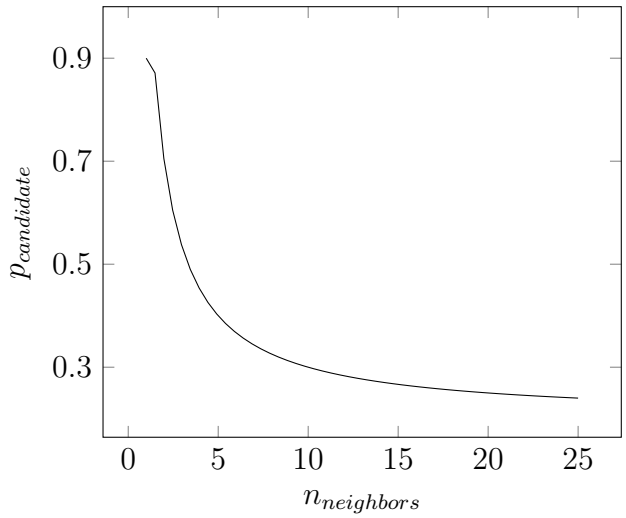
We propose a simple leader election algorithm, where one-hop neighbors become candidates after detecting a possible master failure. First, a possible master failure is detected by a node v when no tick frame is received by v from the master for n_{fail} slots, where $n_{fail} > 1$. Nodes pick n_{fail} locally in a conservative manner based on locally perceived channel qualities such that the probability of the master node having failed $p_{fail} = 1 - p_{loss} > 0.9$, where p_{loss} is the probability of all tick frames being lost, which in this case becomes less than 0.1. We assume that the quality of each channel during a given slot equals the probability of a tick frame being delivered to a certain neighbor of the master node during this slot. This means nodes seeing different channel conditions will assume master node failures at different slots.

Example 4.3.1. Let us assume every channel c has a quality $q_c = 0.5$, which constitutes overall bad but still usable channel conditions. We can then compute the probability of all tick frames being lost after 4 slots as $p_{loss} = (0.5)^4$. In this case, after observing 4 consecutive tick frames losses, in the fifth slot the node assumes the master node has failed, since it calculates the probability of the master having failed as

$$p_{fail} = 1 - (0.5)^4 \approx 0.94$$

Example 4.3.2. A node using channels $C = \{c_1, c_2, c_3, c_4\}$ with respective qualities $\vec{q} = (q_1, q_2, q_3, q_4) = (0.7, 0.6, 0.4, 0.9)$, and expecting a tick frame on channel c_4 will derive $p_{fail} = 1 - (0.1)^2 = 0.99$, assuming already after $n_{fail} = 2$ missed tick frames that the master node has failed.

Upon detecting a potential failure, a node *may* become a candidate for the leadership of the network. By having only neighbors of the previous master node become candidates, we try to minimize the changes in the topology of the network resulting from a



change in leadership. In addition, to lower the number of leader candidates, after detecting a master failure each node decides at every slot with probability $p_{candidate}$ whether to become a candidate, as long as it is not a candidate already or has already chosen a new leader. This probability is chosen by each node based on its current number of neighbors $n_{neighbors}$:

$$p_{candidate} = \min(0.9, 0.2 + \frac{1}{n_{neighbors}}) \tag{4.23}$$

For nodes with 5 neighbors for example, the probability of all of them choosing *not* to become a candidate after one slot is $p_{\neg\forall candidate} = (1-p_{candidate})^{(n_{neighbors}+1)} = (0.2+\frac{1}{5})^6 = (0.4)^6 \approx 0.004$, which is already quite low. The probability of all of them not becoming candidates after two slots is $0.004^2 \approx 0.000016$ and hence negligible. In fact, in a n-node neighborhood we expect on average to have $n \cdot p_{candidate}$ candidates per leader election.

In addition to the probabilistic candidacy, nodes use random voting timers to cast votes for themselves or others in each election slot, i.e. where no leader is present. By combining a probabilistic candidacy with randomized timepoints for casting the votes, we considerably reduce the probability of persisting split votes.

4.3.2 The voting process

Nodes that decide to become candidates broadcast votes for themselves, one vote per slot. Neighbors of a candidate v who are not yet candidate themselves and who have not received any other vote yet, become followers of v upon receiving a vote from it. Followers choose a random time point in the current slot and broadcast a vote for the

supported candidate. Each node votes for at most one candidate in any given slot. Having followers of a node v cast votes for v has a dual purpose: making v aware that these nodes are now its followers as well as telling other nodes (not in communication range of v) that v is a leader candidate.

Furthermore, each vote carries a timestamp with nanosecond precision of the time when the supported node cast its first vote for itself (announced its candidacy). If more than one node within the same neighborhood have voted for themselves in the same slot, a deterministic tie-breaking rule is applied: the older candidate wins, i.e. the node with the lower candidacy timestamp value stays a candidate while the other node becomes its follower. Note that it is irrelevant whether the lower value really proves that one vote was cast before the other, it is only important to notice that the probability of both nodes having the same timestamp is negligible. In addition, any followers of the losing candidate will eventually support the other candidate.

At the beginning of each slot, a leader candidate chooses a random time point in the current slot and casts a vote for itself again. After 3 consecutive slots with no competing votes, a candidate v considers itself a master node. This is a conservative waiting time to cope with the loss of votes or a delayed propagation of votes especially for candidates outside the communication range of v.

During the voting slots, as long as no leader is elected, there is no dissemination of changes to the current communication schedule. However, to cope with re-synchronization of nodes in the absence of a master node, votes in each slot act as tick frames, i.e. carry the current time slot. This allows follower nodes to re-synchronize to the ticks of the current supported node or to followers of the same node.

4.3.3 Raft

The basic ideas behing our leader election algorithm are mainly inspired by the leader election proposed by Raft [OO14], a widespread distributed consensus algorithm, which is considered to be a simplified version of Paxos [Lam05]. Similar to our work, failures due to malicious nodes, i.e. Byzantine failures, are not considered by Raft. Also, similarly to Raft we let *older* candidacies win. However, while we prefer to use the time point where the candidacy started, Raft uses election rounds (terms), which increase monotonically until the election terminates and candidates with higher terms win. The termination in Raft takes place when a node acquires the majority of votes. This would not work as easily in our scenario since nodes do not know how many potential leader candidates there are in the network at the time of the master failure and not all of them can

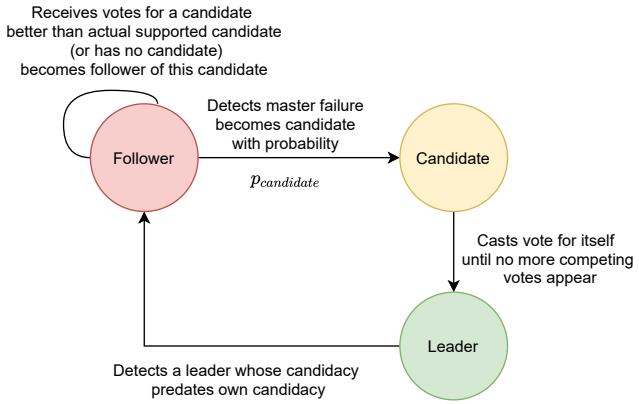


Figure 4.7: Simplified state machine of leader election.

communicate with each other. In fact, one fundamental difference between Raft and our stabilization protocol is its implementation assumption that all nodes in a cluster know of each other and no nodes might join the cluster dynamically. In our scenario, nodes might be aware of only some of the nodes in communication range, and we do not have any statically configured topology.

Same as in Raft, we assume links between nodes are by nature lossy and unreliable and we cannot guarantee that network partitions won't occur. Indeed, depending on the current topology of the network, the failure of the master node might leave the network partitioned and each remaining cluster will eventually elect its own master node. This partitioned state will remain as long as no bridge between both partitions is built, be it new nodes joining the network or nodes moving around, which might change the communication topology. Since network partitions are possible, cluster leaders should eventually become followers of other leaders whose candidacy timestamp is lower. To achieve this goal, we extend tick frames to carry the original candidacy timestamp of the current leader, allowing two partitions with distinct leaders to be merged later on. A simplified state machine can be seen in Fig.4.7 illustrating the state transitions between the `follower`, `candidate` and `leader` states.

4.4 Initial stabilization

In order to minimize bootstrapping overhead, i.e. additional messages on the spectrum to negotiate the joining of a new node, we adopt a conservative *passive* approach: a joining node v stays passive until a number of constraints are satisfied:

1. Node v must have received at least one sync message and hence be synchronized with the master node.

2. The node must be schedule-consistent with at least one node in its communication range, i.e. it must have received a communication schedule. If a node is synchronized and possesses a communication schedule it is considered *stable*.

This conservative transmission strategy also helps avoid skewing the derivation of channel qualities due to chatty joining nodes.

To guarantee that these these constraints are eventually satisfied, we have introduced different measures:

1. After powering up, all nodes start sensing all channels and compute for each of these channels a channel quality. Since we use *passive* channel sensing no further measure must be taken before *unstable* nodes start performing it. Measuring channel qualities is performed during the whole life-cycle of a node.

2. Given q_{min} the minimum acceptable channel quality for effective communication, after powering up the master node detects the set of channels with sufficient quality $C_{min} = \{c \in C \mid q_c > q_{min}\}$ and computes a communication schedule using C_{min}. Since the master node is always trivially synchronized with itself, after computing a schedule it is *stable* and can become active on the medium.

4.4.1 Initial synchronization

To achieve an initial synchronization, unstable non-master nodes have to receive at least one tick frame (a sync message). For this purpose, each node v computes $v.C_{min} = \{c \in C | q_c \geq q_{min}\}$ and listens for a tick frame on each of the channels $c \in v.C_{min}$ in descending order of channel quality. Nodes listen on each channel for a duration of $n_{slot} \cdot d_{slot}$ before hopping to the next channel. This process is repeated until a tick frame is received.

Since for each n_{slot} slots multiple tick frames are sent with high probability on the same channel, this behavior increases the probability of eventually receiving a tick frame. This way, as long as an unstable node has a stable node in communication range, it will eventually receive a tick frame. Furthermore, by starting with the best local channel, a node increases the probability of receiving the tick frame in the first listening period, since the preferred channel for broadcasting tick frames is the channel that had the best aggregated quality at the time of computation of the communication schedule.

If a node has already received a communication schedule but no tick frame, it already knows on which channel the tick frame should arrive with high probability (the channel with the most slots). In this case, the node listens on this channel, and if no tick frame is received after waiting for $n_{wait} \leq n_{slot}$ time slots, the node hops to one of the channels that is used directly after the channel with the highest utilization and waits again. This procedure is repeated anew until a tick frame is received.

4.4.2 Initial communication schedule

Upon receiving a tick frame, a node v knows that the sending node u must be a stable node and that this node will eventually broadcast a schedule report. With this in mind, stable nodes include their best local channel c_{best} in the tick frame such that *synchronized* nodes without a communication schedule can switch to c_{best} and wait there until they have received a communication schedule. This essentially guarantees and speeds up the eventual reception of a schedule report by node v, since nodes broadcast schedule reports on their best local channels and the best local channel of a node u (where the received tick frame originated) will eventually be used for schedule reporting by u and with high probability by any node in communication range of v and u.

After receiving a communication schedule, synchronized nodes can then become active on a channel and appropriate management data can hence be broadcast, in particular, tick frames, channel quality reports and schedule reports.

4.5 Simulation

In this section, we will introduce some metrics to gauge the level of schedule consistency throughout the network w.r.t. the global communication schedule. In addition, we will evaluate the capabilities of our stabilization protocol to maintain and restore network-wide schedule consistency by discussing some of the results obtained through simulations of the protocol.

4.5.1 Schedule consistency Metrics

To analyze the performed experiments, we introduce some schedule consistency metrics.

Definition 4.5.1. (Degree of Consistency). Given s_{com} the global schedule disseminated into the network by the master node, and s the schedule of a node v, we define the degree of consistency Γ of schedule s with respect to s_{com} as

$$\Gamma(s, s_{com}) = \frac{1}{n_{slot}} |\{\, 1 \leq m \leq n_{slot} \mid s(m) = s_{com}(m) \,\}|$$

This way $\Gamma(s, s_{com})$ counts the number of matching slots between schedule s and schedule s_{com}.

Example 4.5.1. (Degree of Consistency). Given a network with 5 macroslots per superslot, i.e. $n_{macro} = 5$, and the schedules of the master node ($v_{master}.s$) and of node v ($v.s$), we compute the degree of consistency Γ of v:

$$\begin{bmatrix} v_{master}.s \\ v.s \end{bmatrix} = \begin{bmatrix} \mathbf{1} & \mathbf{5} & 8 & 4 & \mathbf{1} \\ \mathbf{1} & \mathbf{5} & 6 & 3 & \mathbf{1} \end{bmatrix}$$

Since the schedules match at three slots (shown in bold), the degree of consistency of node v is

$$\Gamma(v.s, v_{master}.s) = \frac{3}{5}$$

Definition 4.5.2. (Average Degree of Consistency). Given s_{com} the global communication schedule and let $v.s$ denote the communication schedule in use by node $v \in V$, we define the *average degree of consistency* $\overline{\Gamma}(s_{com})$ as

$$\overline{\Gamma}(s_{com}) = \frac{1}{|V|} \sum_{v \in V} \Gamma(v.s, s_{com}) \tag{4.24}$$

$\overline{\Gamma}$ delivers hence a metric to gauge the level of network-wide schedule consistency with respect to the global schedule by computing the average Γ over all schedules in use in the network. For the sake of notational simplicty, we will from here on refer to the average degree of consistency as $\overline{\Gamma}$, omitting the global communication schedule from the notation.

In addition to $\overline{\Gamma}$, we define an alternative schedule consistency metric: the *network consistency ratio*.

Definition 4.5.3. (Network Consistency Ratio). Given all nodes $v \in V$, let $v.c$ be the channel in use by v during the current slot and $v_{master}.c$ the channel being used by the master node, we define the network consistency ratio Δ as

$$\Delta = \frac{1}{|V|}|\{\, v \in V \mid v.c = v_{master}.c \,\}| \tag{4.25}$$

This means Δ computes the fraction of nodes that listen on the same channel as the master node during a given slot.

Example 4.5.2. (Network Consistency Ratio). Given a network with five nodes, i.e. $|V| = 5$, where V is the set of nodes, the master node is currently using channel 2 and $v_i.c$ denotes the channel currently being used by node v_i. We compute the network consistency ratio Δ, i.e. the fraction of the whole network that currently uses the same channel as the master node:

$$\begin{array}{cccccc} v_{master}.c & v_1.c & v_2.c & v_3.c & v_4.c \\ \left[\begin{array}{ccccc} \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{2} & 4 \end{array}\right] \end{array}$$

After comparing the channels being used by every node in the network, we can compute Δ as in (4.25). Since v_1, v_2, v_3 and v_{master} all use the same channel, the network consistency ratio is

$$\Delta = \frac{4}{5}$$

4.5.2 Simulation environment

We have implemented our stabilization protocol in the ns-3 simulation framework. ns-3 is a discrete event network simulator that allows us not only to evaluate the performance of the main components of our stabilization protocol, but lets us to do so with different topologies and with a network size larger than our testbed. Moreover, simulation plays an important role in network research in general by making results reproducible and easier to share with the research community. ns-3 has become the de facto go-to network simulator in special for wireless networks. Some of the reasons for its wide adoption in the networks research community are:

1. Modularity (module-based).

2. Extensive documentation and active discussion forums.

3. Integrated unit testing framework, which makes test-driven development easier.

4. Development relies on a single programming language: C++.

4.5.3 Scope of the simulation

In our simulation, we first focus on the main functionalities of our protocol: computation and dissemination of communication schedules triggered by aggregated channel qualities, which are measured and reported from non-master nodes to the master node. The main focus of the results shown here is then to determine how well our stabilization protocol can keep the network schedule consistent. Left out of the simulation results focusing on schedule consistency are synchronization aspects (all nodes are always synchronized), the leader election mechanism (the master node is always present) and the initial stabilization process (all nodes can become active from slot 1 onward). The purpose of constraining the scope of the simulation is twofold: to spend more time evaluating the main functionalities and minimize the number of distinct active components in order to avoid making the simulation too complex, which would make it more difficult to validate the main functionalities of the stabilization protocol. After evaluating the main capabilities of our stabilization protocol, we briefly evaluate our tick synchronization algorithm through a couple of simulation experiments.

4.5.4 Channel sensing

Nodes in our simulation measure channel qualities by reading stored quality values for the current channel. Each node applies a random variation to these channel qualities (usually with a conservative variation upper bound, e.g. 0.2) to simulate the different foreign traffic patterns in each region of the network. The base channel qualities used for the simulations are historical records stored in our database from real channel quality measurements performed by our testbed nodes. This helps us use more realistic quality values while still allowing some random variations. This combination of base quality values and bounded variations helps us create different scenarios where channel conditions throughout the network are distinct with different levels of heterogenicity.

4.5.5 Physical model of the wireless channel

In our simulations, we use a physical model for the wireless channel that uses a propagation model that only considers large-scale fading disregarding large object shadowing effects. This means the signal attenuation depends solely on the distances between transmitter and receiver. Moreover, the used physical model uses a constant propagation delay. Furthermore, nodes successfully receive messages from a transmitting node with a probability corresponding to the quality of the current channel (measured by the receiver) if and only if they are within the transmitter's communication range. The used model allows us then to create different multi-hop topologies in the network by either changing the position allocation or the communication range of the nodes.

4.5.6 Experiments

In this section, we will discuss some of the simulation results achieved with the ns-3 implementation of our stabilization protocol.

Network self-healing

The following experiments show how our stabilization protocol is able to cope with high losses in schedule consistency throughout the network. As seen in Fig. 4.8, the network can self-heal, i.e. it eventually converges from a very low value of $\overline{\Gamma}$ towards the best possible average degree of consistency $\overline{\Gamma} = 1$. For this experiment, the simulation was configured to run for 500 slots. The total number of channels is 12 with a communication schedule size of $n_{slot} = 30$. Moreover, the network has 80 nodes arranged in a grid with 5 meters between nodes both horizontally and vertically and the maximum transmission range of all nodes is 18 meters. The maximum network diameter resulting from this configuration was 26 hops. Despite the huge drop in schedule consistency at the beginning and the long convergence time, the network was fully schedule consistent 43.6% of the time.

As expected, increasing the rate of dissemination of schedule reports helps decrease the convergence time needed to reach the maximum average degree of consistency. For instance, simulating the previous scenario and doubling the n_{dis} parameter, i.e. the maximum number of disseminated schedule reports within every n_{slot} slots, the same drop in schedule consistency is repaired almost 100 slots faster, as seen in Fig. 4.9. As shown in Fig. 4.10, the convergence towards network-wide schedule consistency is not always monotonically increasing. Drops in the average degree of consistency can

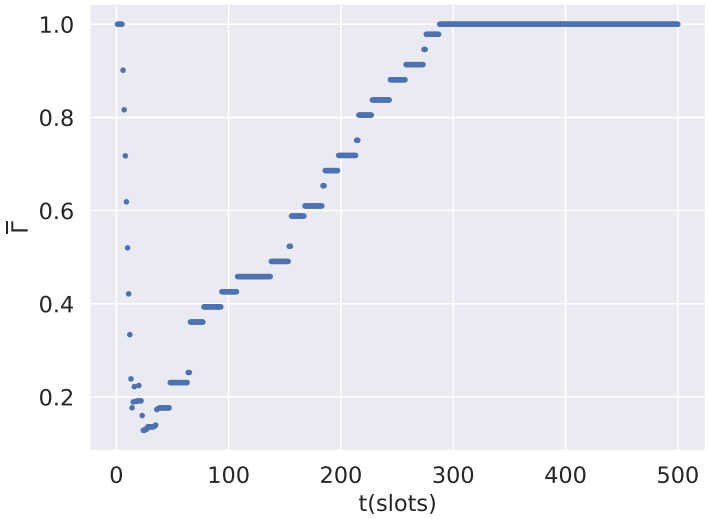


Figure 4.8: A huge drop in schedule consistency happens at the beginning of the simulation and is progressively repaired.

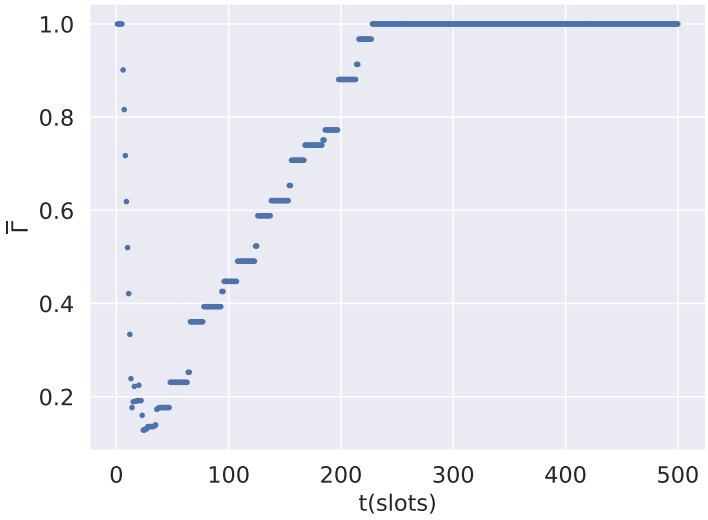


Figure 4.9: The same drop in schedule consistency as in Fig. 4.8 is repaired almost 100 slots faster doubling the maximum number n_{dis} of disseminated schedule reports for every n_{slot} slots.

happen during the convergence due to the differences in the dissemination rate of new schedule deltas (regardless of whether they are from the same schedule towards which nodes are already converging or from a newly computed communication schedule) and the propagation delay of these deltas through multiple hops (which takes multiple slots).

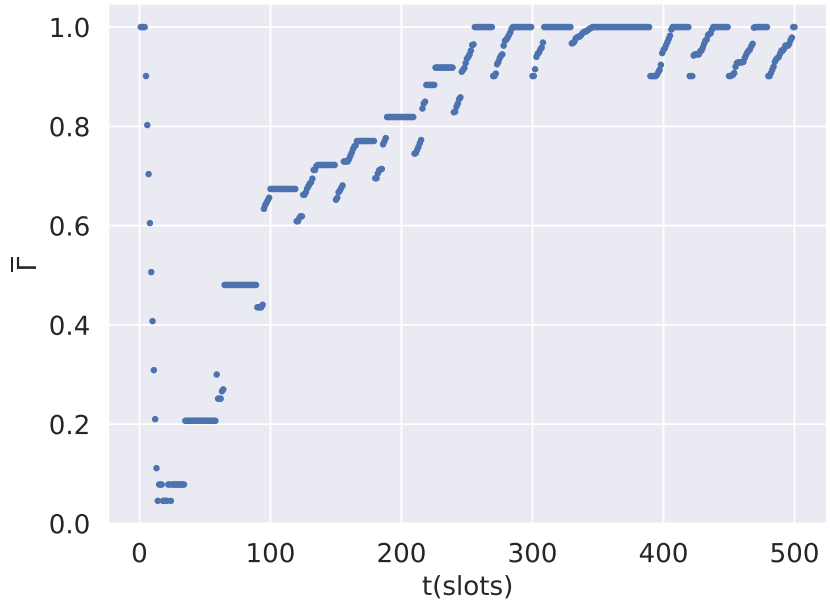


Figure 4.10: Convergence to $\overline{\Gamma} = 1$ is not always monotonically increasing.

Channel conditions and communication schedule re-computation

Despite the robustness of our stabilization protocol, it can only maintain and restore network-wide schedule consistency if minimal working conditions are provided.

In our next simulation, we had a network with 80 nodes using the same position allocation as the previous simulations, but with an increased maximum transmission range of 20 meters, resulting in a maximum network diameter of 15 hops. In order to test the limits of the self-healing capabilities of our protocol, for this simulation we used channel qualities with higher volatility and set $d_{comp} = 0$, effectively removing the rate-limit for the dissemination of a new communication schedule. As can be seen in Fig. 4.11, since channel conditions are varying too much and the master node has no rate-limitation for computing new schedules, we end up having excessive re-computations that prevent the network from ever reaching $\overline{\Gamma} = 1$, even though the network always steers towards network-wide schedule consistency. However, by setting $d_{comp} = 25$, we not only restrict the number of possible re-computations, but also allow our stabilization protocol to reach and hold $\overline{\Gamma} = 1$ for multiple consecutive slots until the next significant change in channel conditions triggers another re-computation (see Fig. 4.12).

Even if we can already limit to a certain degree the rate of dissemination of new communication schedules by using $d_{comp} > 0$, we still have another parameter that

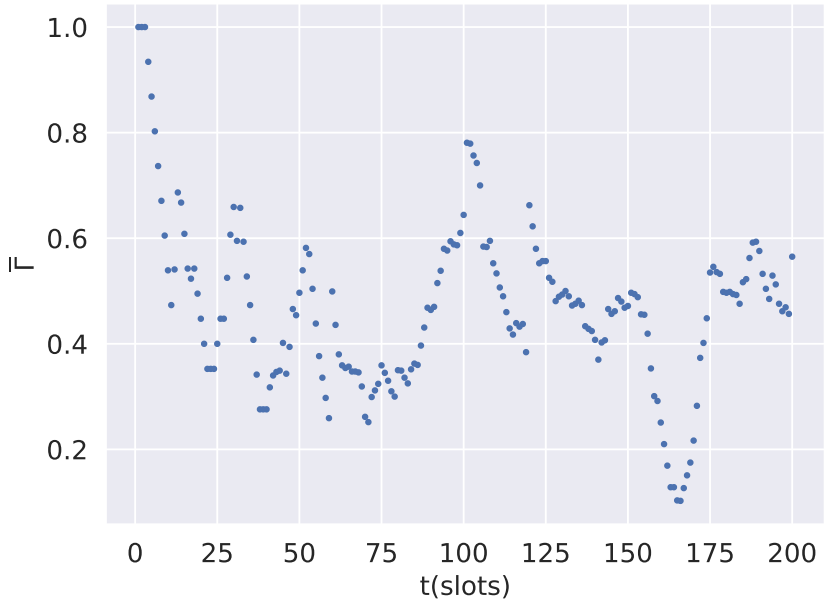


Figure 4.11: Our stabilization protocol steers towards $\overline{\Gamma} = 1$, but due to excessive re-computations never reaches it.

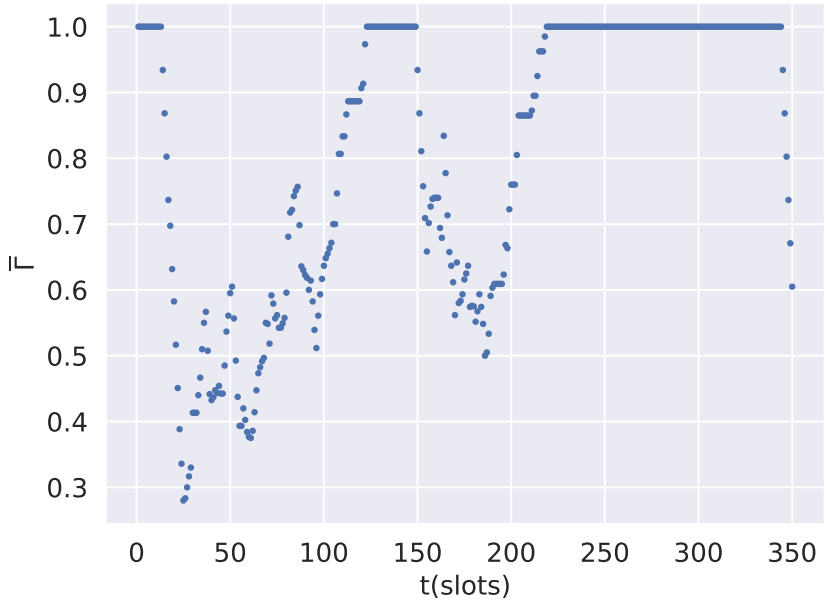


Figure 4.12: Our stabilization protocol is able to make the whole network schedule consistent for some consecutive slots.

triggers re-computation and re-dissemination of schedules based on channel conditions and the current communication schedule, i.e. the throughput variance threshold.

In Fig. 4.13 and Fig. 4.14 we show two simulations using the same base quality values, but with different throughput variance thresholds. Fig. 4.13 shows the simulation with threshold $R_1 = 0.03$, and as we can see within 500 slots we have 3 re-computations, whereas in Fig. 4.14 with threshold $R_2 = 0.15$, we only had 1 re-computation.

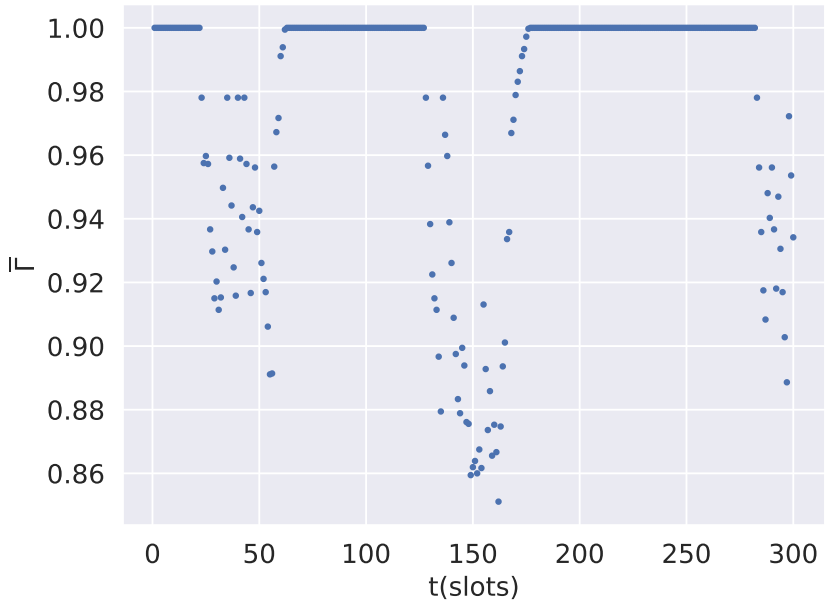


Figure 4.13: Simulation of our stabilization protocol with throughput variance threshold $R_1 = 0.03$.

As a general note, it is worth noting that $\overline{\Gamma}$ is a somewhat pessimistic metric. For instance, if we look at the network consistency ratio for the simulation in Fig. 4.13 (see Fig. 4.15), we can see that since we have $n_{slot} = 45$, it takes a long time for any nodes to use channels that are inconsistent with the hopping sequence of the master node, and most of the inconsistencies are solved by the dissemination of schedule reports before they ever become a problem. In the case of Fig. 4.16 (the same simulation of Fig. 4.14), we can see no wrong channel was used during the convergence to the new optimal communication schedule. The shown results clearly show that our stabilization protocol is able to self-repair w.r.t. network-wide schedule consistency while keeping schedule drifts small. In addition, by choosing appropriate values for the minimum time between re-computations and the throughput variance threshold, we are able to keep the network consistent for multiple slots, only re-computing and disseminating a new communication schedule upon reaching significant changes in channel quality.

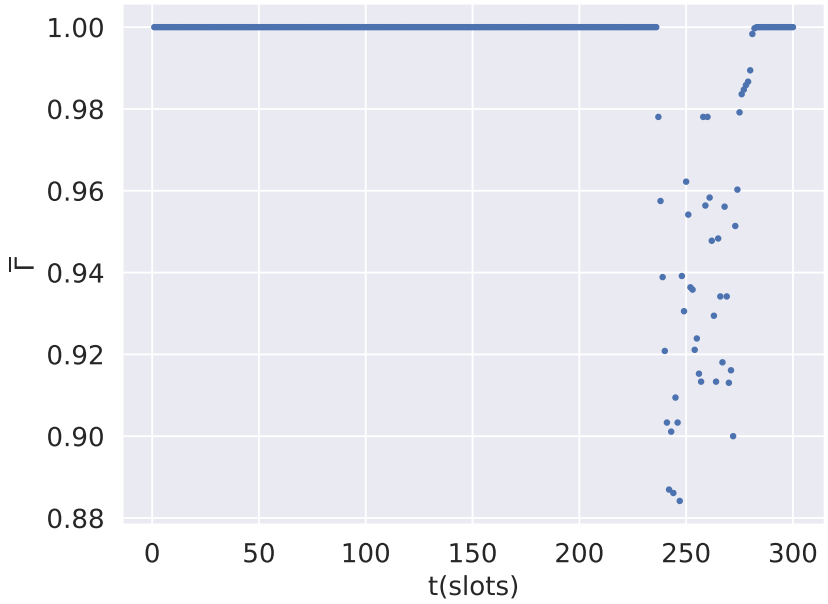


Figure 4.14: Simulation of our stabilization protocol with throughput variance threshold $R_2 = 0.15$.

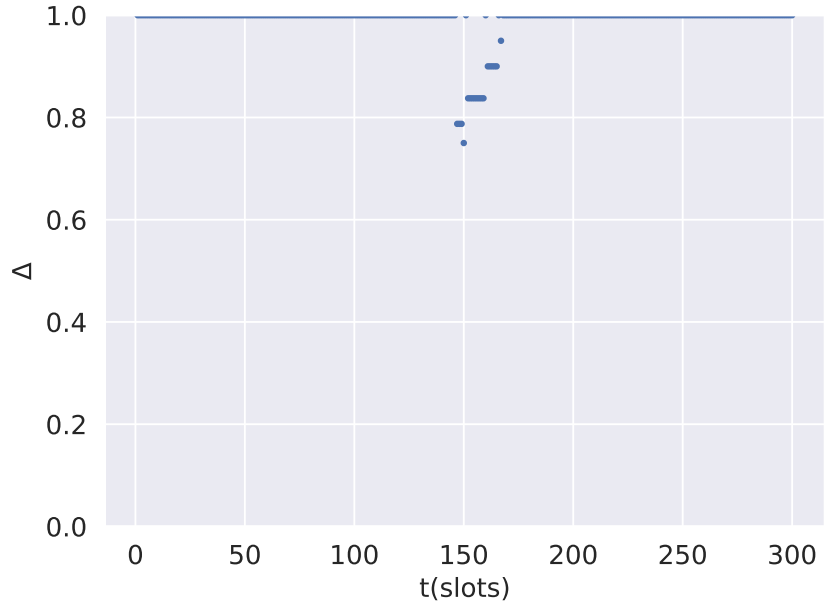


Figure 4.15: Simulation of our stabilization protocol with throughput variance threshold $R_1 = 0.03$.

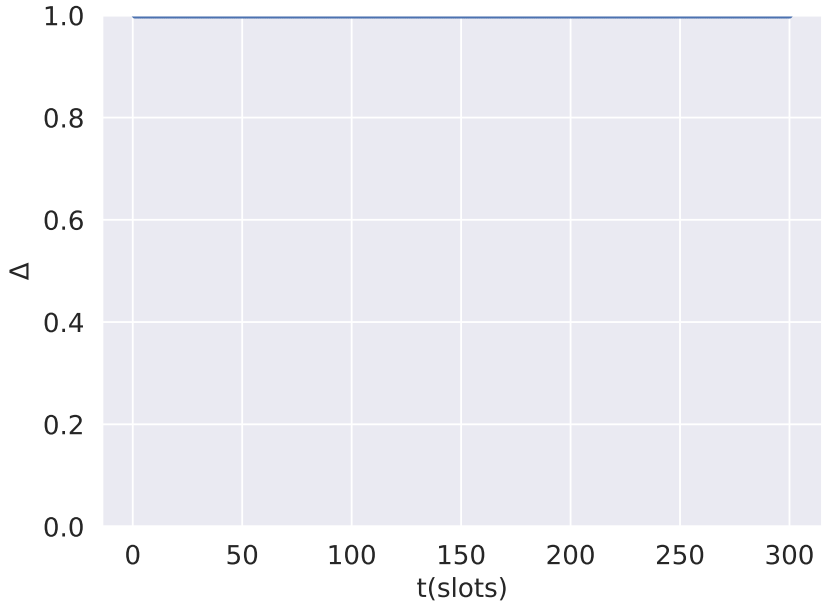


Figure 4.16: Simulation of our stabilization protocol with throughput variance threshold $R_2 = 0.15$.

Topology changes

In the next experiments, we will show that our stabilization protocol is robust against topology changes and as long as enough connectivity is still present in the network, it can restore network-wide schedule consistency, even if with different speeds of convergence.

With this in mind, we change some parameters regarding the position allocation of nodes such that the resulting topology will lead to distinct convergences to $\overline{\Gamma} = 1$ when re-computing and disseminating a new communication schedule.

In Fig. 4.20, we see the average degree of consistency and the network consistency ratio of a simulation with 50 nodes where 40 nodes are located on the bottom row and the 10 remaining nodes are positioned on a second row on top of the first one (see Fig. 4.17). The horizontal and vertical distances between nodes are both 10 meters,

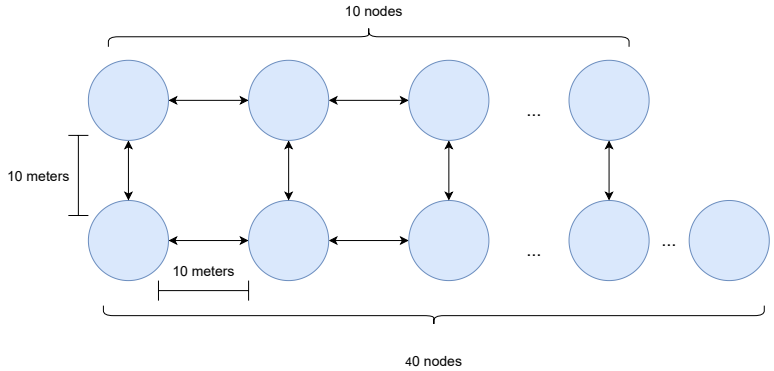


Figure 4.17: Node position allocation with 40 nodes on the bottom row and 10 nodes on the top row with 10 meters (max. range) between each node (in both the x and y direction).

the maximum transmission range is also 10 meters and the first node to the left on the bottom row is the master node. We can see that in this configuration it takes the network almost 200 slots to self-heal. In a second simulation we take the 10 nodes on the top row and position them on the bottom one yielding 50 nodes in a single line (see Fig. 4.18). The horizontal distances between the nodes and the position of the master node stay the same as in the previous simulation. As we can see in Fig. 4.19, the minimum average schedule consistency $\overline{\Gamma} \approx 0.18$ is less than half the minimum average schedule consistency in Fig. 4.20 ($\overline{\Gamma} = 0.4$) and the convergence time to $\overline{\Gamma} = 1$ in the second simulation is almost 4 times the convergence time of the first simulation. Moreover, if we look at the consistency ratio, we can see that during the second simulation for the most part of the first 400 slots, we have at least 50% or more of the nodes using the wrong channel, which is one of the possible reasons for the long convergence time, since

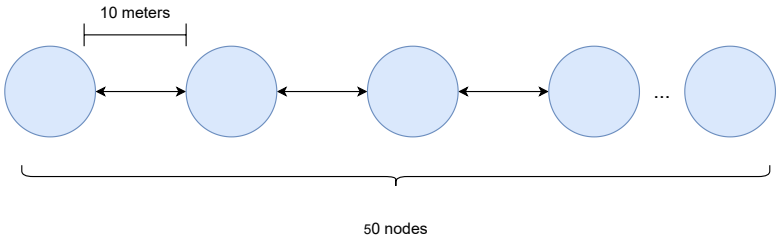


Figure 4.18: Node position allocation with 50 nodes on a single row. Nodes can only communicate with the direct neighbors to either side.

multiple nodes will listen on a channel where potentially no messages are to be received. It is worth noting, that even in such an extreme case as in Fig. 4.19, our stabilization protocol is still able to restore the network-wide schedule consistency.

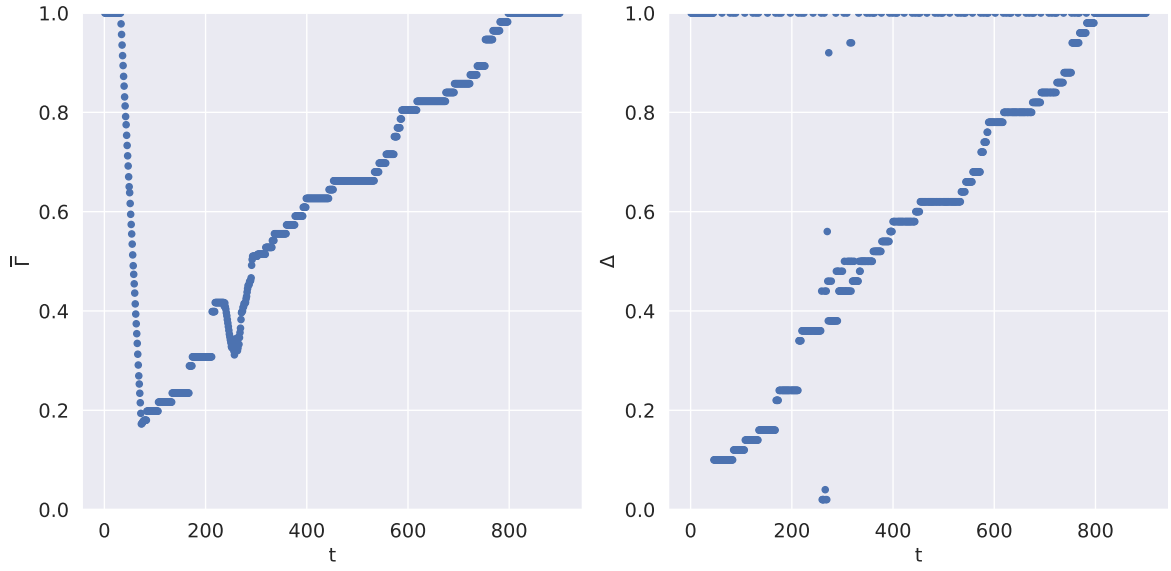


Figure 4.19: Simulation of our stabilization protocol with 50 nodes in a line with 10 meters (max. range) between each node.

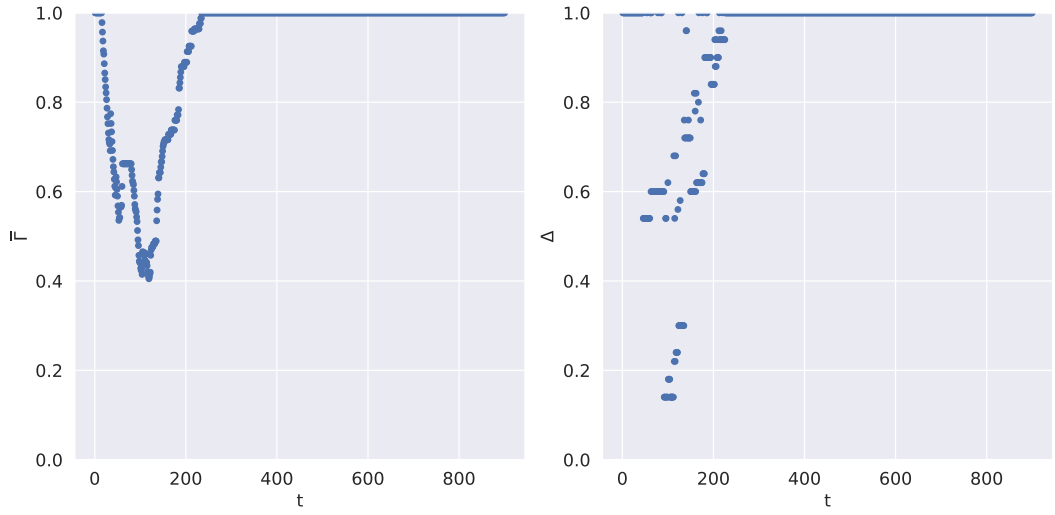


Figure 4.20: Simulation of our stabilization protocol with 40 nodes on the bottom row and 10 nodes on the top row (see Fig. 4.17)

Tick synchronization

In addition to our main experiments regarding schedule consistency, we have also performed some simulation experiments to validate our tick synchronization algorithm. For these experiments, we have used a static communication schedule with five channels, i.e. $s_{com} = \{c_3, c_1, c_2, c_5, c_3, c_1, c_4, c_5, c_3\}$ and a fixed channel quality for channel c_3, the channel picked for transmitting the tick frames, since it is the channel that has the higher channel utilization in s_{com}. Similarly to the previous experiments, 50 nodes were placed in a grid with 50 meters between nodes, both horizontally and vertically. Furthermore, the maximum transmission range for all nodes is 50 meters. This somewhat limited transmission range was chosen to construct a scenario in which the master node cannot reach many nodes with its tick frame, relying on other nodes also broadcasting tick frames to complete the tick synchronization process. Moreover, the simulation runs for 1000 slots, and each slot has a duration of 300 ms. To model the tick offsets we have used a simple clock model, in which each non-master node is assigned a random constant clock skew factor δ_{skew} (relative to the master node) such that $4 \cdot 10^{-5} \leq \delta_{skew} \leq 8 \cdot 10^{-5}$. The resulting tick offset of a node v and the master node v_{master} after one time slot is computed as:

$$d_{offset}(v, v_{master}) = \delta_{skew}(v) \cdot d_{slot}$$

As expected, we can see in Fig. 4.21 that if left unattended the average tick offset accumulates, surpassing 5 ms after only 5 minutes of operation. On the other hand, as

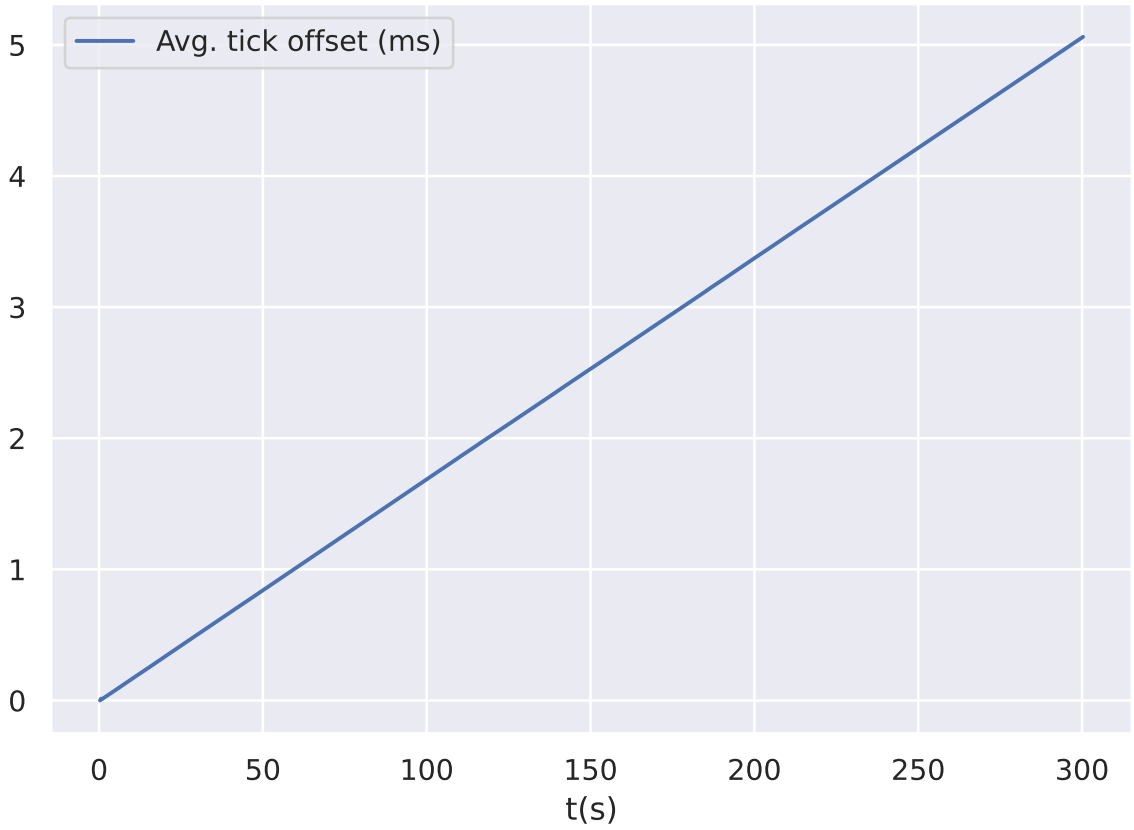


Figure 4.21: Average tick offset values with no tick synchronization.

shown in Fig. 4.22, after an initial synchronization, our approach is clearly capable of keeping the average tick offset bounded in the order of 10 us, which is negligible when compared to the duration of a time slot of 300 ms. However, it is important to note that this is only the *average* value of the tick offset, and that the *maximum* tick offset measured in our experiments was in the order of 100 to 300 us. This maximum value depends not only on the range and distribution of the skew values among the nodes but also on the topology of the network and the duration of a time slot. Furthermore, for the shown experiments we assumed good channel qualities and static communication schedules (all nodes always have the same communication schedule). It is realistic to expect slightly worse tick offset values if, for instance, the channel picked for the tick frames suddenly (or recurringly) displays a bad quality, for instance worse than 0.5, during multiple time slots.

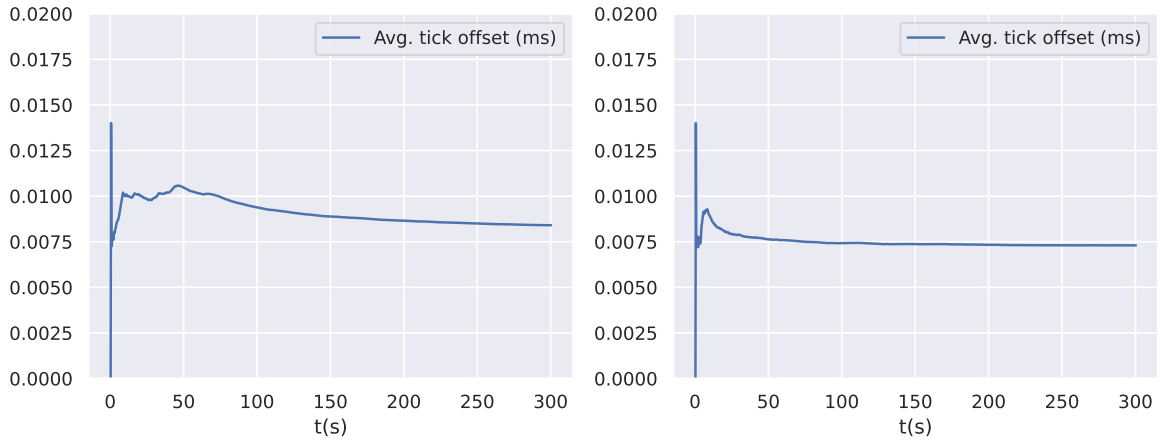


Figure 4.22: Comparison of average tick offset values where the channel used for broadcasting tick frames has channel quality 0.6 (left) and 0.8 (right).

4.6 Conflict-minimal channel orderings for communication schedules

In Chap. 3, we have introduced different heuristics to construct high-quality sensing schedules. One possible further application of these heuristics could be to construct interference-aware channel orderings for communication schedules. In this scenario, we have two networks N_1 and N_2 with nodes that communicate over a common channel following a communication schedule. Due to space constraints or independent operators, both networks are placed in proximity to each other. To maximize throughput and QoS, both networks want to communicate on channels that have minimal channel overlap. By making use of our time-slotted channel sensing approach, nodes are already capable of detecting different levels of interference and reacting to the current channel conditions by communicating on each available channel with a frequency proportional to each channel's quality. The benefits of using high-quality channels can be further improved by minimizing the conflict metric between communication schedule $N_1.s_{com}$ in use by network N_1 and communication schedule $N_2.s_{com}$ in use by N_2. This means that by using conflict-minimal hopping sequences, the networks try to preemptively lower the expected interference due to the channel selection and ordering in the adopted communication schedules, which results in throughput benefits for both networks.

Changes in channel ordering would ideally not only attempt to minimize the total overlap between two communication schedules, but also preserve the optimal re-use distances for each channel as proposed by Engel [EG18a]. However, further investigation

is needed to determine in which cases keeping re-use distances is possible and when it would be necessary to relax the condition that states that usages of the same channel should be as far apart as possible.

Given two networks N_1 and N_2, we summarize the main steps needed to create conflict-minimal channel orderings below:

1. N_1 has been active for some time and its nodes broadcast schedule reports to keep schedule-consistency.

2. After N_2 bootstraps, $N_2.v_{master}$ senses all available channels and waits for a certain time on its best local channel. It eventually receives $N_1.s_{com}$ by overhearing a schedule report and sees that $N_1.start < N_2.start$.

3. N_2 then adopts $N_1.d_{slot}$ and $N_1.n_{slot}$ for its operation if possible. If $N_1.n_{slot}$ cannot be adopted, N_2 adopts $N_2.n'_{slot} = lcm(N_1.n_{slot}, N_2.n_{slot})$, such that the communication schedules are aligned.

4. Moreover, minimizing overlap only makes sense if the networks are at least partially tick-synchronized, which is also possible to achieve by $N_2.v_{master}$ capturing sync messages broadcast by nodes in N_1.

5. $N_2.v_{master}$ computes $N_2.s_{com} = f(\vec{Q}, N_1.s_{com})$, where $\vec{Q} = \{q_1, \ldots, q_{|C|}\}$ is the set of all channel qualities measured by $N_2.v_{master}$. With this, we compute a seed communication schedule with Engel's method (solving the apportionment problem) and then use our local-search-based heuristic to create a permutation of the computed schedule in which channel overlap between both communication schedules is minimized. This last step could either preserve the re-use distances, which would yield a lower improvement in conflict metric or relax the optimality of the channel re-use distances, achieving a lower total channel overlap between the communication schedules.

By default, the network that has the latest starting time of operations adapts its communication hopping sequence to the network which has been active for a longer period. However, in a scenario where three or more networks are present (see Fig. 4.23), it is possible for a network N_2 to already have adapted its communication schedule due to a network N_3. Meanwhile network N_1 keeps its original communication schedule, since $N_1.start < N_2.start$, even though it would be beneficial for N_2 and N_1 if the latter eventually tried to minimize the channel overlap between the used communication

schedules. A possible solution to this problem is for each master node to keep track of whether the neighbor network eventually adapts its channel ordering or not, and if this change does not take place within n_{adopt} slots, $N_1.v_{master}$ decides to adapt its own channel ordering upon its next communication schedule re-computation. A thorough experimental investigation of this scenario, however, is left for future work.

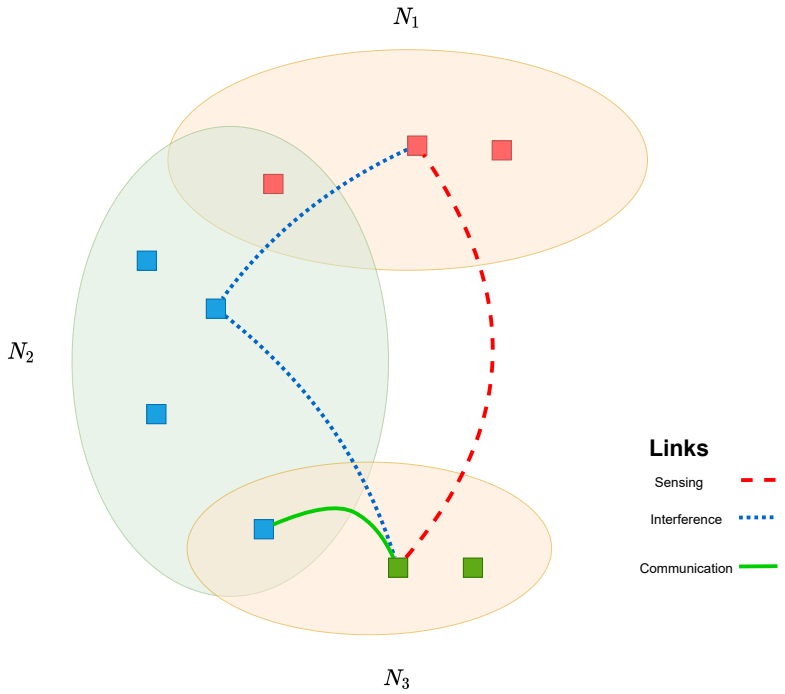


Figure 4.23: Scenario where $N_2.s_{com}$ uses an ordering based on $N_3.s_{com}$. After n_{adopt} slots, $N_1.v_{master}$ decides to optimize its channel ordering regarding interference awareness w.r.t. N_2 , even though N_1 is longer in operation than N_2.

4.7 Summary

In this chapter, we proposed and implemented a stabilization protocol for keeping nodes in an ad-hoc network synchronized and schedule-consistent w.r.t. a communication schedule, i.e. a channel hopping sequence used for communication. This stabilization protocol makes use of special messages, namely tick frames for synchronization, channel quality reports for sharing local views of channel conditions and schedule reports for disseminating the global communication hopping sequence. The communication schedules are computed based on an aggregation of channel qualities and their dissemination is triggered by significant changes in channel conditions. In addition, simulation results

were presented confirming the robustness of our stabilization protocol with respect to restoring network-wide schedule consistency in face of changes in channel conditions and topology.

To the man who only has a hammer, everything he encounters begins to look like a nail.

— Abraham Maslow

5 Developed tools

Contents

In this chapter, we briefly describe the tools developed and used for our experiments. Moreover, we introduce the architecture used for collecting experimental data and briefly describe how our code is deployed to testbed nodes.

5.1 Data logging

Running channel sensing experiments for hours produces a deluge of data and additional metadata associated with every experiment. In addition, running many of the experiments is time consuming and needs many working parts. Hence, running throw-away experiments is not a sustainable way of doing things. This means, we need a way to store our experimental data for later analysis. Initially, we used log files with comma-separated values (CSV), but this approach did not scale well. The lack of any additional structure makes organizing the data quite challenging and querying the data very ineffective. Besides, writing to files is slow, since disk i/o latency is quite high, which also led to some inconsistent data loss problems at high sampling rates. Moreover, there is the problem of data availability: each node has to copy each log file to our central server. Furthermore, this approach also makes it quite hard to combine data of different experiments, which limits our flexibility for data exploration.

Therefore, we need an architecture where data can be easily centralized in a structured manner. Furthermore, the data store should support high throughput of data ingestion. We ended up with following architecture (see Fig. 5.1):

1. Different parameters related to each experiment (metadata) are stored into a document database, and a universally unique identifier (UUID) is generated.

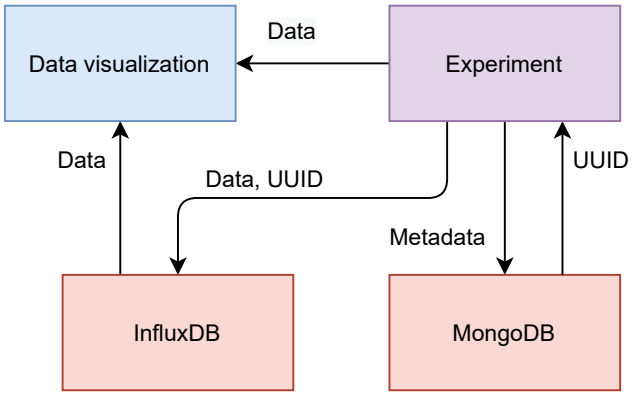


Figure 5.1: Experimental data logging and visualization architecture.

2. Experimental data, e.g. q_{cbt}, is logged at the end of every time slot to a time-series database together with its UUID to become associated with the current experiment and for later retrieval of associated metadata.

Our database of choice for storing metadata is MongoDB [mon], a general purpose, document-based database. The main advantages of using MongoDB are its ease of use and its support of continuous changes in the underlying data model. This was very important, since in a research environment, requirements change quickly. Document-based databases are basically key-value stores, and the stored data lies in versioned semi-structured documents. In the case of MongoDB, documents use the Binary Json (BSON) format. This is a variant of the classic JSON format, optimized for both data storage and faster scanning. Furthermore, MongoDB generates a UUID for each new entry by default, which, as already mentioned, we use as a foreign key in the data store, so that data and metadata are linked.

Our chosen data store is a time series database: InfluxDB [inf]. The key features that led us to choose it were:

- High write performance, optimized for time series data.
- Built-in data compression.
- Ease of deployment, since it is released as a single binary with no external dependencies.
- Posting data to the database and queries use a simple HTTP API.
- Built-in SQL-like query language and the possibility of aggregating data.
- The possiblity of adding tags to index each experiment in order to optimize queries.
- Built-in auto-expire retention policies, i.e. stale data can be automatically removed.

Both databases run on a central server reachable by all testbed nodes.

5.2 Data visualization

In order to better describe our dataset and explore relationships between different data samples, we need different ways to visualize our data. This data exploration leverages visual cues, allowing us to gain qualitative insights into the captured data. These insights can range from the presence and frequency of outliers or getting a feel for the shape of the overall data distribution, allowing us to better understand the nature of the computed metrics, and to better differentiate anomaly from normal behavior. Moreover, the integration of diverse statistics provides an additional quantitative dimension to this analysis. We have implemented two types of visualization: an offline and a real-time visualization. Our language of choice is Python due to the plethora of available data science libraries.

For our offline visualization, we ingest data over InfluxDB's HTTP API. For this type of visualization we have the whole dataset (consisting of multiple experiments) at our disposal. This allows us to process, combine and analyze data in a grander scale than by only looking at a single experiment. We can not only compute different statistics, but also produce plots to analyze the whole dataset, which would not be possible in real-time.

For the real-time visualization, data has to be forwarded from each experiment to our visualization module, which can be run on any node accessible by one of the testbed nodes. With that in mind, sensing nodes create a WebSocket [MF11] connection to the node running the visualization. The WebSocket Protocol allows bidirectional communication among sensing and visualization nodes in a TCP full-duplex connection. Data is then posted over the WebSocket connection in a JSON payload. After being processed on the Python side, data is posted to a web server and can be then visualized in any web browser of choice. Also, both sensing and visualizing nodes can be started in an asynchronous manner. For this, we have implemented a queue following a producer-consumer model. This real-time visualization is best suited for quick visual inspection, validation of the plausibility of sensed data and parameter tuning. It allows us, for instance, to see immediate effects on q_{cbt} on multiple channels by activating the traffic generator with varying traffic loads on any given channel (see Fig. 5.2) or to pick the best available channel at the moment in order to carry out specific single-channel experiments (see Fig. 5.3).

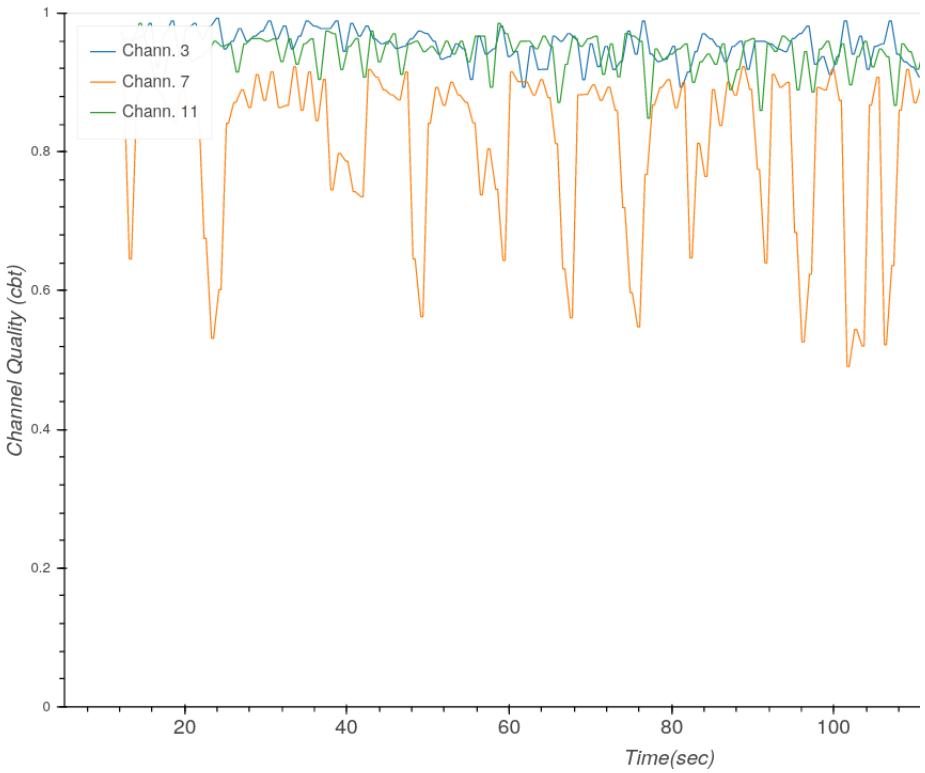


Figure 5.2: Real-time visualization as a line graph.

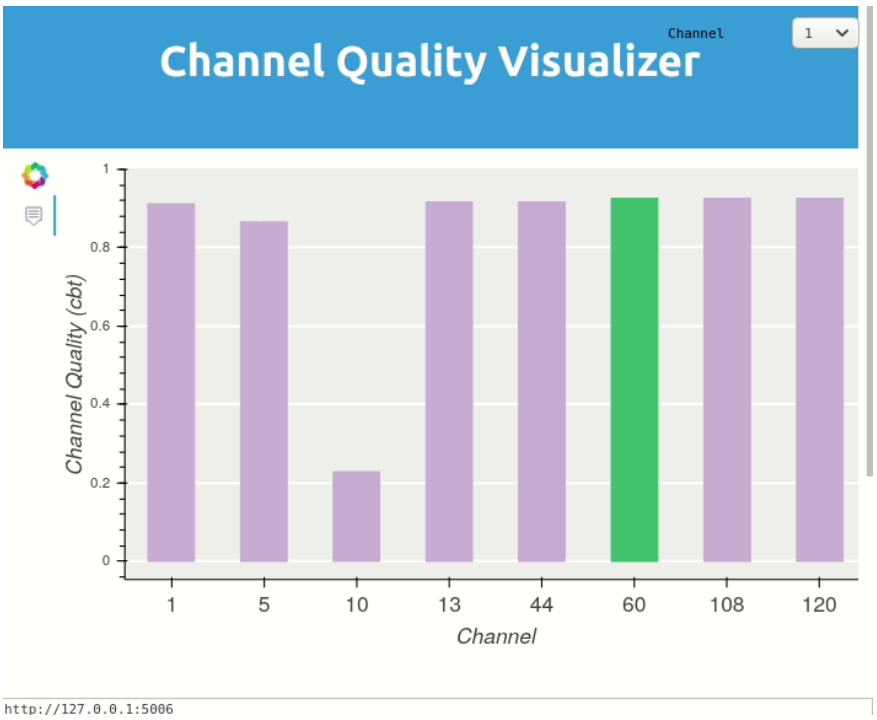


Figure 5.3: Real-time visualization in bar form, the green channel is the current best channel according to the aggregated channel quality.

5.3 Traffic generator

As mentioned in Chap. 2, we have implemented a 802.11 traffic generator, allowing us to conduct reproducible and controlled experiments.

Traffic generators of stateful and realistic wireless traffic are mostly:

- expensive
- not flexible
- bound to specific hardware platforms

To support multiple experiments related to channel sensing, we needed to create flexible and detectable realistic foreign traffic patterns in a controllable way. Depending on the effects we want to observe (say test for statistical independence in certain phenomena), it is not sufficient to transmit on a channel for defined intervals, or with different rates. In fact, foreign traffic is an overlay of traffic created by a set of multiple nodes, which if in communication or interference range will contend on the medium and accordingly transmit on the channel or defer to another node. In our approach to traffic generation, we replace this set of nodes by a single node trying to emulate the traffic that would be otherwise generated by multiple contending nodes.

5.3.1 Streams

Our solution to generate flexible traffic patterns is to combine streams with different properties. The traffic generator works in a single thread and streams generate frames that are added to a single FIFO queue. If a frame cannot be transmitted due to certain conditions (rate limits, stream is passive, etc.), no new frame is generated until these conditions change. Each node has a configurable list of generated streams. Frames are then scheduled for transmission according to the properties of each stream. In addition, conflicts between different streams are handled by postponing transmissions of deferring streams to a later slot (see Fig. 5.4). After determining the final schedule, frames are sent to the transceiver at scheduled points in time.

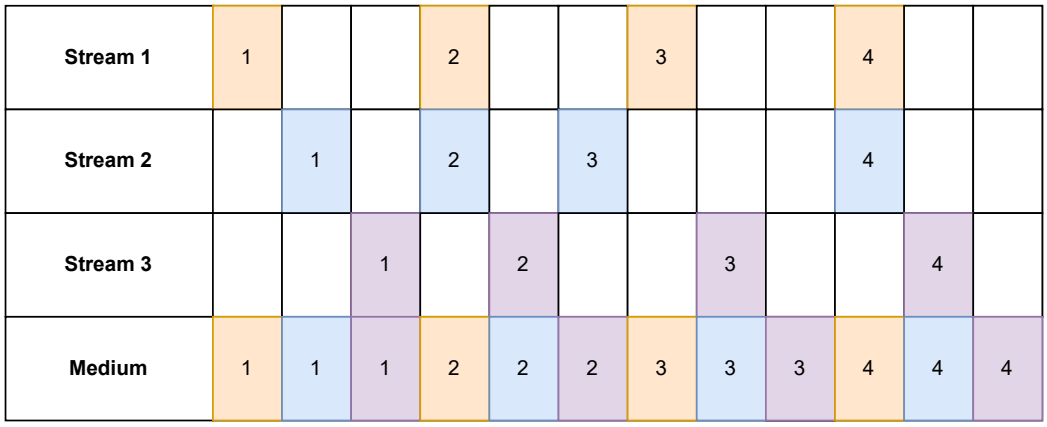


Figure 5.4: Stream 1 is periodical, streams 2 and 3 generate frames in a random manner following a certain chosen distribution. Frames are scheduled to be sent at a later point when a collision happens, e.g. the second frame of stream 2 is only set at the fifth slot due to a collision at slot 4.

Stream properties

Streams have multiple properties. We briefly describe these properties below:

- All streams start by default in `PASSIVE` mode.
- After a chosen start delay the stream becomes `ACTIVE` .
- The default start delay is 0, i.e. streams switch immediately to `ACTIVE` mode.
- Streams in `DISABLED` state never turn active. This is useful when adding different stream profiles to a configuration file and disabling some streams from time to time. This avoids having to have multiple configuration files or having to copy the streams out of the configuration file.
- If a stream is of type `PERIODIC`, it transmits frames with a fixed chosen frame rate. On the other hand, if a stream is of type `RANDOM_INTER_FRAME_SPACE`, it only transmits a frame on a given transmission opportunity with a probability of $p_{transmit}$, a configurable parameter.
- If a stream generates frames of type `FIXED`, the number of bytes in each frame's payload is constant for the whole duration of the traffic generation. Alternatively, if a stream generates frames of type `RANDOM_NORMAL`, the number of bytes in each frame is randomly sampled from a normal distribution with mean 413.0 and standard deviation 509.0 with minimal and maximal frame sizes of respectively 8 and 1200 bytes. These parameter values for the normal distribution were obtained from measurements of IP traffic patterns with 85% of it being TCP traffic [MC00].

5.3.2 Configuration and debugging

The traffic generator can be configured with help of a configuration file in json format, see Fig. 5.5.

```
{
    "duration" : 2,
    "interface": "wlan0mon",
    "channel" : 7,
    "random_seed" : 43,
    "streams": [{
        "burst_size": 1,
        "frame_length_type": "FIXED",
        "frame_type": "BEACON",
        "rate": 1,
        "stream_type": "PERIODIC",
        "start_delay" : 10
     },
     {
        "burst_size": 1,
        "frame_length_type": "RANDOM_NORMAL",
        "frame_type": "DATA",
        "max_payload_length": 1200,
        "min_payload_length": 8,
        "rate": 10,
        "stream_type": "PERIODIC",
        "start_delay" : 15
     }]
}
```

Figure 5.5: Json configuration file defining two streams.

If no configuration file is present, a number of different streams with different configuration parameters are generated by default.

In order to debug the traffic generation we can log the streams and the generated frames, see Fig. 5.6.


```
{
    "burst_size": 10,
    "frame_length_type": "FIXED",
    "frame_type": "BEACON",
    "frames": [
        {
            "curr_burst": 1,
            "length": 64,
            "timestamp": 9870.01
        },
        {
            "curr_burst": 2,
            "length": 64,
            "timestamp": 1008957.217
        }
    ],
    "rate": 1,
    "start_delay": 10,
    "stream_status": "ACTIVE",
    "stream_type": "PERIODIC"
}
```


Figure 5.6: Stream log file, showing the generation of two beacon frames with a frame rate of 1 frame per second.

5.3.3 Deployment

To rapidly adjust to changing requirements, and incremental changes to our implemented libraries and tools, as well as to avoid creating library dependency conflicts on our testbed nodes, we deployed our channel sensing code, traffic generator and visualization module as Docker [doc] images to our testbed. Docker engine is the de-facto container environment for Linux systems. A Docker image packages an application with all its dependencies and configuration to be run as a container on top of the host operating system (see Fig. 5.7). This way, containers can be seen as a lightweight virtual machines.

In order to automate and better manage the deployment of our tools to multiple testbed nodes at once, we make use of Ansible [HM17]. The main selling points of Ansible are:

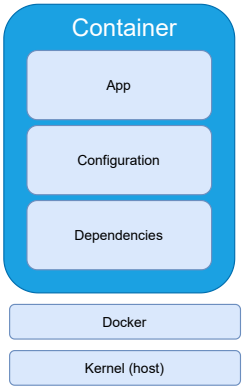


Figure 5.7: Illustration of a container runtime.

- It is easy to configure: Ansible uses YAML (which was designed to be readable) to define *executable* configurations.
- Ansible is push-based and agentless, so there is no need to install anything besides Python and SSH on controlled nodes.

With Ansible we use a *playbook* (a collection of configured tasks) to orchestrate changes in multiple testbed nodes at the same time. This could be either deploying a new version of a tool, starting it on the node, or copying over configuration files. Our overall workflow with Docker and Ansible is illustrated by Fig. 5.8.

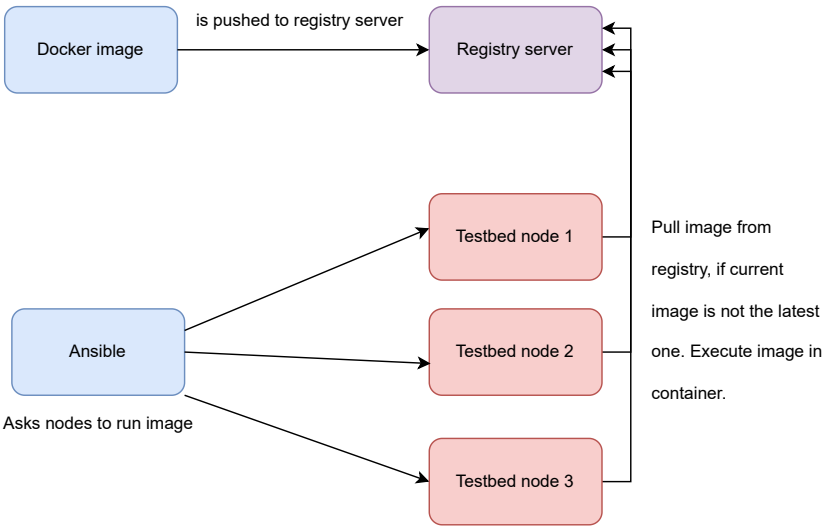


Figure 5.8: Deployment workflow with Docker and Ansible.

There is no real ending. It's just the place where you stop the story.

— Frank Herbert

6

Summary & Future Work

Contents

6.1 Summary

In this thesis, we have proposed a holistic time-slotted schedule-based approach for channel hopping and channel sensing to deal with spectrum scarcity and spectrum sharing in wireless ad-hoc networks. Even though many of the techniques introduced in this thesis are general enough and could be easily adapted for other wireless protocols, we have focused on applying them to IEEE 802.11-based ad-hoc networks. Our techniques try to minimize interference among networks while optimizing quality of service levels for an effective communication. Moreover, the main results in this thesis were evaluated through both simulations and deployment to a testbed with commodity 802.11 transceivers. In additon, whenever possible and relevant, we formalized our concepts and provided proofs or sketches thereof in order to steer towards precision and away from ambiguities. However, most of this thesis followed an empirical mindset, in which all our main assumptions were put to test and evaluated for both performance and plausibility.

In our proposed solution, nodes gather channel quality information in order to assess the state of a channel, allowing them to select high-quality channels for effective communication. First, we introduced our main channel quality metric, q_{cbt}, an energy-detection-based metric that agnostically assesses the channel with focus on foreign traffic. We then introduced steps to:

1. Correct q_{cbt} for internal traffic.

2. Aggregate the corrected metric for stability and adaptivity through a combination of an arithmetic mean and an exponential weighted moving average.

3. Update the resulting quality metric w.r.t. downward volatility levels with a penalty-based scheme.

We also evaluated the performance of q_{cbt} and confirmed that is is a robust metric that strongly correlates with achievable throughput on a chosen channel.

Moreover, we have proposed alternative channel quality metrics based on frame-derived properties such as the foreign traffic aware SNR and node degree that can be combined with our main channel quality metric for a multidimensional assessment of the channel conditions.

With an effective channel quality assessment in place, channel selection then bootstraps a proactive channel hopping scheme, in which nodes hop to different channels for communication over a common channel. The communication schedule that dictates the

adopted hopping sequence is computed based on the assessed channel qualities, as proposed by [EG18a]. This adopted paradigm still leaves one important problem unsolved: in which order should channels be sensed and how should this step be combined with communication happening on multiple channels. In Chap. 3, we have solved this problem by adopting a schedule-based sensing approach, in which two independent transceivers are used respectively for communication and channel sensing. The channel sensing order is dictated by a sensing schedule, which is locally constructed by each node based on its current communication schedule. To construct high-quality sensing schedules we had three main optimization goals in mind:

1. The sensing schedule should minimize channel sensing bias by being balanced.
2. Minimize the effects of internal traffic on q_{cbt} by aiming for conflict minimality between sensing and communication schedule.
3. Attempt to maximize overlap-fairness to better distribute the negative effects of remaining conflicts among all sensed channels.

Our final algorithm for constructing high-quality sensing schedules makes use of a stochastic iterated local-search-based heuristic and achieves remarkable near-optimal results. However, we determined that conflict minimality and overlap fairness often cannot be both fully optimized at the same time, making it essential to prioritize one over the other. In Chap. 3, we also showed how conflict-minimal sensing schedules can be constructed with help of the Hungarian Method and showed that the sensing schedules delivered by opt_conflict_ils (without any fine-tuning) fair remarkably well when compared to global optima w.r.t. conflict minimality.

In Chap. 4, we solved the remaining piece of our puzzle by introducing a master-based stabilization protocol that both disseminates global communication schedules as well as keeps the network schedule-consistent (all nodes adopt the same hopping sequence for communication) and tick-synchronized (all nodes agree on the beginning and duration of each time slot). Furthermore, a global view of channel conditions is achieved by aggregating the local channel quality assessments of all nodes in the network, forwarded to the master node through channel quality reports. Simulation results showed that the proposed protocol is robust against topology changes as well as sudden or recurring changes in channel conditions.

Finally, to evaluate our findings, we have developed and refined through years of research many different tools, of which the most relevant were briefly described in Chap. 5.

These tools deal with data collection pipelines, data visualization and wireless traffic generation as well as with the deployment of these tools to our testbed.

6.2 Future Work

As with any attempt at science, we are never completely done. There are always roads left untraveled or at least briefly visited. One of these avenues is the combination of our local search heuristics and our stabilization protocol to create conflict-minimal communication schedules, as briefly described at the end of Chap. 4. Such an approach could bring further benefits in quality of service by taking into consideration information broadcast by other networks w.r.t. their channel hopping behavior.

Another open challenge left for future work is implementing and evaluating our proposed leader election algorithm to cope with master node failures as well as devising an alternative decentralized (with no master node) mode of operation. However, such a masterless approach brings a series of additional challenges, such as:

- Much as in the centralized approach, nodes can broadcast channel quality reports to share their local views of channel quality conditions. However, these reports are not forwarded upstream since by lacking a master node, there is no upstream direction. Nodes that receive channel quality reports need to aggregate the received qualities with their own, ideally leading to nodes within a one-hop neighborhood having similar aggregated channel qualities. Nonetheless, since neighbor nodes might be in different (mutually exclusive) neighborhoods, they might end up with diverging aggregated channel qualities.

- In addition, even if nodes agree on an aggregated quality, they still have to agree on the sequence of the channel usages, i.e. the channel ordering, otherwise even though they see similar conditions they might synthesize schedules with very little matching slots. It is hard to devise a solution to this scenario without either electing a cluster head responsible for computing a communication schedule per cluster or alternatively attempting a best-effort communication in which every node computes communication schedules locally, but might have to assume that with some slots cannot be used for communication with certain probability for lack of a neighbor on the channel. Both approaches bring additional performance trade-offs, and a further investigation is left for future work.

Another interesting avenue worth exploring would be to conduct additional experiments to evaluate the robustness of our protocols in the presence of competing wireless technologies that use overlapping frequencies with 802.11 such as Zigbee and Bluetooth. Also interesting would be applying the techniques developed in this thesis to comparable technologies such as TSCH (IEEE 802.15.4-2015).

Furthermore, in addition to our channel sensing techniques, QoS requirements in wireless networks can be further supported by applying reliability-constrained routing on any given channel, i.e. creating a feasible transmission path from source node to a chosen destination such that reliability requirements are satisfied. This category of routing protocols usually rely on *active* link quality metrics and an integration of such a protocol with our *passive* channel quality metrics is left for future work.

Moreover, to detect corner cases overseen by our simulation model and to create an even stronger performance evaluation of our stabilization protocol, we may implement it on 802.11 commodity hardware and deploy it to our testbed. Finally, a deployment to a larger testbed with hundreds of mobile nodes is also worth considering.

Bibliography

[80205] Iso/iec 8802-11: 1999 [ieee std 802.11-1999(r2003)] information technology–telecommunications and information exchange between systems– local and metropolitan area networks–specific requirements–part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *ISO/IEC 8802-11 IEEE Std 802.11 Second edition 2005-08-01 ISO/IEC 8802 11:2005(E) IEEE Std 802.11i-2003 Edition*, pages 1–707, 2005.

[ACG04] Giuseppe Anastasi, Marco Conti, and Enrico Gregori. Ieee 802.11 ad hoc networks: protocols, performance and open issues. *Mobile Ad hoc networking*, pages 69–116, 2004.

[ACKR] Lakshmi Anantharamu, Bogdan S. Chlebus, Dariusz R. Kowalski, and Mariusz A. Rokicki. Deterministic broadcast on multiple access channels.

[AG20] P. Aragao and R. Gotzhein. Constructing balanced, conflict-minimal, overlap-fair channel sensing schedules. In *L. Barolli et al. (Eds.), Advanced Information Networking and Applications, Proceedings of the 34th International Conference on Advanced Information Networking and Applications (AINA 2020)*, pages 804–816. Springer, 2020.

[Agg06] Charu C. Aggarwal. On biased reservoir sampling in the presence of stream evolution. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, VLDB '06, page 607–618. VLDB Endowment, 2006.

[ALB11] Ian F. Akyildiz, Brandon F. Lo, and Ravikumar Balakrishnan. Cooperative spectrum sensing in cognitive radio networks: A survey. *Physical Communication*, 4(1):40–62, 2011.

[ARW07] M. Abusubaih, B. Rathke, and A. Wolisz. A dual distance measurement scheme for indoor ieee 802.11 wireless local area networks. In *2007 9th IFIP International Conference on Mobile Wireless Communications Networks*, pages 121–125, 2007.

[AS12] M. Akhlaq and Tarek R. Sheltami. The recursive time synchronization protocol for wireless sensor networks. In *2012 IEEE Sensors Applications Symposium Proceedings*, pages 1–6, 2012.

[Bau86] EB Baum. Iterated descent: A better algorithm for local search in combinatorial optimization problems. *Manuscript*, 1986.

[BCM+18] Flavia Bonomo, Maria Chudnovsky, Peter Maceli, Oliver Schaudt, Maya Stein, and Mingxian Zhong. Three-coloring and list three-coloring of graphs without induced paths on seven vertices. *Combinatorica*, 38(4):779–801, Aug 2018.

[Bil13] Ana Bildea. *Link Quality in Wireless Sensor Networks*. PhD thesis, Université de Grenoble, 2013.

[Box76] George E. P. Box. Science and statistics. *Journal of the American Statistical Association*, 71(356):791–799, 1976.

[BRA20] SPECTRUM BAND IN BRAZIL. Assessing the economic value of unlicensed use of the 6 ghz spectrum band in brazil. 2020.

[Bun21] Bundesnetzagentur. Allgemeinzuteilung von Frequenzen im Bereich 5945 MHz - 6425 MHz für drahtlose Zugangssysteme, einschließlich lokaler Funknetze WAS/WLAN („Wireless Access Systems including Wireless Local Area Networks“). `https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/Telekommunikation/Unternehmen_Institutionen/Frequenzen/Allgemeinzuteilungen/MobilfunkDectWlanCBFunk/vfg552021WLAN6GHz.pdf?__blob=publicationFile&v=3`, 2021. [Online; accessed 4-November-2021].

[BY75] M. L. Balinski and H. P. Young. The quota method of apportionment. *The American Mathematical Monthly*, 82(7):701–730, 1975.

[CD15] Artur Czumaj and Peter Davies. Optimal leader election in multi-hop radio networks. *ArXiv e-prints*, 2015.

[Che97] Joseph Cheriyan. Randomized $\tilde{O}(m(|v|))$ algorithms for problems in matching theory. *SIAM J. COMPUTING*, 26:1635–1655, 1997.

[CHT04] Adam Chlipala, Jonathan Hui, and Gilman Tolle. Deluge: data dissemination for network reprogramming at scale. *University of California, Berkeley, Tech. Rep*, 2004.

[CRS06] Kameswari Chebrolu, Bhaskaran Raman, and Sayandeep Sen. Long-distance 802.11b links: Performance measurements and experience. In *Proceedings of the 12th Annual International Conference on Mobile Computing and Networking*, MobiCom '06, page 74–85, New York, NY, USA, 2006. Association for Computing Machinery.

[CSZ+15] Wu Chen, Jianhua Sun, Lu Zhang, Xiang Liu, and Liang Hong. An implementation of ieee 1588 protocol for ieee 802.11 wlan. *Wireless networks*, 21(6):2069–2085, 2015.

[CZ11] Ho Ting Cheng and Weihua Zhuang. Simple channel sensing order in cognitive radio networks. *IEEE Journal on Selected Areas in Communications*, 29(4):676–688, 2011.

[DANLW15] Simon Duquennoy, Beshr Al Nahas, Olaf Landsiedel, and Thomas Watteyne. Orchestra: Robust mesh networks through autonomously scheduled tsch. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, SenSys '15, page 337–350, New York, NY, USA, 2015. Association for Computing Machinery.

[DBFP09] Thanh Dang, Nirupama Bulusu, Wu-chi Feng, and Seungweon Park. Dhv: A code consistency maintenance protocol for multi-hop wireless sensor networks. In Utz Roedig and Cormac J. Sreenan, editors, *Wireless Sensor Networks*, pages 327–342, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[DKS10] P. Dely, Andreas Kassler, and Dmitry Sivchenko. Theoretical and experimental analysis of the channel busy fraction in ieee 802.11. In *Future Network and Mobile Summit 2010*, pages 1 – 9, 07 2010.

[doc] Docker. `https://www.docker.com/`. Accessed: 2022-02-15.

[EG18a] Markus Engel and Reinhard Gotzhein. Dynamic computation and adjustment of channel hopping sequences for cognitive radio networks based on quality metrics. In *Proceedings of the 2018 International Conference on Embedded Wireless Systems and Networks*, EWSN ’18, pages 79–89, USA, 2018. Junction Publishing.

[EG18b] Markus Engel and Reinhard Gotzhein. Dynamic computation and adjustment of channel hopping sequences for cognitive radio networks based on quality metrics. In *Proceedings of the 2018 International Conference on Embedded Wireless Systems and Networks*, EWSN '18, page 79–89, USA, 2018. Junction Publishing.

[EGE03] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Oper. Syst. Rev.*, 36(SI):147–163, December 2003.

[Eng20] Markus Engel. *Robust Channel Hopping Sequences in Cognitive Radio Networks*. PhD thesis, TU Kaiserslautern, 2020.

[FVR07] Paul Fuxjaeger, Danilo Valerio, and Fabio Ricciato. The myth of non-overlapping channels: interference measurements in ieee 802.11. 01 2007.

[Gas12] Matthew Gast. *802.11n: A Survival Guide*. O'Reilly Media, Inc., 2012.

[GcC05] Fanglu Guo and Tzi cker Chiueh. Sequence number-based mac address spoof detection. In *in Proceedings of 8th International Symposium on Recent Advances in Intrusion Detection (RAID*. Springer, 2005.

[GH12] Mohsen Ghaffari and Bernhard Haeupler. Near optimal leader election in multi-hop radio networks. *CoRR*, abs/1210.8439, 2012.

[GHHP13] Philippe Galinier, Jean-Philippe Hamiez, Jin-Kao Hao, and Daniel Porumbel. *Recent Advances in Graph Vertex Coloring*, pages 505–528. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[GK11] R. Gotzhein and T. Kuhn. Black burst synchronization (bbs) – a protocol for deterministic tick and time synchronization in wireless networks. *Computer Networks*, 55(13):3015–3031, 2011.

[GKS03] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, page 138–149, New York, NY, USA, 2003. Association for Computing Machinery.

[Gol05] Andrea Goldsmith. *Wireless communications*. Cambridge university press, 2005.

[Gra16] Ali Grami. Chapter 12 - wireless communications. In Ali Grami, editor, *Introduction to Digital Communications*, pages 493–527. Academic Press, Boston, 2016.

[HH09] Simon Hay and Robert Harle. Bluetooth tracking without discoverability. In Tanzeem Choudhury, Aaron Quigley, Thomas Strang, and Koji Sugiinuma, editors, *Location and Context Awareness*, pages 120–137, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[HHSW10] Daniel Halperin, Wenjun Hu, Anmol Sheth, and David Wetherall. Predictable 802.11 packet delivery from wireless channel measurements. *SIGCOMM Comput. Commun. Rev.*, 40(4):159–170, August 2010.

[HM17] Lorin Hochstein and Rene Moser. *Ansible: Up and Running: Automating configuration management and deployment the easy way.* " O'Reilly Media, Inc.", 2017.

[HMCV07] Hao Hu, Steven A. Myers, Vittoria Colizza, and Alessandro Vespignani. Wifi epidemiology: Can your neighbors' router make yours sick? *CoRR*, abs/0706.3146, 2007.

[HS05] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search, Foundations and Applications*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann, San Francisco, 2005.

[HXY05] Hongqiang Zhai, Xiang Chen, and Yuguang Fang. How well can the ieee 802.11 wireless lan support quality of service? *IEEE Transactions on Wireless Communications*, 4(6):3084–3094, 2005.

[inf] InfluxDB. `https://www.influxdata.com/`. Accessed: 2022-02-15.

[IvSV06] Konrad Iwanicki, Maarten van Steen, and Spyros Voulgaris. Gossip-based clock synchronization for large decentralized systems. In Alexander Keller and Jean-Philippe Martin-Flatin, editors, *Self-Managed Networks, Systems, and Services*, pages 28–42, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[JCH98] R. Jain, D. Chiu, and W. Hawe. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. *arXiv e-prints*, page cs/9809099, Sep 1998.

[JEG18] Paulo Fernando Aragao Alves Junior, Markus Engel, and Reinhard Gotzhein. A three-dimensional stabilization protocol for time-slotted multi-hop cognitive radio networks with channel hopping. In *32nd IEEE International Conference on Advanced Information Networking and Applications, AINA 2018, Krakow, Poland, May 16-18, 2018*, pages 32–39, 2018.

[JM08] David S. Johnson and Lyle A. McGeoch. The traveling salesman problem: A case study in local optimization. 2008.

[Joh90] David S Johnson. Local optimization and the traveling salesman problem. In *International colloquium on automata, languages, and programming*, pages 446–461. Springer, 1990.

[KLDLa13] Zaheer Khan, Janne J. Lehtomäki, Luiz A. DaSilva, and Matti Latva-aho. Autonomous sensing order selection strategies exploiting channel access information. *IEEE Transactions on Mobile Computing*, 12(2):274–288, 2013.

[Kru56] J. B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. In *Proceedings of the American Mathematical Society, 7*, 1956.

[Kuh55] Harold W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly*, 2(1–2):83–97, March 1955.

[Lam05] Leslie Lamport. Generalized consensus and paxos. 2005.

[LL08] Kaisen Lin and Philip Levis. Data discovery and dissemination with dip. In *2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)*, pages 433–444, 2008.

[LLG+17] Liwang Li, Tong Li, Jincheng Ge, Lijun Kong, and Jie Liu. Channel sensing order for distributed cognitive networks with multi-user and multi-channel. In *2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN)*, pages 44–50, 2017.

[LMP+05] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. *TinyOS: An Operating System for Sensor Networks*, pages 115–148. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[LMS10] Helena Lourenço, Olivier Martin, and Thomas Stützle. *Iterated Local Search: Framework and Applications*, volume 146, pages 363–397. 09 2010.

[LMS19] Helena Ramalhinho Lourenço, Olivier C. Martin, and Thomas Stützle. *Iterated Local Search: Framework and Applications*, pages 129–168. Springer International Publishing, Cham, 2019.

[LP09] László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

[LPCS04] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *In NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*. USENIX Association, 2004.

[LSW+20] Olof Liberg, Mårten Sundberg, Y.-P. Eric Wang, Johan Bergman, Joachim Sachs, and Gustav Wikström. Chapter 15 - multefire alliance iot technologies. In Olof Liberg, Mårten Sundberg, Y.-P. Eric Wang, Johan Bergman, Joachim Sachs, and Gustav Wikström, editors, *Cellular Internet of Things (Second Edition)*, pages 633–685. Academic Press, second edition edition, 2020.

[Lá79] Lovász László. On determinants, matchings and random algorithms. volume 79, pages 565–574, 01 1979.

[Mar59] Harry M. Markowitz. *Portfolio Selection: Efficient Diversification of Investments*. Yale University Press, 1959.

[MC00] Shannon McCreary and K. Claffy. Trends in wide area ip traffic patterns - a view from ames internet exchange. *Proceedings of 13th ITC Specialist Seminar on Internet Traffic Measurement and Modeling, Monterey, CA*, 01 2000.

[MF11] Alexey Melnikov and Ian Fette. The WebSocket Protocol. RFC 6455, December 2011.

[MKSL04] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, SenSys '04,

page 39–49, New York, NY, USA, 2004. Association for Computing Machinery.

[MO96] Olivier C Martin and Steve W Otto. Combining simulated annealing with local search heuristics. *Annals of operations research*, 63(1):57–75, 1996.

[MOF91] Olivier Martin, Steve W Otto, and Edward W Felten. *Large-step Markov chains for the traveling salesman problem*. Citeseer, 1991.

[mon] MongoDB. `https://www.mongodb.com/`. Accessed: 2022-02-15.

[MSBA06] Arunesh Mishra, Vivek Shrivastava, Suman Banerjee, and William Arbaugh. Partially overlapped channels not considered harmful. In *Partially overlapped channels not considered harmful*, volume 34, pages 63–74, 06 2006.

[Mun57] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[OO14] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, pages 305–319, 2014.

[Ope17] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

[Par80] Michael Parkinson. The extreme value method for estimating the variance of the rate of return. *The Journal of Business*, 53(1):61–65, 1980.

[RKC21] Juan Jung Raúl Katz and Fernando Callorda. The Economic Value of Wi-Fi: a global view (2021 – 2025, 2021.

[RPD+05] Joshua Robinson, Konstantina Papagiannaki, Christophe Diot, Xingang Guo, and Lakshman Krishnamurthy. Experimenting with a multi-radio mesh networking testbed. 04 2005.

[RVKF13] Mubashir Husain Rehmani, Aline Carneiro Viana, Hicham Khalife, and Serge Fdida. Surf: A distributed channel selection strategy for data dissemination in multi-hop cognitive radio networks. *Computer Communications*, 36(10):1172–1185, 2013.

[SCT+16] Elena Saltikoff, John Y. N. Cho, Philippe Tristant, Asko Huuskonen, Lynn Allmon, Russell Cook, Erik Becker, and Paul Joe. The threat to weather radars by wireless technology. *Bulletin of the American Meteorological Society*, 97(7):1159 – 1167, 2016.

[SGSK07] T. Salonidis, M. Garetto, A. Saha, and E. Knightly. Identifying high throughput paths in 802.11 mesh networks: a model-based approach. In *2007 IEEE International Conference on Network Protocols*, pages 21–30, 2007.

[SHFS05] R. Simon, Leijun Huang, E. Farrugia, and S. Setia. Using multiple communication channels for efficient data dissemination in wireless sensor networks. In *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, 2005.*, pages 10 pp.–439, 2005.

[Sta22] Statista. Global smartphone penetration rate as share of population from 2016 to 2020, 2022.

[STC+08] Yong Sheng, Keren Tan, Guanling Chen, David Kotz, and Andrew Campbell. Detecting 802.11 mac layer spoofing using received signal strength. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1768–1776. IEEE, April 2008.

[SZQ+19] Ke Shi, Lin Zhang, Zhiying Qi, Kang Tong, and Hongsheng Chen. Transmission scheduling of periodic real-time traffic in ieee 802.15. 4e tsch-based industrial mesh networks. *Wireless Communications and Mobile Computing*, 2019, 2019.

[TA20] Francisco Tirado Andrés. *Methodology for implementation of synchronization strategies for wireless sensor networks.* PhD thesis, Telecomunicacion, 2020.

[Tol05] Gilman Tolle. Design of an application-cooperative management system for wireless sensor networks. pages 121–132, 2005.

[tsc12] Ieee standard for local and metropolitan area networks–part 15.4: Low-rate wireless personal area networks (lr-wpans) amendment 1: Mac sublayer. *IEEE Std 802.15.4e-2012 (Amendment to IEEE Std 802.15.4-2011)*, pages

1–225, 2012.

[Tut47] W. T. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, s1-22(2):107–111, 1947.

[Var12] Pramod K Varshney. *Distributed detection and data fusion*. Springer Science & Business Media, 2012.

[VCP+20] Marcos A. M. Vieira, Matheus S. Castanho, Racyus D. G. Pacífico, Elerson R. S. Santos, Eduardo P. M. Câmara Júnior, and Luiz F. M. Vieira. Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications. *ACM Comput. Surv.*, 53(1), February 2020.

[VJP08] Frank Visser, Gerard J. M. Janssen, and Przemyslaw Pawelczak. Multinode spectrum sensing based on energy detection for dynamic spectrum access. In *VTC Spring 2008 - IEEE Vehicular Technology Conference*, pages 1394–1398, 2008.

[Wal99] Joachim Paul Walser. *Integer Optimization by Local Search, A Domain-Independent Approach*. Springer, Germany, 1999.

[WD21] Jianjun Wen and Waltenegus Dargie. Characterization of link quality fluctuation in mobile wireless sensor networks. *ACM Transactions on Cyber-Physical Systems*, 01 2021.

[WiG] WiGLE.net. Wireless network mapping.

[WPG15] Thomas Watteyne, Maria-Rita Palattella, and Luigi Alfredo Grieco. Using ieee 802.15. 4e time-slotted channel hopping (tsch) in the internet of things (iot): Problem statement. *Internet Engineering Task Force*, 2015.

[WTB+12] Tim Winter, Pascal Thubert, Anders Brandt, Jonathan W Hui, Richard Kelsey, Philip Levis, Kris Pister, Rene Struik, Jean-Philippe Vasseur, Roger K Alexander, et al. Rpl: Ipv6 routing protocol for low-power and lossy networks. *rfc*, 6550:1–157, 2012.

[XLL15] Yaxiong Xie, Zhenjiang Li, and Mo Li. Precise power delay profiling with commodity wifi. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, MobiCom '15, page 53–64, New York, NY, USA, 2015. ACM.

[XWW+15] Yuhua Xu, Qihui Wu, Jinlong Wang, Liang Shen, and Alagan Anpalagan. Robust multiuser sequential channel sensing and access in dynamic cognitive radio networks: Potential games and stochastic learning. *IEEE Transactions on Vehicular Technology*, 64(8):3594–3607, 2015.

Curriculum Vitae

Contact details

Paulo Fernando Aragao Alves Junior

pauloaragao1952@gmail.com

Education

PhD in Computer Science, TUK, Kaiserslautern, Germany | 2017 - present

M.Sc. in Computer Science, TUK, Kaiserslautern, Germany | 2015 - 2017

Bachelor in Computer Engineering, UFPE, Recife, Brazil with honours (Láurea Universitária) | 2009 - 2015

Exchange program, TUK, Kaiserslautern, Germany | 2013 - 2014

Work experience

Research assistant, TUK | 2017 - 2022

Student research assistant, TUK | 2016 - 2017

Student research assistant, UFPE | 2012

Teaching assistant in Computer Networking, UFPE | 2011

Teaching assistant in Operating Systems, UFPE | 2010 - 2011

Teaching assistant in Physics, UFPE | 2010

Teaching assistant in Discrete Mathematics, UFPE | 2009

Publications

Paulo Fernando Aragao Alves Junior, Markus Engel, and Reinhard Gotzhein. **A Three-Dimensional Stabilization Protocol for Time-Slotted Multi-hop Cognitive Radio Networks with Channel Hopping.** *In: 32nd IEEE International Conference on Advanced Information Networking and Applications, AINA 2018, Krakow, Poland, May 16-18, 2018. Ed. by Leonard Barolli et al. IEEE Computer Society, 2018*, pp. 32–39. doi: 10.1109/AINA.2018. 00018.

Paulo Aragao and Reinhard Gotzhein. **Constructing Balanced, Conflict-Minimal, Overlap-Fair Channel Sensing Schedules.** *In: Advanced Information Networking and Applications. Ed. by Leonard Barolli et al. Cham: Springer International Publishing, 2020*, pp. 804–816. isbn: 978-3-030-44041-1.

Paulo Fernando Aragao Alves, Reinhard Gotzhein, and Lucas Sonntag Hagen. **Volatility-Aware Channel Sensing with Commodity 802.11 Hardware.** *In: 2021 IEEE Global Communications Conference (GLOBECOM). 2021,* pp. 1–7. doi: 10.1109/GLOBECOM46510.2021.9685841.