Universidade Federal Fluminense

Diego Passos

Flow-Based Interference-Aware Routing in Multihop Wireless Networks

NITERÓI

2013

Universidade Federal Fluminense

Diego Passos

Flow-Based Interference-Aware Routing in Multihop Wireless Networks

Thesis presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Doctor of Science.

Advisor:
Prof. Célio Albuquerque, Ph.D.

NITERÓI

2013

Flow-Based Interference-Aware Routing in Multihop Wireless Networks

Diego Passos

Thesis presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Doctor of Science.

Aproved by:

Prof. Célio Albuquerque, Ph.D. / IC-UFF (Advisor)

Prof. Artur Ziviani, Ph.D. / LNCC

Profa. Débora Christina Muchaluat Saade, D.Sc. / IC-UFF

Prof. José Ferreira de Rezende, Ph.D / COPPE-UFRJ

Profa. Lúcia Maria de Assumpção Drummond, D.Sc. / IC-UFF

Niterói, September 19th 2013.

I firmly believe that any man's finest hour, the greatest fulfillment of all that he holds dear, is that moment when he has worked his heart out in a good cause and lies exhausted on the field of battle – victorious.

(Vince Lombardi)

To my parents, Rubem and Rita, and my wife, Fernanda.

Acknowledgments

This thesis would not be possible without the assistance and support of a number of individuals and institutions. Although there are far too many to cite individually, I believe some special thanks are due.

First and foremost, I would like to thank my family. My father, Rubem, is my role model, my primary reference for honesty, hard work, and dedication. He is possibly the proudest person for this work, and rightfully so: he taught me to persevere and thrive during difficult times, an essential skill during the past four years. My mother, Rita, is the sweetest person I know: someone who is always available to provide comforting words, to help, and to take care. Together and with a lot of effort, they provided me with everything I needed to pursue my dreams: financial support, education, freedom, and love. I also would like to thank my sister, Thais, whom I could not visit during all these years. And finally my wife, Fernanda, who always supports me, even in my worst days.

I would like to thank all the professors at IC-UFF for sharing their vast knowledge, not only through their classes, but also on a daily basis. Special thanks are due to Professor Célio Albuquerque, who has been my advisor for the past eight years and whom I look up to as a researcher and a lecturer.

I also would like to thank the members of Laboratório MidiaCom, students, employees, and professors, for the support, the infra-structure, the companionship, and the advices. Specially, I would like to thank Marister Outão, Joacir Silva, and Helga Balbi, for always being willing to help without asking for anything in return. I would like to thank Clayton Reis, a good friend who could always make me laugh, even during the most stressful days. Finally, I would like to thank Professors Ricardo Carrano and Juliano Kazienko, with whom I could share this experience of being a graduate student. They both provided me with uncountable hours of technical and philosophical input for this thesis, as well as for the academic life in general.

Finally, I would like to acknowledge CAPES for the scholarship that provided the financial support without which this thesis would have been impossible.

Resumo

Redes sem fio de múltiplos saltos são empregadas em vários tipos de cenários e ambientes. Elas são utilizadas em redes móveis, redes de acesso em locais sem infra-estrutura prévia e redes de sensores. Um dos mecanismos de maior importância neste tipo de rede é o roteamento, devido à possível existência de múltiplos caminhos entre quaisquer dois nós da rede. No entanto, diferentemente do que ocorre nas redes cabeadas, a qualidade de um caminho não é dependente apenas das qualidades individuais dos enlaces que o compõem e do estado das filas dos nós intermediários. Dada a natureza de difusão do meio sem fio, o desempenho de um caminho pode ser afetado pelo uso de outros enlaces, o que introduz mais complexidade ao problema de encontrar rotas ótimas. Este fenômeno, no qual o desempenho de um enlace ou caminho é afetado pelo uso simultâneo de outros enlaces, é conhecido como auto-interferência.

Uma técnica já estudada na literatura que tem o potencial de mitigar os efeitos da auto-interferência é a codificação de rede. Através desta técnica, nós da rede podem combinar pacotes distintos em uma única mensagem de tamanho similar, reduzindo o número de transmissões necessárias para uma dada comunicação.

O objetivo desta tese é estudar o problema da auto-interferência do ponto de vista da seleção de rotas em redes sem fio de múltiplos saltos. Especificamente, esta tese introduz um novo método de seleção de rotas que leva em consideração a auto-interferência entre os fluxos de rede, bem como a possibilidade de utilizar codificação de rede para mitigar este problema. Diferentemente de outras propostas encontradas na literatura, este método utiliza uma abordagem nova que emprega uma visão global dos fluxos de rede, em oposição aos algoritmos de seleção de rotas tradicionais que analisam fluxos separadamente.

O método proposto foi implementado tanto em um ambiente simulado, quanto em ambiente real. Experimentos conduzidos no ambiente simulado demonstram a capacidade do mecanismo de encontrar conjuntos de rotas alternativos que resultam em ganhos de até 90% em termos de vazão agregada. Por outro lado, os testes no ambiente real mostram a viabilidade da proposta na prática, mesmo quando implementada em equipamentos de prateleira com severas restrições de processamento e armazenamento.

Palavras-Chave

1. Roteamento.
2. Redes Sem Fio de Múltiplos Saltos.
3. Interferência.
4. Codificação de Rede.

Abstract

Multihop wireless networks are employed in a wide range of scenarios and environments. They are used for mobile networks, ad hoc wireless access networks, and sensor networks. Routing is a core function in this kind of network due to the possible existence of multiple paths between any pair of nodes. Differently from wired networks, though, the quality of a path is not determined only by the individual links' qualities and the state of intermediate nodes' buffers. Given the shared nature of the wireless medium, the performance of a path may be affected by the usage of other links, further complicating the problem of finding the optimal routes. This phenomenon, in which the performance of a link or path is affected by the simultaneous usage of other links, is known as self interference.

A technique that has already been studied in the literature and has the potential to mitigate self interference is network coding. Through the employment of this technique, network nodes can combine distinct packets into a single message of similar size, reducing the number of transmissions required to establish a certain communication.

The goal of this thesis is to study the self interference problem from the standpoint of route selection in multihop wireless networks. Specifically, this thesis introduces a novel method for selecting routes that takes into account the self interference between network flows, as well as the possibility of employing network coding to mitigate this issue. Differently from other proposals found in the literature, this method employs an approach that uses a global view of the network flows, in contrast with the traditional route selection algorithms that analyze each flow separately.

The proposed method was implemented in both simulated and real environments. Experiments on simulated environments demonstrate the ability of our mechanism to finding alternative path sets that result in gains up to 90% in terms of aggregated throughput. On the other hand, tests on the real environment show that the method is viable in practice, even when implemented on off-the-shelf equipment with severe processing and memory constraints.

Keywords

1. Routing.
2. Multihop Wireless Networks.
3. Interference.
4. Network Coding.

Acronyms

ACK	:	Acknoledgement
AL	:	Activity Level
AODV	:	Adhoc On-demand Distance Vector
ARQ	:	Automatic Repeat Request
BSSID	:	Basic Service Set Identifier
CBR	:	Constant Bit Rate
CPTT	:	Coding-aware Path Transmission Time
CRM	:	Coding-aware Routing Metric
CSMA/CA	:	Carrier Sense Multiple Access with Collision Avoidance
CTS	:	Clear To Send
DCAR	:	Distributed Coding-Aware Routing
DSDV	:	Destination Sequenced Distance Vector
DSR	:	Dynamic Source Routing
ETX	:	Expected Transmission Count
EWMA	:	Exponentially Weighted Moving Average
HLCR	:	Heuristic Load-balanced Coding-aware Routing
HOP	:	Hop Count
IAR	:	Interference-Aware Routing
ICAR	:	Interference and Coding-Aware Routing
IEEE	:	Institute of Electrical and Electronics Engineers
IRU	:	Interference-aware Resource Usage
MAC	:	Medium Access Control
MARA	:	Metric Aware Rate Adaptation
MARIA	:	Mesh Admission control and qos Routing with Interference Awareness
ML	:	Minimum Loss
MPR	:	MultiPoint Relay
MTU	:	Maximum Transmission Unit
ns-2	:	Network Simulator 2
OLSR	:	Optimized Link State Routing

QoS	:	Quality of Service
RREP	:	Route Reply
RREQ	:	Route Request
RTS	:	Request To Send
SINR	:	Signal to Interference plus Noise Ratio
SNR	:	Signal to Noise Ratio
SLSP	:	Simple Link State Protocol
TCP	:	Transmission Control Protocol
UDP	:	User Datagram Protocol
XOR	:	Exclusive Bitwise OR
WCETT	:	Weighted Cumulative Expected Transmission Time
WETCC	:	Weighted Expected Transmission Count with Coding

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Multihop wireless networks have motivated research interest for at least 37 years [33]. Due to their capacity of covering large areas and the possibility of being deployed in regions without much previous infrastructure, those networks are employed by a number of different applications. This flexibility resulted in the creation of many specialized variations, such as Wireless Mesh Networks [4], Mobile Ad Hoc Networks [85] and Wireless Sensor Networks [2].

One of the core issues in multihop wireless networks is the problem of routing [4]. In this kind of network, routing poses a number of challenges, including the evaluation of link quality [17], the reduction of inter-flow interference [90], and the reduction of the control traffic overhead [80]. Given the possibility of an exponential number of paths between a given pair of nodes (with respect to the number of nodes in the network) and the variability of quality among these paths, routing decisions have a deep influence in network performance [4, 2, 3, 5].

One of the characteristics that most differentiate the problem of routing in multihop wireless networks from its counterpart in wired networks is the possibility of nodes interfering with one another. Due to the broadcast nature of the wireless medium, whenever a node transmits a frame, it effectively acts as an interference source for any other communication attempt within its *interference radius*, *i.e.*, the region within which the signal generated by this node is still strong enough to disrupt another transmission or reception. Consider, for example, the situation depicted in Figure 1.1. Suppose the interference radius for node 0 is delimited by the dashed circumference. While node 0 transmits a frame to node 2, node 5 cannot transmit to node 1, because both signals would be mixed, making node 1 unable to decode the desired frame. Moreover, if a carrier sense based protocol is employed at the MAC layer, a transmission from node 1 to node 5 would not be possible as well, since node 1 would detect the ongoing transmission and would wait

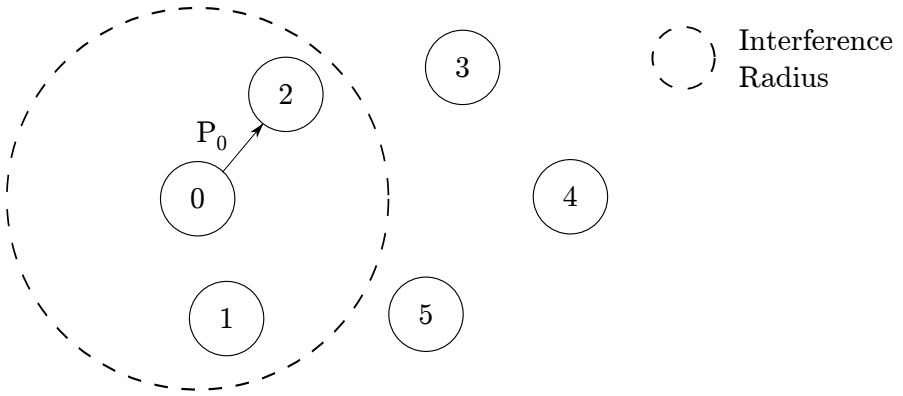


Figure 1.1: Example of a situation in which the usage of a link can affect other links of a multihop wireless network. While node 0 transmits packet P_0 to node 2, node 1, within the interference radius, cannot receive or transmit packets.

until the medium becomes idle again. Effectively, the usage of link $0 \rightarrow 2$ conflicts with both links $5 \rightarrow 1$ (both packets would collide at node 1) and $1 \rightarrow 5$ (node 1 would detect the ongoing transmission and would defer its own transmission), *i.e.*, these links cannot be used simultaneously.

The previous example illustrates the generic phenomenon of a link (or node) interfering with others in the same network, hereinafter referred to as *self interference*[1]. Depending on the specific case, however, one may categorize this self interference as either an *intra-flow interference* or an *inter-flow interference*. The intra-flow interference happens when the usage of a certain link prevents the usage of one or more other links along the same path (for a given flow). The inter-flow interference is characterized by the usage of a link preventing the usage of one or more links on other paths.

In any case, the existence of self interference in multihop wireless networks can complicate the evaluation of the performance of a given path by a routing protocol. Consider, for instance, the situation depicted in Figure 1.2. The figure shows two disjoint paths connecting nodes 0 and 4, and nodes 5 and 9. The weight shown for each arrow represents the average total delay for a frame to be successfully transmitted through each link in milliseconds (estimated by a routing metric). The figure also shows the spacing between nodes. If we assume the interference radius of each node to be 10 meters, it is possible to conclude that nodes from Path 1 interfere with other nodes from the same path, as well as with nodes from Path 2. Under these circumstances, we would like to quantify the quality of each path. Since the individual link delays are available (as discussed on

[1]Notice that some authors use this term to refer only to the case in which the usage of a link conflicts with the usage of another hop in the same path, such as in [15]. In this thesis, we opted for using this term in a broader sense, while using the terms *intra-flow interference* and *inter-flow interference* for the more specific cases.

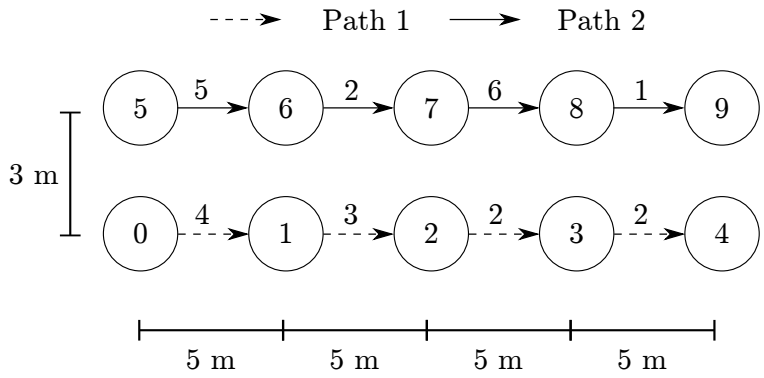


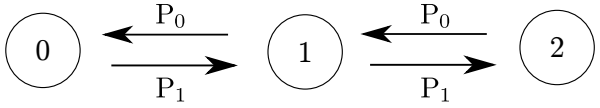
Figure 1.2: Example of a situation in which self interference affects the evaluation of the quality of a path. Once we acknowledge the existence of flows in both paths, the performance of each path may not be the same as it would if only individual flows were considered. Even in the case of a single flow, it is difficult to exactly quantify the quality of a path due to the possibility of intra-flow interference.

Chapter 2, it is possible to estimate this information in practice), one could try to use the average end-to-end delay as a metric for this task.

A first simple approach would be to simply sum the weights of all links for each path, to which one would obtain the values of 11 ms and 14 ms for Paths 1 and 2, respectively. Notice, however, that link $5 \rightarrow 6$ cannot be used simultaneously with link $0 \rightarrow 1$. Therefore, if two packets [2] are generated at the same time by the flows on paths 1 and 2, one of them would have to gain exclusive access to the medium, leading the other packet to wait its turn. For this reason, if one analyzes this situation considering that both paths will transmit packets concurrently, the resultant end-to-end delay must be higher than in the case in which paths are considered individually.

But even if one chooses to consider each flow in this example individually, this evaluation still poses further challenges. If we consider a single packet flowing through the nodes of Path 1, its expected end-to-end delay will, indeed, be 11 ms (the sum of the individual link delays), disregarding eventual processing delays within each node. In terms of throughput, by looking only at the first packet of the flow, the average during that span is of $1/11 \approx 0.09$ packets per millisecond. But by analyzing this same path considering now the existence of multiple packets for this flow backlogged at node 0, the conclusion changes. After the first packet reaches node 1, there will be a dispute for the wireless medium because links $0 \rightarrow 1$ (needed by the second packet) and $1 \rightarrow 2$ (needed by the first packet) cannot be used simultaneously. Assume that, somehow, packet 1 always has priority over packet 2. Under these conditions, packet 1 will be transmitted to node 2,

[2] In this thesis, we employ the term *frame* in the context of transmissions at the link layer, while the term *packet* is used in all other contexts.



(a) Without network coding

(b) With network coding

Figure 1.3: Example of a scenario in which a simple network coding strategy may yield considerable gains. In this example throughput can be increased by 25% under ideal conditions.

while packet 2 still waits its turn on node 0. After another dispute for the usage of the wireless medium (notice that link $0 \rightarrow 1$ cannot be used while node 2 is transmitting), packet 1 reaches node 3 at time $t = 9\ ms$. At that moment, it is finally possible for packet 2 to be transmitted because links $0 \rightarrow 1$ and $3 \rightarrow 4$ do not interfere with each other. After two more milliseconds, at time $t = 11\ ms$, packet 1 is delivered, at which point packet 2 is still being transmitted. Notice, however, that there are only 2 ms remaining for packet 2 to arrive at node 1. Assuming packet 2 has priority over packet 3, it will require only $2 + 3 + 2 + 2 = 9\ ms$ to be delivered for its final destination after the first packet has been delivered. By extending this procedure to the next packets of the flow, it is possible to conclude that the moments of arrival for each packet will be $t = 11, 20, 29, 38, 47, \ldots$. As the number of sent packets increases, the average throughput also increases, asymptotically approaching $1/9 \approx 0.11$ packets per millisecond.

Another issue that can further add complexity to the process of evaluating the performance of multiple paths in a multihop wireless network is network coding. Network coding is a new routing paradigm first proposed by Ahlswede *et al.* [1]. The traditional routing paradigm defines that the forwarding process consists in replicating a packet received through an input link to one or more output links. However, except for eventual changes in some control fields in the lower layers, packets' contents are never altered. Network coding, on the other hand, allows packets to be altered in order to improve network performance. Specifically, under the paradigm of network coding, two or more packets in an intermediate node's buffer can be combined resulting in a single message, called a coded packet (as opposed to native packets, which are the original non-coded packets). The coded packet, which has roughly the same size as its correspondent native packets, is then transmitted through one or more links, so that it can be eventually received by the destinations of the original native packets. Assuming some given conditions are met,

these destinations are able to decode the packet, thus retrieving their correspondent native packets.

In order to combine native packets into a coded one, it is necessary to use a predetermined code, *i.e.*, an operation to be performed over the native packets that can be uniquely reversed provided that the necessary parameters are available. One common such operation is the XOR (Exclusive Bitwise Or). Figure 1.3 illustrates how this operation can be employed in order to improve the performance of a specific scenario. In this scenario, nodes 0 and 2 exchange packets through node 1. Suppose in a given moment, both nodes 0 and 2 have packets in their buffers to be transmitted (respectively, packets P_1 and P_0). Links $0 \rightarrow 1$ and $2 \rightarrow 1$ cannot be used simultaneously. Therefore, one of the packets gets to be transmitted first (say packet P_0), while the other is transmitted later. Suppose that the scheduling in which the transmissions occur is such that, after the first two transmissions, node 1 has both P_0 and P_1 in its buffer. At this point, if the network does not employ network coding, each packet will have to be sent separately, as shown by Figure 1.3a, resulting in a total of 4 transmissions. However, if the network is capable of performing network coding, node 1 can compute a new coded packet $P_c = P_0 \oplus P_1$, *i.e.*, a combination of both native packets. As shown in Figure 1.3b, P_c is transmitted to both nodes 0 and 2, which can be done in a single transmission given the broadcast nature of the wireless medium. Once node 0 receives P_c, it can obtain P_0 by simply computing $P_0 = P_c \oplus P_1$ (assuming it still has P_1 stored in its memory). Likewise, node 2 can retrieve P_1, provided P_0 is still known.

In the previous example, the number of transmissions required to complete the bidirectional communication between nodes 0 and 2 drops from 4 to 3 with the usage of network coding, a gain of 25% in terms of aggregated throughput, assuming all transmissions take the same amount of time. Although the feasibility of network coding depends on a number of aspects (*e.g.*, the availability of useful packets in a node's buffer and the capability of the receiving nodes to decode coded packets), there are already in the literature methods that dynamically detect cases that fulfill the necessary conditions (called *Coding Opportunities*) and combine packets accordingly [56].

Notice that there is a connection between network coding and the problem of interflow interference in multihop wireless networks. In the scenario depicted in Figure 1.3a, for example, individually, each flow has an end-to-end delay equivalent to two transmissions. However, the two paths are completely interfering, which results in a lack of opportunities for parallel transmissions. Ultimately, this lack of parallelism leads to an aggregated

throughput 50% lower than would be obtained if those two paths were completely independent. By using network coding, it is possible to mitigate this issue by "transforming" links $1 \rightarrow 0$ and $1 \rightarrow 2$ (which interfere with each other) in a single link, thus reducing the level of interference both paths produce on each other.

This leads to the interesting question of whether a coding-aware routing protocol — *i.e.*, a protocol that is aware of the coding capabilities of the network — could provide performance gains by proactively routing flows through paths that result in network coding opportunities in order to reduce inter-flow interference. In fact, there are already proposals of full routing protocols that try to achieve such gains [30, 58, 61, 87, 88, 101].

One shortcoming of these proposals, though, is that they do not integrate network coding with other resources to reduce self interference (such as simply separating flows). This reduces the scope of the achievable gains, since those proposals have limited resources to reroute flows in order to avoid interference, as will be shown in greater detail throughout this thesis. On the other hand, there is a group of proposals that do not take network coding into consideration and instead explores only the possibility of choosing less interfering paths by further separating flows. Besides not considering network coding as a valuable tool for avoiding inter-flow interference, many of those proposals are based on complex models which are infeasible in practice (such as models based on linear programming [10]), due to the time constraints involved in computing routes on such networks. Other more practical proposals exist, but they employ traditional path finding algorithms that cannot properly handle the effects that flows have on one another.

The goal of this thesis is, therefore, to tackle the problem of route selection in multihop wireless networks taking into consideration the issue of self interference (both of the inter and intra-flow kinds). Specifically, we develop a framework for interference aware routing algorithms. Within this framework, we explore both the cases of networks with the capability of performing network coding — resulting in a proposal called ICAR (Interference and Coding-Aware Routing) — and of networks with no coding capabilities — resulting in IAR (Interference Aware Routing). For the purposes of this thesis, we define as the objective of the routing problem the selection of a set of paths that can deliver packets from all active network flows with the maximum aggregated throughput. We take a novel approach to this problem, based on a new path selection algorithm which is capable of taking into consideration the effect that the selection of a given path has on other paths. The adoption of such algorithm allows us to solve the problem of route selection having a global vision of all flows, instead of using traditional shortest path algorithms.

We further investigate some practical issues regarding the implementation of path selection methods in multihop wireless networks, such as the effect of data traffic on the estimates of link qualities taken by routing protocols. Based on our investigations, we are able to propose practical solutions to such problems, further increasing the gains of our proposal. We also study issues regarding the implementation of network coding in real networks, providing optimizations and solutions to these problems.

In order to evaluate the performance of IAR and ICAR, we implement them in both simulated and real network environments. Through experiments in both environments, we demonstrate the importance of considering the route selection problem under the prism of the existent network flows, as well as the efficiency of our proposals in recognizing the situations in which it is possible to obtain gains by selecting path sets that differ from the set of individual optima. In simulated environments, we demonstrate that our routing selection method can increase the network aggregated throughput up to 90% in comparison to traditional routing mechanisms. We also show that gains can stem from a number of different network topologies with different characteristics. On the other hand, we provide an implementation of our method that can be used in real networks deployed with off-the-shelf hardware with limited resources. This implementation allows us to demonstrate the viability of our proposal even in such a constrained environment.

1.1 Conventions and Formal Problem Description

Throughout this thesis, we employ the following notations and conventions. A network topology is represented by a graph $G = (V, E)$, where V is the set of network nodes and E is the set of links formed between them. The graph G is usually considered to be directed, although in some cases we employ undirected graphs because the weights in each direction are considered to be equal or very similar. Throughout the text, nodes are represented by numbers, starting always from 0, or italic lowercase letters (especially when referring to a generic node).

A flow between two nodes a and b is represented by $a \Rightarrow b$, which must not be confused with a link between nodes a and b, in this case represented by $a \rightarrow b$. Notice that a flow can be bidirectional, in which case it is represented as in $a \Leftrightarrow b$. A path from node a to node c using node b as a relay is denoted by $a \rightarrow b \rightarrow c$. Every flow f has a source node, denoted by $src(f)$, as well as a destination, denoted by $dst(f)$. The same applies to any

path $\mathcal{P}$[3].

Neither IAR or ICAR assume a special type of multihop wireless network to work. However, in this thesis we maintain as a focus the case of wireless mesh networks. Specifically, all our experiments are performed in static multihop wireless networks (in opposition to mobile ad hoc networks) and our discussions about the computational costs of our solution assume the typical computing power of an off-the-shelf IEEE 802.11 router (in opposition to a typical low power sensor node). Due to limitations in the current state of art in terms of network coding (that will be discussed in more depth in Chapters 2 and 3), we assume each node to use a single fixed transmission rate.

Both the simulated and real environments used for evaluation in this thesis are based on nodes with IEEE 802.11 wireless interfaces. Although IAR and ICAR can be used with other MAC and physical layer protocols, we assume that the MAC layer uses ARQ (Automatic Repeat Request) as a mechanism for transmitting unicast frames with a maximum number of k retries. We also assume that broadcast frames are transmitted a single time.

Regarding the flows, we assume all network flows are backlogged, *i.e.*, all flows generate packets at a rate that is sufficient to saturate any path. In Chapter 5 we further discuss this assumption. A flow is defined as the set of packets that share common source and destination nodes. In other words, all packets transmitted from a given node a to a given node b are considered to belong to the same flow, regardless of concepts from higher layers, such as connections, transport protocols or ports.

It is beyond the scope of this thesis to find the optimal scheduling for transmissions in the MAC layer. We assume a fair and deterministic scheduling, in which all nodes have equal opportunities to transmit (with respect to their neighbors). More details about this scheduling are given in Chapter 3.

A formal description of the problem studied by this thesis can be stated as follows. Let $G = (V, E)$ be a graph representing the current state of the network (in terms of nodes and links). Let the weights associated with each edge on the graph represent the delivery probability of the respective link. Let F denote the set of flows currently active in the network. Under these conditions, find a path set PS such that:

- for every flow $f \in F$, there is a correspondent path $\mathcal{P} \in PS$, such that $src(f) =$

[3]For the sake of clarity, in this work a generic path is always referred to as $\mathcal{P}$, as opposed to a probability p and a packet P.

$src(\mathcal{P})$ and $dst(f) = dst(\mathcal{P})$, as long as it is feasible (*i.e.*, that there actually is at least one such path in the network);

- the sum of the throughputs for all flows achieved using PS must be maximum.

The evaluation of the aggregated throughput achievable with path set PS must take into account the self interference generated by each link in the solution, as well as the network coding capabilities of the network.

1.2 Text Outline

The rest of this thesis is organized as follows. Chapter 2 presents the state of art regarding the topic studied by this thesis. The chapter covers traditional routing approaches for multihop wireless networks, approaches aware of inter and intra-flow interferences, as well as network coding and coding aware routing protocols.

Chapter 3 presents our proposal for solving the problem of routing in multihop wireless networks. In this chapter, we present the algorithms employed by IAR and ICAR divided in two groups: evaluation of the performance of a given path set and generation of candidate sets. The algorithms are analyzed in terms computational complexity and we also discuss their accuracy in choosing the best path set. We also further discuss some of the assumptions made in this work, as well as project decisions that had to be made.

Chapter 4 discusses practical aspects of the implementation of IAR and ICAR. We consider issues such as detecting and announcing the existence of network flows, the transmission of coded packets in the MAC layer and the estimation of the links' qualities.

Chapter 5 presents the evaluation of both proposals in terms of network performance. We present and discuss experiments conducted in both simulated and real testbeds. Our experiments cover a range of aspects, using both especially crafted scenarios and generic topologies.

In Chapter 6, we summarize the main findings and contributions of this work. Ideas for future work are also presented.

Appendix A shows some interesting and unexpected results regarding the errors in the estimate of links' qualities on networks under heavy loads. Our results suggest a common hardware deficiency in implementations of the IEEE 802.11 standard that causes higher estimation errors than expected.

Finally, Appendix B describes a technique for estimating the joint reception probability for a coded packet (*i.e.*, the probability that all intended destinations of a coded packet correctly receive it).

Chapter 2

Related Work

In this chapter, we discuss the current state of the research in the area of routing in multihop wireless networks. This discussion is organized in three major parts:

- traditional routing;
- interference aware routing; and
- coding aware routing.

We begin by presenting a historical overview of the evolution of some of the most relevant proposals in terms of traditional routing mechanisms, including routing protocols and routing metrics. We proceed by presenting proposals that specifically target to choose paths with low intra-flow and inter-flow interference. Finally, we approach the area of network coding, with the main focus on coding aware routing, *i.e.*, routing protocols and metrics that use network coding to improve network performance.

2.1 Traditional Routing

There is a vast literature on routing in multihop wireless networks. Most of the traditional routing protocols were proposed during the late 1990s or early 2000s.

Perhaps one of the first widely adopted routing protocols was DSDV [81] (Destination-Sequenced Distance Vector), a proactive protocol: it proactively maintains a table containing routes for all network nodes. DSDV is said to be a distance vector protocol, because it uses the Bellman-Ford algorithm [22].

Another distance vector-based protocol is AODV [82] (Adhoc On-demand Distance Vector). One major difference between AODV and DSDV is the fact that AODV is a

reactive protocol, meaning that routes are only computed when necessary. Whenever a new flow is started, its source node triggers a route discovery process by broadcasting a *route request*. This request is diffused by network nodes (recording the sequence of nodes it has traversed) until it reaches the destination (possibly multiple times, using different paths) or an intermediate node that already has a valid route to the destination. In both cases, a route reply is returned to the source node, containing the route just found.

DSR [52] (Dynamic Source Routing) is similar to AODV in the sense that both are reactive protocols and use route discovery mechanisms based on flooding. However, DSR is based on source routing, *i.e.*, once the source node learns the best route for the destination, it includes this path in all data packets. Hence, intermediate nodes do not need to store route information, instead they find the information of the next hop in the packet itself.

Another traditional protocol is OLSR [21] (Optimized Link State Routing). Similarly to DSDV, OLSR is proactive, constantly maintaining routes for all network nodes. However, OLSR is a link state protocol, meaning it relies on keeping a complete view of the network in each node. Nodes detect their neighborhood by periodically broadcasting *hello* packets (when a node receives a *hello* packet, it recognizes the sender as its neighbor). Also periodically, an OLSR node broadcasts *topology* packets containing its current view of the neighborhood. Differently from *hello* packets, *topology* packets are retransmitted by other nodes, flooding the network. This results in all nodes having knowledge of the complete network topology. Whenever a change is detected in the topology (as internally known by nodes), OLSR executes the well-known Dijkstra algorithm [26] to find the new best routes.

Possibly, the main goal of those and other proposed routing protocols for multihop wireless networks was reducing the overhead of finding and maintaining routes. For example, as a reactive protocol, AODV has the advantage of only consuming network bandwidth when a new flow is started for which there is no previously known route. As another example, OLSR employes the concept of Multipoint Relays (MPR) [21]. An MPR set is a subset of neighbors such that any two-hop neighbor (*i.e.*, nodes that cannot be reached directly, but by using a single relay) can be reached in two hops through a node of this set. Although the complete set of neighbors fits the definition of MPR, there usually exists another smaller subset for which the property holds. In OLSR, whenever a node receives a control packet that has to be flooded through the network (such as a *topology* packet), it only performs a retransmission if it belongs to the MPR set of the last hop traversed by the packet. With this strategy, OLSR can reduce the overhead of the flooding, while

still guaranteeing that all nodes receive the information (not considering possible packet losses).

Although other routing protocols have been proposed more recently (such as [16, 67]), those traditional protocols established many techniques and frameworks that can be seen in most modern proposals. One aspect that has deeply evolved, though, regards the *routing metrics*, *i.e.*, the mathematical models used to classify the quality of links and routes in order to provide the basis for selecting the best paths. As originally proposed, those traditional routing protocols all employed the same simplified approach to route quality evaluation that often resulted in poor routing decisions. This simplified approach consisted in employing a routing metric known as Hop Count, which, as the name implies, simply classifies a route based on its number of hops. As will be further discussed in this chapter, the number of hops can have a low correlation with the actual performance of a path (in terms of metrics such as end-to-end delay, throughput and packet loss rate). Since Hop Count could result in low performance, much work has been done in proposing alternative routing metrics [17].

The Expected Transmission Count [24, 23, 25] (ETX) metric is possibly the most important proposal in terms of routing metrics. Although there are newer proposals which are capable of outperforming ETX, it still serves as the base for many other methods in multihop wireless networks in general.

The ETX metric classifies links and paths according to the expected number of link layer transmissions needed to transmit a packet (through a single link or the complete path). To this end, ETX first estimates the delivery probability of each network link by periodically broadcasting probe packets. Whenever a node receives a probe from one of its neighbors, it updates a statistic of the percentage of probes successfully received for that neighbor among a window of the last w probes (where w is a configurable parameter). Nodes periodically report their computed estimates for the delivery probabilities back to their neighbors (usually by simply adding them to the probes). By applying this method, a node a always has relatively up-to-date estimates of $d_{a\to b}$ and $d_{b\to a}$, the delivery probabilities in each direction[1] for its link for any neighbor b.

Since the successful transmission of a unicast frame according to the IEEE 802.11 standard (the main focus of the work that proposes ETX) requires an acknowledgment frame to be received back by the transmitter, Couto *et al.* [24] define the success proba-

[1] Although they represent probabilities, in this thesis we opted for representing the delivery probability of a link with the letter d, instead of p, to differentiate it from other kinds of probabilities associated with network links

bility in one link layer transmission attempt through a link $a \rightarrow b$ as:

$$p_{a \rightarrow b} = d_{a \rightarrow b} \times d_{b \rightarrow a}. \tag{2.1}$$

Authors also assume that the transmitter keeps retrying to transmit a packet until the first success. Hence, the number of transmissions until the first success on a generic link $a \rightarrow b$ can be modeled as a geometric random variable with probability $p_{a \rightarrow b}$. The ETX metric for the link $a \rightarrow b$ is then given by:

$$ETX_{a \rightarrow b} = \frac{1}{p_{a \rightarrow b}} = \frac{1}{d_{a \rightarrow b} \times d_{b \rightarrow a}}. \tag{2.2}$$

For a complete path composed of multiple links, the ETX metric is defined as the sum of the individual links' ETX values.

One of the underlying assumptions of the ETX metric is that paths that need less link layer transmissions to deliver a packet will offer better throughput. If all network links had the same delivery probabilities, then the ETX metric would be equivalent to the Hop Count metric. However, since that is seldom the case in real topologies, ETX often outperforms Hop Count, due to the latter tending to include very long and lossy links in its paths.

Despite its superiority with respect to Hop Count, a number of issues can be pointed out in the formulation of the ETX metric. One such issue is the size of the probe packets used to estimate the delivery probabilities. Couto *et al.* [24] suggest that *hello* packets, used by most protocols to discover neighborhood, can be used as probes, avoiding the necessity of creating and transmitting another set of control packets. However, they do not define the size of the probes. In practice, it is common to find implementations of the ETX metric that simply use variable probe sizes (*i.e.*, the size of each probe is defined by its contents, which usually vary with the number of neighbors of each node). The result is that the delivery probabilities used by ETX may be computed with different probe sizes, even though the packet size directly affects it [75]. Incidentally, nodes with few neighbors may have their links' qualities overestimated with respect to those from nodes in more dense regions of the network. A related issue is that of using the same value $d_{a \rightarrow b}$ for the delivery probability of an acknowledgment frame and a data frame (possibly much larger).

Finally, the ETX metric is completely oblivious to the data rate of each link. The IEEE 802.11g standard, for example, defines 8 different transmission rates, varying from

6 Mb/s up to 54 Mb/s [49] and usually wireless interfaces implement some kind of rate adaptation algorithm, such as the ones found in [45, 40, 100, 57, 79, 12, 53]. That disregard for the link layer transmission rate results in two problems. The first is the potential mismatch between the estimates for the delivery probabilities as computed by ETX (usually performed at the lowest available transmission rate due to the use of broadcast probes) and the actual links' delivery probabilities (at the transmission rate used at the link layer). As a rule of thumb, higher transmission rates result in lower delivery probabilities [75]. Hence, when evaluating a link operating at a high transmission rate, ETX has the tendency to overestimate the delivery probability. On the other hand, if multiple transmission rates are available, it is possible that different links operate at different transmission rates within the same network. In this case, ETX may evaluate two links with the same estimated delivery probability as being equally good when, in fact, one of them operates with a much higher transmission rate, thus resulting in higher throughput.

The Expected Transmission Time metric [28] (ETT) is an evolution of the ETX metric that aims at correcting the issue of ignoring the transmission rate of the link when computing its cost. The idea is to divide the ETX of the link by its transmission rate, thus obtaining a value that roughly represents the **time** necessary until the first success in a unicast frame transmission. To this end, two different implementations are proposed: one by Draves *et al.* [28] and another by Bicket *et al.* [13].

Draves *et al.* [28] suggest that the transmission rate of the link should be estimated using a technique known as *packet-pair probing*, in which two probes are sent, one immediately after the other. The receiver then calculates the time difference between the arrival of both probes and uses this value to estimate the link's transmission rate. Bicket *et al.* [13], on the other hand, suggest that probes should be sent at every transmission rate available at the link layer and the respective delivery probabilities computed. The ETT metric would then be computed for each transmission rate (using its respective delivery probability) and the lowest value (the best one) should be used as the link cost.

The implementation suggested by Draves *et al.* [28] has the disadvantage of possibly using a wrong estimate for the link's delivery probability, since they are computed based on probe data sent at the basic rate irrespective of the actual link transmission rate. On the other hand, the method proposed by Bicket *et al.* [13] introduces a greater overhead due to the probes at many different transmission rates. Another issue with the former method is the assumption that the link layer transmission rate will actually match the

transmission rate that yields the best ETT value. Since there is no coordination between the metric and the rate adaptation algorithm (assuming there is one), the link may use a different rate, resulting in a wrong evaluation of the link quality.

The Metric Aware Rate Adaptation (MARA) mechanism [75] does not fit the traditional definition of routing metric. It is actually a joint solution for two problems: routing metric and rate adaptation. From the routing metric perspective, this mechanism tries to solve the implementation issues with the ETT metric by employing a different strategy.

Similarly to the implementation proposed by Bicket *et al.* [13], MARA estimates the delivery probabilities in every available transmission rate. However, instead of sending probe packets at all rates, MARA is able to do so using fewer probes (for example, with 4 of the 12 transmission rates of the IEEE 802.11b/g mixed mode). With the estimates at each rate of this subset, MARA employs a probability conversion technique to obtain estimates for the remaining rates.

MARA then proceeds to compute the link cost at each transmission rate and chooses the lowest value as the definitive cost. Since MARA performs both routing and rate adaptation decisions, the rate that yields the lowest cost is assigned to the link, guaranteeing the consistency between the routing metric estimates and the actual link parameters. Like the previously cited metrics, though, MARA does not consider any intra-flow or inter-flow interference effects during route choices.

In [75], the performance of MARA is evaluated using a basic link state routing protocol called SLSP (Simple Link State Protocol). This protocol is proposed and implemented in [75], with the goal of being easily extensible with respect to routing metrics.

The Minimum Loss metric [78] (ML) uses a different approach for classifying links and routes. Instead of trying to model the delay like the previously presented proposals (notice that ETX uses the number of link layer transmissions as an estimator for the delay), ML tries to achieve better performance by minimizing the end-to-end packet loss.

To this end, ML assigns as the cost for each link $a \rightarrow b$ the success probability for a single attempt of transmission at the link layer, as computed by the ETX metric (basically, the reciprocal of the ETX metric for the link). The cost of a path composed of multiple links is then defined as the product of the individual link costs.

The rationale behind the ML metric is that wireless links can become very lossy, which reduces their effective throughput. The formulation of the metric would make it avoid such lossy links (if possible), possibly resulting in good throughput.

All the issues related to the disregard for the transmission rate found in the ETX formulation apply to ML as well. Another issue with ML is its lack of sensitivity in terms of the number of hops: a very long path composed only of perfect links (*i.e.*, with delivery probability of 1) is preferred by ML over another shorter option containing a single non-perfect link. In some scenarios, this may result in ML choosing paths with more hops than other metrics, possibly accentuating issues such as inter-flow and intra-flow interference [75].

2.2 Interference Aware Routing

Traditional routing metrics such as ETX and ETT are still the most widely adopted options in practical implementations. Nevertheless, there are already a number of proposals in the literature of routing metrics and protocols that are aware of intra and inter-flow interference and try to avoid them. In the next few sections some of the most relevant work in that regard is discussed.

2.2.1 Theoretical Results

Many works have discussed the theoretical aspects regarding the effects of interference on the achievable throughput in multihop wireless networks. Gupta and Kumar [39] provide bounds for the per node throughput assuming random node placement and that each node communicates with exactly one other node. Li *et al.* [62] extend the work of Gupta and Kumar by analyzing different traffic patterns and the effect of the usage of the IEEE 802.11 medium access schedule instead of a global schedule scheme.

Jain *et al.* [50] explore a different approach. Instead of looking at networks with random node placements and traffic patterns, they create mathematical models that can be applied to specific networks (given their topologies) and specific traffic demands. Their models, based on linear and integer programming, are able to output the optimum routes and optimum medium access schedule for a given instance. They consider mostly the case of a single source/destination pair, but discuss how the model can be extended for multiple pairs. They present models for multipath routing (*i.e.*, assuming the flow can be split between multiple paths) and also for the single path case, which they argue is more useful in practice.

In the same work, Jain *et al.* also discuss how to find upper and lower bounds for the throughput based on a structure called *Conflict Graph*, since solving their models may be

computationally unfeasible for larger networks. A conflict graph is a representation of the relationships between the links of a network in terms of interference. In a conflict graph, each link of a network becomes a vertex and two vertices are connected if, and only if, they cannot be used simultaneously (*i.e.*, they interfere with each other). Jain *et al.* prove that, if all links have the same capacity R, the sum of the used capacities for all links forming a clique in the conflict graph of a network cannot exceed R. This effectively places an upper bound in the maximum achievable network throughput considering interference. Authors, however, show that this upper bound is not always tight, *i.e.*, they cannot give guarantees for how close this bound is to the actual maximum throughput.

The method presented by Jain *et al.* is not supposed to be used as a practical route selection mechanism for a number of reasons. First, it can only find optimal routes assuming the scheduling for the access of the wireless medium can be globally controlled in order to correspond to the schedule output by the solution of their model. Under a distributed non-deterministic schedule (such as the one obtained by employing the IEEE 802.11 standard), the routes may not be optimal. Moreover, in order to find the optimal results, their method needs the computation of all maximal independent sets of the network conflict graph. Not only this is an NP-Hard problem [32], but the number of independent sets may grow exponentially, further increasing the complexity of solving their mathematical programming models. Nevertheless, by using simulations, authors are able to demonstrate that optimal routes found using their models can double the throughput in some scenarios, with respect to traditional routing.

The work by Jain *et al.* suggests that there is a strong synergy between routing choices and the scheduling of link usage at the link layer. In fact, there is a vast literature on the topic of link scheduling in multihop wireless networks [38, 35, 36, 37]. In [38], Grönkvist and Hansson compare two approaches for assigning a schedule for the usage of links. The first, known as *graph model*, is based only on knowledge of the network topology, while the second considers knowledge of propagation characteristics of each link (*e.g.*, the level of received signal for the receiver of a link). Using simulations, they show that the second approach tends to result in more aggressive schedules (*i.e.*, schedules with more links being used simultaneously), which may yield better network performance. Gore *et al.* provide an algorithm to find the best schedule under the graph model.

Goussevskaia *et al.* [36] tackles the issue of finding good schedules in networks capable of operating with multiple rates at the link layer. In a later paper, Goussevskaia and Wattenhofer [37] study the scheduling problem in networks that employ a method known

as *Successive Interference Cancellation*. Under this method, when two or more packets collide at a node, it might be able to decode the strongest signal provided that it is sufficiently stronger than the others. After the respective packet is retrieved, the node might be able to subtract the corresponding signal from the total received signal in the digital domain. The remaining signal can then be reevaluated, in order for another packet to be decoded. In theory, assuming favorable conditions and enough hardware resolution, it is possible for a node to receive multiple packets at the same time. Under these conditions, more links can be schedule simultaneously.

Notice that, although link scheduling is an active research area, as explained in Chapter 1, in this thesis we assume predefined medium access rules, instead of trying to determine the optimal scheduling.

2.2.2 Practical Proposals

In addition to the theoretical results, the literature contains a number of practical proposals of protocols and metrics for the problem of interference aware routing. For the sake of organization, we divide these proposals in two groups: the ones targeted at multi-radio networks and the ones for single radio networks.

2.2.2.1 Multi-radio Networks

Although there are exceptions, most proposals of the protocols and metrics for the interference-aware version of the routing problem consider the case of multi-radio networks [28, 91, 90]. In this kind of network, each node possesses multiple radio interfaces capable of operating at different non-interfering channels. Thus, there might be multiple links connecting a given pair of nodes. At the same time, when representing the topology of this kind of network, the channel for each link must also be known.

Draves *et al.* [28] propose a routing solution for the problem of intra-flow interference aware routing in multi-radio networks that consists of using a routing metric that extends the ETT metric to include information regarding the channel diversity of the path. This new metric, called Weighted Cumulative Expected Transmission Time (WCETT), is defined for a path composed of n links as:

$$WCETT_n = (1-\beta)\cdot\sum_{l=1}^{n} ETT_l + \beta\cdot max(X_j),\ \text{for } 1\leq j\leq k \text{ and } 0\leq\beta\leq 1. \qquad (2.3)$$

In the expression, k denotes the number of available channels and X_j is the sum of ETT values for all links in the j-th channel. The expression is simply an average (weighted by the adjustable parameter β) between the traditional ETT of the path and a second component that depends on the distribution of the channel selection through the path. A path using all links in the same channel receives a WCETT value equivalent to its ETT value. However, as links with different channels are used, the total cost decreases due to the second term. By having a tendency to find paths with channel diversity, authors expect to reduce intra-flow interference, achieving higher throughputs.

One issue with the formulation of the WCETT metric is the lack of isotonicity. This mathematical property states that, given two paths $\mathcal{P}_1$ and $\mathcal{P}_2$ such that the cost of $\mathcal{P}_1$ is greater than the cost of $\mathcal{P}_2$, the addition of a new link l to each path cannot change the relative order of the costs. In other words, the new path $\mathcal{P}_1 \rightarrow l$ cannot have a lower cost than the new path $\mathcal{P}_2 \rightarrow l$ [28]. Traditional algorithms for the problem of shortest paths in graphs, such as the algorithms by Dijkstra and Bellman-Ford [22, 26], usually require the metric to be isotonic. If this property is not respected, those algorithms cannot guarantee the optimal solution [89]. Therefore, these traditional algorithms are not sufficient to find the optimal solution considering the WCETT metric. Authors acknowledge that, but they opt for using the Dijkstra algorithm regardless. Despite this issue, Draves *et al.* show performance improvements of WCETT with respect to more traditional metrics in their evaluation.

Notice that WCETT is only different from ETT in multi-radio networks. Authors do not propose a solution for the case of single radio networks. In this thesis, we chose to evaluate the single radio scenario, which does not benefit from the usage of WCETT.

In [91], authors also explore the problem of routing in multi-radio networks. Differently from the work of WCETT, Tang *et al.* [91] split the problem in two separate issues: channel assignment and routing. They propose a channel assignment heuristic that constructs a multi-channel topology with, hopefully, low potential interference (*i.e.*, nearby links tend to use different channels, allowing them to be used simultaneously without interfering with each other).

After channels are assigned, authors propose the usage of a QoS-aware routing method devised in the same paper. This method is based on a linear programming model created by the authors with the objective of minimizing the sum of interfering traffic. For each network link, authors define a set of interfering links. Given the amount of traffic allocated to a given flow, authors define the amount of interfering traffic for that link as the product

of the total traffic amount of the link by the number of interfering links. The goal of the linear programming formulation is then to minimize the summation of this value for all network links. Notice, however, that this formulation uses the concept of data as fluid, meaning that data can be split arbitrarily among multiple paths. Since authors base their work in packet networks, they propose a heuristic for finding a single path solution for the routing problem.

The work by Tang *et al.* is interesting from a theoretical standpoint, but it presents some practical issues. For instance, authors define the capacity of a wireless link as simply the link layer transmission rate. They do not take into account, for example, the effect of the delivery probability in the actual achievable throughput. They also disregard the fact that links with lower delivery probabilities occupy the medium for a longer period of time (due to retransmissions), increasing the interference.

Subramanian and Buddhikot [90] also tackle the problem of interference aware routing in the context of multi-radio networks. To this end, they propose the iAWARE routing metric which is strongly based on WCETT. They define the cost of a link as the ratio between its ETT and a value called IR (Interference Ratio) defined for a link $a \rightarrow b$ as:

$$IR_{a \rightarrow b} = \frac{SINR_{a \rightarrow b}}{SNR_{a \rightarrow b}}, \tag{2.4}$$

where $SNR_{a \rightarrow b}$ is the Signal to Noise Ratio of the link, while $SINR_{a \rightarrow b}$ represents the Signal To Interference plus Noise Ratio. To compute the second term, authors sum the background noise and the signal strength for all links that can be overheard by node b (*i.e.*, links that can potentially interfere with transmissions to node b), weighted by the percentage of time each link is used. For a path composed of multiple links, Subramanian and Buddhikot use the same formulation employed by Draves *et al.* in the WCETT metric, but substituting the ETT of the links by their new metric iAWARE.

Due to their resemblance, the iAWARE metric and the WCETT metric share most virtues and issues. The most relevant difference between them is the fact that iAWARE puts more emphasis on avoiding links that are currently more affected by other interfering links. This makes iAWARE useful for usage on single radio networks, differently from WCETT that becomes identical to the ETT metric in this case.

One potential issue with iAWARE is the possibility of instability in the routing decisions. Whenever a new flow begins, it starts sending packets through the links that compose the path selected by iAWARE. The fact that those links are now in use will

probably change their usage rate, which is an information used by the metric. Once that information is updated by the routing protocol, it might change the route selection, resulting in new routes. To the best of our knowledge, however, there are no studies on the stability of the route selections by iAWARE.

2.2.2.2 Single Radio Networks

Yang *et al.* propose an interference aware routing metric called Interference-aware Resource Usage (IRU) [102]. For computing the cost of a link $a \rightarrow b$, authors define the metric IRU as:

$$IRU_{a \rightarrow b} = ETT_{a \rightarrow b} \cdot |N_a \cup N_b|, \tag{2.5}$$

where N_a and N_b represent the set of neighbors of nodes a and b, respectively. In the model considered by Yang *et al.*, a node that is neighbor of a or b cannot transmit simultaneously with transmissions on link $a \rightarrow b$. For that reason, $IRU_{a \rightarrow b}$ represents the total amount of time that a transmission on link $a \rightarrow b$ would consume of the medium usage time available for those neighbors. By finding paths that minimize the sum of IRU values for their individual links, Yang *et al.* expect to minimize the amount of network resources used by the path.

An issue with the IRU metric is that it is oblivious to the actual existent network flows. It always chooses paths that use low amount of network resources, even if there are no other flows that would benefit from those savings. For instance, Yang *et al.* use the example depicted in Figure 2.1 to illustrate a case where IRU is able to find a better route than ETT would. In this example, authors assume that $ETT_{0 \rightarrow 2}$ is slightly lower than $ETT_{0 \rightarrow 1}$ (Figure 2.1a), which would cause node 0 to route all its traffic through relay 2. Due to interference, that would decrease the performance of both flows $0 \Rightarrow 3$ and $4 \Rightarrow 5$. Since node 1 has less interfering neighbors, IRU would choose 1 as a relay (Figure 2.1b), which would lead to better performance by reducing interference. Notice, however, that IRU does not take into account whether flow $4 \Rightarrow 5$ actually exists in the network, rerouting flow $0 \Rightarrow 3$ through a worse path regardless.

Cheng *et al.* propose a method for admission control and QoS routing in wireless mesh networks called MARIA (Mesh Admission control and qos Routing with Interference Awareness) [19]. Their method uses the concept of Conflict Graph in order to evaluate the bandwidth available for a given node.

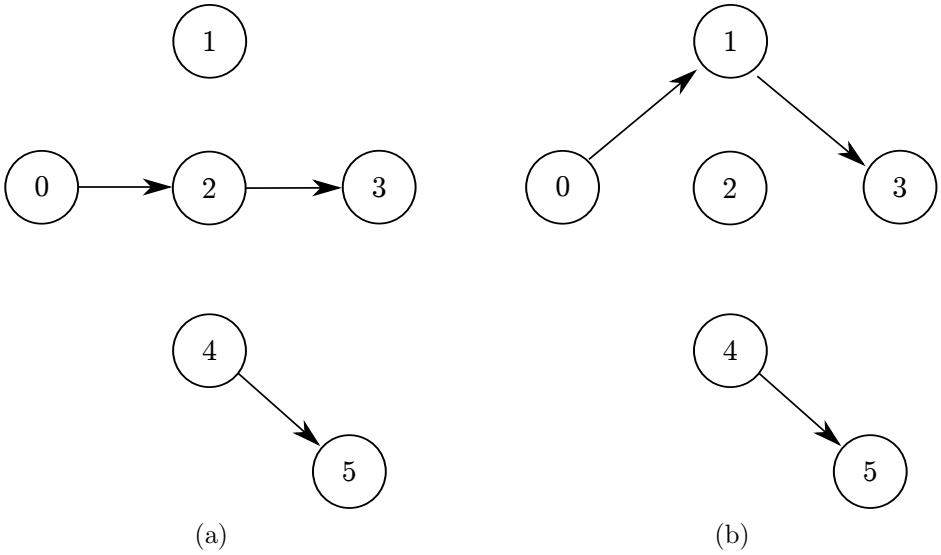


Figure 2.1: Example of a network topology in which the IRU and ETT metrics might result in different route choices. Figure 2.1a shows a possible choice made by ETT, assuming link $0 \rightarrow 2$ has a slightly lower ETT than link $0 \rightarrow 1$. Figure 2.1b shows the choice made by IRU under the same conditions.

The method proposed by Cheng *et al.* works as follows. When a new flow begins, the source node starts a route discovery and bandwidth allocation process. This source node assembles a local conflict graph (*i.e.*, containing only two hop information), based on information obtained through previously exchanged hello packets. The node then computes the cost of all maximal cliques it belongs to and checks if the available bandwidth is enough for the new flow. If it is, the source node broadcasts a route request packet. Otherwise, the flow is rejected. Upon receiving a route request packet, an intermediate node computes its available bandwidth using the same process. If the bandwidth is enough to allocate the new flow, the node forwards the route request by rebroadcasting it. Each node traversed by the route request records its current available bandwidth in the packet. When the destination node receives multiple route requests (by multiple paths) it chooses the one with the highest minimum bandwidth and informs the path back to the source.

Notice that MARIA does not have as a goal finding the path set that provides the highest aggregated throughput for the current network flows. Its objective, then, is different from the proposal presented in this thesis. MARIA simply seeks to guarantee that enough bandwidth will be available to each network flow. One issue with this work is that MARIA may allow flows to be allocated in situations where there is not enough available bandwidth. This happens because the maximal cliques only provide an upper bound on the available bandwidth, *i.e.*, the fact that the costs of all cliques are lower than the

capacity R is necessary but not sufficient to guarantee that the bandwidth is available to the new flow [50]. However, Cheng *et al.* do not discuss this possibility of allocating flows to paths with insufficient resources. Another issue with this proposal is the lack of usage of delivery probability information when computing the available bandwidth of a path.

Ashraf *et al.* propose the Expected Link Performance metric (ELP) [6], which also takes into account interference in its formulation. Authors define the ELP metric for a link $a \rightarrow b$ as:

$$ELP_{a\rightarrow b} = \frac{1}{\alpha d_{a\rightarrow b} + (1-\alpha) d_{b\rightarrow a}} \times \frac{Max(IF_a, IF_b)}{1 + Max(IF_a, IF_b)}, \tag{2.6}$$

where $d_{a\rightarrow b}$ and $d_{b\rightarrow a}$ are the delivery probabilities for links $a \rightarrow b$ and $b \rightarrow a$, α is an adjustable weight between 0.5 and 1, and IF_a and IF_b are *Interference Factors* for nodes a and b, respectively. The interference factor for a node is defined by Ashraf *et al.* as the percentage of time a node sees the medium occupied (either due to a reception or because it is reserved for another node).

The first factor in the expression of the ELP metric is similar to the ETX of the link. However, instead of multiplying the delivery probabilities in each direction, Ashraf *et al.* use a weighted average between these values. The rationale for this approach is that data packets are usually much larger than the acknowledgments, resulting in a higher loss probability. For this reason, authors consider that the estimates for the delivery probability obtained through probing are more representative for data packets than for ACK frames. Consequently, they argue that, by selecting an appropriate value for parameter α, it is possible to obtain a more precise estimate for the probability of success of a unicast transmission than by simply multiplying the delivery probabilities in each direction.

Therefore, according to the authors, the meaning of the first factor is the same of the ETX metric: the expected number of transmission attempts necessary until the first success in a unicast frame transmission. Nevertheless, this value is weighted by the second factor which is a monotonically increasing function on $Max(IF_a, IF_b)$ (the maximum value between the interference factors of nodes a and b). The idea is that, if link $a \rightarrow b$ is currently being heavily interfered by other network links, values IF_a and IF_b will be high. In that case, the cost of the ELP metric increases, even if its ETX remains low.

There are a number of issues with the ELP metric. Firstly, authors do not approach the issue of the existence of multiple transmission rates in the link layer. By using ETX as

a base for their metric (instead of ETT, for instance), their proposal completely disregards possible differences in links in terms of transmission rate. Moreover, similarly to iAWARE, ELP uses information that is directly affected by the route selection. For instance, consider a link $a \rightarrow b$ that belongs to a path that is selected by ELP. Assume that prior to the selection of that path, IF_a and IF_b were very close to zero (*i.e.*, there was almost no network traffic near those nodes). Once a flow starts to send traffic through the path, IF_b is rapidly increased due to the usage of link $a \rightarrow b$, which would directly affect the cost of the link. Authors do not discuss the implications of such sensitivity in the route selection stability. Moreover, notice that if the network does not have active flows, the interference factors for all nodes are very close to zero. Since the interference factor is multiplied by ETX, the precision of the representation used for metric ELP becomes an issue: a low precision representation could result in most links being evaluated as being equally good, possibly leading to bad routing choices.

2.3 Coding Aware Routing

The concept of network coding was introduced by Ahlswede *et al.* [1]. In that work, authors define the network coding paradigm and study the Maximum Flow problem for a single data source. Authors argue that the traditional vision of information as fluid flowing through the network is not valid in the general case. That happens because, contrary to fluids, that can only be routed in the intermediate nodes, information can also be replicated and combined. That opens a range of new possibilities for data networks.

The paper shows a series of example networks in which the theoretical maximum flow given by the Max-Flow Min-Cut theorem [29] cannot be reached by employing the traditional routing paradigm. In those same examples, however, it is possible to achieve the theoretical limit by using network coding strategies.

Consider, for instance, the network illustrated in Figure 2.2a. This topology is commonly known in the literature as the *Butterfly Network* [43] and is frequently used to illustrate network coding on a wired network. The value associated with each link shows the number of bits it is able to transmit per unit of time. The maximum flow between nodes s and t_1, according to the theorem, is 2 (the lowest capacity among all s-t cuts). The same is valid for nodes s and t_2. In fact, considering a unicast flow directed at only one of the destination nodes (for instance, t_1), it is possible to send two simultaneous bits. One possible transmission order to achieve this maximum flow is illustrated in Figures 2.2b

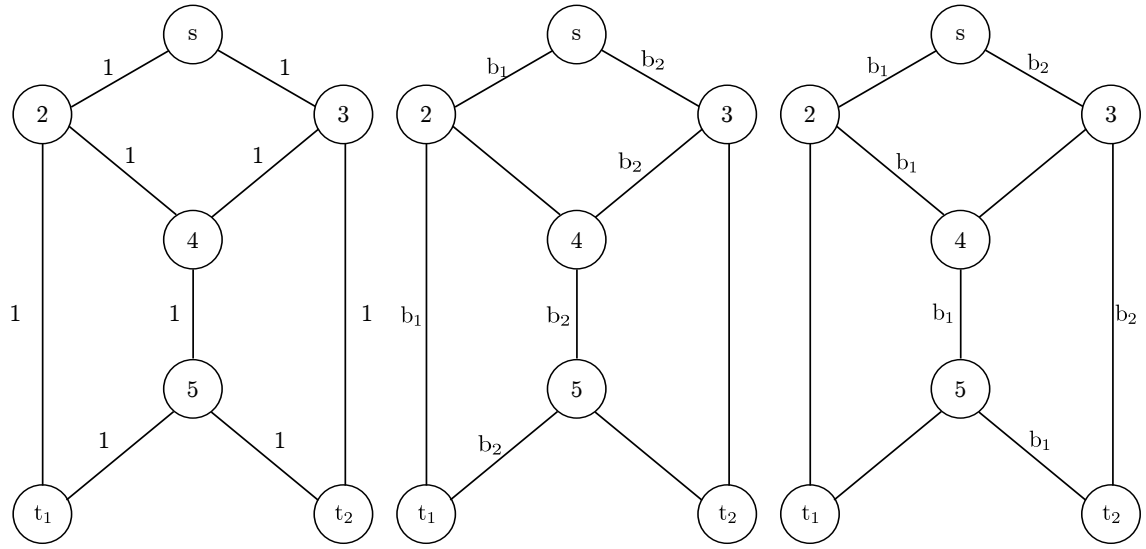


Figure 2.2: Maximum flow between a source node and a single destination in the Butterfly Network. The leftmost figure shows the capacity of each link, while the other two show the transmissions necessary to achieve the maximum theoretical flow (which is 2).

and 2.2c (for destinations t_1 and t_2, respectively).

However, when one considers a multicast flow addressed to nodes t_1 and t_2 simultaneously, it is not possible to achieve the maximum flow of two bits for both using the traditional routing paradigm. Figure 2.3a illustrates an attempt to do so. Indeed, with the transmission schedule illustrated in the figure, one of the nodes receives only half of the flow (1 bit). In order to achieve a flow of two bits for both destinations simultaneously, it is necessary that nodes 2 and 3 received distinct bits from the source. Hence, links $2 \rightarrow t_1$ and $3 \rightarrow t_2$ necessarily carry distinct bits. Therefore, the missing bit for each destination must come from the nodes in the middle of the network (nodes 4 and 5). Nevertheless, since there is an intersection between the paths $3 \rightarrow 4 \rightarrow 5 \rightarrow t_1$ and $2 \rightarrow 4 \rightarrow 5 \rightarrow t_2$, only one bit per time can be sent. In conclusion, it is impossible for both destinations to receive two bits simultaneously.

This limitation can be overcome by the employment of network coding, as shown in Figure 2.3b. The scheduling of the transmissions in each link is similar to that of the previous example. A notable difference is in the bit transported by links $4 \rightarrow 5$, $5 \rightarrow t_1$ and $5 \rightarrow t_2$. When node 4 receives bits b_1 and b_2, they are combined using a XOR operation. The result is transmitted to node 5, which in turn replicates it to both nodes t_1 and t_2. The destination nodes, then, recover their respective necessary bits by using another XOR operation (using the bit they received from the other path).

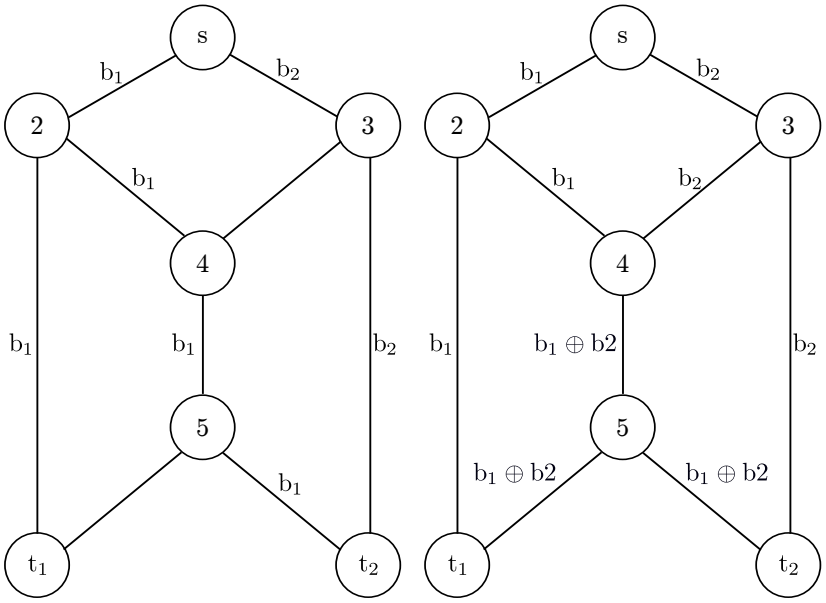


Figure 2.3: Maximum flows achieved in the Butterfly Network with and without network coding.

2.3.1 Applications of Network Coding

Although the first efforts in the area of network coding were very theoretical, there is now a vast literature on concrete solutions and studies on practical implementation issues [43]. In the next sections some of this work will be discussed.

2.3.1.1 Unicast Flows

The initial efforts in the network coding area focused mostly in multicast transmissions [44, 42, 20, 105, 34, 41, 97, 98, 99]. The reason for that is the fact that, at first glance, it is not obvious how to obtain gains from the network coding paradigm for unicast flows. In terms of maximum flow, for example, it is a known fact that network coding does not result in gains for unicast flows [1, 64, 63].

There are, however, papers that explore the usage of random network coding for unicast traffic in lossy networks [64, 65, 66]. In those papers, authors demonstrate that the usage of random network coding can arbitrarily decrease the message loss probability for a unicast flow with the increase in the number of coded packets.

Unicast flows can also take advantage from network coding when one considers multiple concurrent flows. That is illustrated in the example from Figure 1.3, presented in Chapter 1. In this example, two concurrent flows pass, on different directions, through

three nodes in sequence, 0, 1, and 2. By forcing that nodes 0 and 2 keep their original packets in memory after transmitting them, node 1 can code packets from each flow together, obtaining a single message to be transmitted, as explained in Chapter 1.

It is interesting to notice, however, that the gains resultant from network coding in this case are global, instead of local. For instance, if we consider the delay and throughput of the flow from node 0 to node 2, assuming there are no other concurrent flows, there is no gain from the usage of network coding. The gain is only perceivable when we consider the aggregated throughput of both flows. Generally speaking, the gain obtained with network coding can be measured as the ratio between the number of transmissions made by all flows with and with and without network coding [56]. In the example of Figure 1.3, for instance, this gain is $\frac{4}{3} = 1.\bar{3}$.

2.3.1.2 Multihop Wireless Networks

In the specific case of multihop wireless networks (*e.g.*, ad hoc networks, wireless mesh networks), the potential gains of network coding are promising. The diffusion nature of the wireless medium results in a series of possible scenarios with coding opportunities. Besides the case illustrated in Chapter 1, other coding opportunities may arise from the promiscuous reception of frames by nodes [56].

For instance, consider the topology illustrated in Figure 2.4a. In this topology, known in the literature as the *cross topology* [56], there are two flows: $3 \Rightarrow 4$ and $2 \Rightarrow 0$. Both flows use central node 1 as a relay. It is assumed that all transmissions made by node 3 can be received by both nodes 1 and 0. Similarly, transmissions by node 2 reach both nodes 1 and 4.

In this topology, there is a coding opportunity at node 1. This node can code together packets from each flow (using an bitwise XOR operation, for example) and forward both of them in a single broadcast transmission of the coded packet. Assume, for instance, that node 1 has two packets in its buffer: packet $P_{3\Rightarrow4}$ (from flow $3 \Rightarrow 4$) and packet $P_{2\Rightarrow0}$ (from flow $2 \Rightarrow 0$). Since node 4 can overhear transmissions by node 2, it could have stored packet $P_{2\Rightarrow0}$ when the latter was transmitted from node 2 to node 1. The same applies to node 0 and packet $P_{3\Rightarrow4}$. Hence, if node 1 codes both packets together and transmits the results (packet $P_{3\Rightarrow4} \oplus P_{2\Rightarrow0}$), both nodes 0 and 4 could decode it and obtain their respective packets. The coding gain in this example is $4/3 = 1.\bar{3}$.

A similar approach can be applied to the scenario illustrated in Figure 2.4b. In the

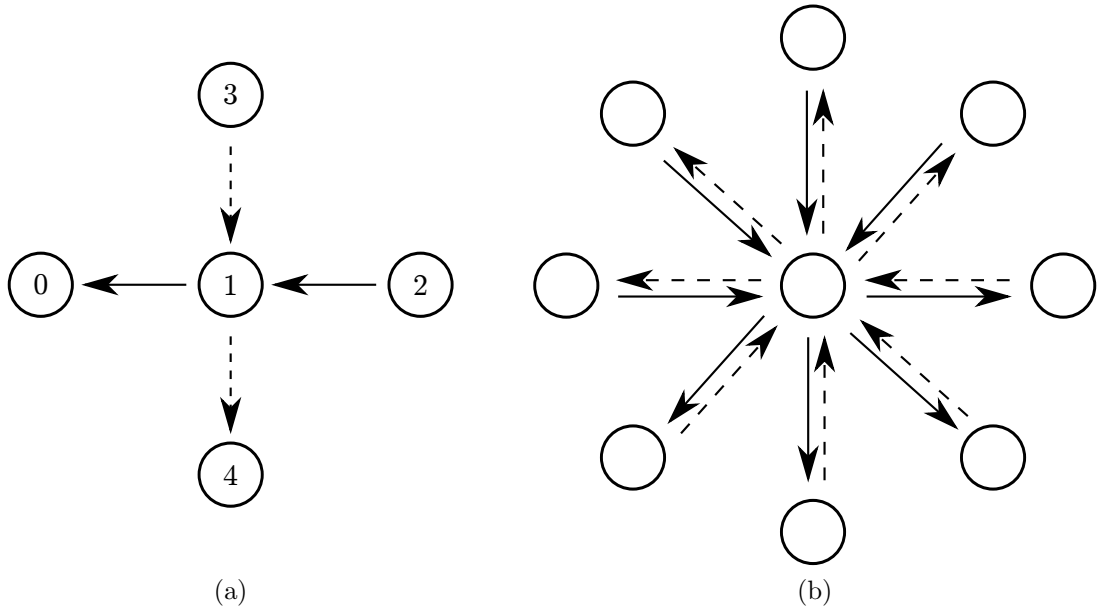


Figure 2.4: Examples of wireless network scenarios with coding opportunities.

topology, known in the literature as *wheel topology* [56], there is a central node that is used as a relay for the communication of n other nodes (for any even value n) forming a circle around it. Each of the n nodes in the circle communicates with its diametrically opposite counterpart. For each pair of diametrically opposing nodes, there are always flows in both directions. In this topology, it is assumed that, when a node transmits, all other nodes are able to receive the frame, except for its diametrically opposite counterpart (the central node can be heard by all other nodes).

This scenario presents the following coding opportunity. Assume that each of the n nodes in the circle transmits the first packet of the flow it sources. After these n initial transmissions, the central node has in its buffer the first packet of all flows. Similarly, nodes from the circle have overheard all packets, except for the one actually addressed to them. Therefore, the central node can code all packets together (again, using a simple XOR) and transmit the resultant coded message in broadcast. Assuming all nodes in the circle receive this message, they can decode it by simply executing the XOR of the coded message with each of the $n-1$ packets previously overheard during their neighbors' transmissions.

Without network coding, it takes a total of $2n$ transmissions to accomplish all communications (considering a single packet per flow). With network coding, this number drops to $n+1$ transmissions. As n increases, the network coding gains arbitrarily approach 2. Katti *et al.* also notice another benefit from the usage of network coding in this sce-

nario [56]. Assuming the link layer provides an approximately fair share of transmission opportunities among nodes, the central node receives a number of transmission opportunities comparable to that of the other nodes. However, since the central node acts as a relay for all network flows, it actually needs n times more transmission opportunities than its peers. If that is not the case, this node becomes a bottleneck for network performance. By using network coding as described, the central node ceases to be a bottleneck, since each of its transmission opportunities is equivalent to n transmissions without coding.

Based on this line of thought, Katti *et al.* define another metric to evaluate coding gains, called *Coding+MAC Gain*. This metric is defined as the ratio between the number of packets removed from the queue of the bottleneck node in each transmission opportunity with and without network coding. In the example of the wheel topology, this gain is $\frac{n}{1} = n$. In this case, as the number of nodes around the central node increases, the Coding+MAC gain also increases arbitrarily [56]. However, in [60], Le *et al.* demonstrate that, in practice, this scenario is not feasible for any value of n. Assuming signal propagation has euclidean geometrical properties, authors show that the maximum number of packets coded by the central node is limited by the expression:

$$L = \frac{\pi}{arccos(r/(r+\delta))} \tag{2.7}$$

In the expression, L is the maximum number of packets coded by the central node (this number is based on the number of packets that can actually be decoded by the destination nodes). The value r represents the distance between each node in the circle and the central node (in other words, it is the radius of the circle). Finally, δ is defined by the authors as the difference between the circle radius and the distance at which the reception probability of a frame becomes too low. In other words, authors assume that a link between two nodes distanced by at most r has a "good" delivery success probability. On the other hand, when the distance is larger than r, the delivery success probability of the link becomes too low, according to some predefined criterion.

In any case, the broadcast nature of the wireless medium makes it very favorable to the network coding paradigm. In a multihop wireless network, it is usually possible to identify several subgraphs homeomorph to the graphs of the topologies of the previous examples. Depending on the network flows, coding opportunities can be common.

The question, in this case, is exactly how to identify these opportunities. In several applications of network coding, it is reasonable to assume that the topology, as well as the

data flows, are static (at least during a sufficiently long interval). Hence, a preliminary setup of the rules for each node to choose which packets to code is feasible. On multihop wireless networks, though, that may not be the case, since the network topology is highly dynamic, both due to nodes' movements and due to instability in links' quality [78]. The flows in this kind of network can also be short, resulting in frequent alterations in the set of active network flows.

In order to make network coding feasible in this kind of scenario, an alternative form of coding, known as *Opportunistic Network Coding*, was proposed in [56]. The basic idea is somewhat simple: instead of trying to establishing a predefined static set of rules for each node to choose which packets to code, an algorithm is used to automatically and dynamically detect coding opportunities. Each time a node gains control of the wireless medium to transmit, this algorithm is executed to select the packet to be transmitted. The node's buffer is traversed in search of sets of packets that can be combined. A set of packets can be combined only if all respective destinations have a high probability of being able to decode the potential coded packet.

The main proposal in this area was presented by Katti *et al.*, in [55, 56]. In this work, authors propose COPE, a complete architecture for implementing opportunistic network coding in multihop wireless networks. In the paper, authors approach a series of practical issues on the implementation of this kind of network coding. However, the main contribution of the paper is the algorithm for deciding the set of packets to be coded at each transmission opportunity by a node.

The packet selection procedure used by COPE is described in Algorithm 2.1. Among the variables manipulated by the algorithm, there are two sets: $Native$, the set of native packets that are coded during the procedure, and $NextHops$, the set of destination nodes for the resultant coded packet (the next hops for the native packets).

Aside from the actual packet queue, the node must organize the packets in its buffer in other auxiliary data structures. Specifically, a node must keep two queues for each neighbor: one for small packets (defined by Katti *et al.* as packets of 100 bytes or less), and another for large packets (of more than 100 bytes). In Algorithm 2.1, this auxiliary queues are accessed using the notation $Q(i, queue)$, where i represents a neighbor and $queue$ represents the queue of small packets (for $queue = 0$) or large packets (for $queue = 1$). One last data structure stored by each node is the *Packet Pool*, a buffer of known packets stored during a predefined period of time to be possibly used in future decodings. Whenever a node forwards a packet or overhears a transmission made by a

Algorithm 2.1 Packet selection algorithm in the COPE architecture [56]

1: Choose the first packet P in the node's buffer.
2: $Native \leftarrow \{P\}$
3: $NextHops \leftarrow \{NextHop(P)\}$
4: **if** $size(P) > 100$ bytes **then**
5: $\quad queue \leftarrow 1$
6: **else**
7: $\quad queue \leftarrow 0$
8: **end if**
9: **for** $Neighbor\ i \leftarrow 1$ until M **do**
10: $\quad$ Choose packet P_i from the beginning of virtual queue $Q(i, queue)$.
11: $\quad$ **if** $\forall n \in (NestHops \cup \{i\}), p[\text{n can decode } P \oplus P_i] \geq G$ **then**
12: $\quad\quad P \leftarrow P \oplus P_i$
13: $\quad\quad Native \leftarrow Native \cup \{P_i\}$
14: $\quad\quad NextHops \leftarrow NextHops \cup \{i\}$
15: $\quad$ **end if**
16: **end for**
17: $queue \leftarrow (queue + 1)\ mod\ 2$
18: **for** $Neighbor\ i \leftarrow 1$ until M **do**
19: $\quad$ Choose packet P_i from the beginning of virtual queue $Q(i, queue)$.
20: $\quad$ **if** $\forall n \in (NextHops \cup \{i\}), p[\text{n can decode } P \oplus P_i] \geq G$ **then**
21: $\quad\quad P \leftarrow P \oplus P_i$
22: $\quad\quad Native \leftarrow Native \cup \{P_i\}$
23: $\quad\quad NextHops \leftarrow NextHops \cup \{i\}$
24: $\quad$ **end if**
25: **end for**
26: Return p.

neighbor, it stores the packet in the packet pool so that it can be used in the future, if needed.

The algorithm always chooses the first packet of the actual packet queue for transmitting (line 1). Afterwards, it identifies the size class to which the packet belongs (lines 4 to 8). Initially, only packets from the same class are considered for coding. For each of the M neighbors, the algorithm verifies if the first packet of the respective queue, p_i, can be coded with the other already selected packets in the set $Native$ (lines 9 to 16). To make this decision, the algorithm supposes the packet will be coded and checks if all nodes from the set $NextHop$ (including the possible next hop i) have a high probability of being able to decode the resultant coded message (line 11).

Katti *et al.* define the probability of a node being able to decode a coded message p_D composed of n packets (besides the desired packet) as:

$$p_D = p_1 \times p_2 \times \cdots \times p_n, \tag{2.8}$$

where $p_1, p_2, \ldots, p_n$ denote the probabilities of the node having already received each of the n packets. To compute the probability of a node having received a given packet, Katti *et al.* suggest that two cases must be accounted for. The first happens when a node was used as a previous hop for the packet. In this case, the probability is considered to be 1. Otherwise (*i.e.*, in the second case), COPE assumes the existence of a routing protocol capable of providing packet delivery probability information for all links. In this case, COPE assumes that the only way the node could know the packet is if it has overheard a previous transmission. Specifically, COPE considers only the last transmission of the packet, evaluating the link delivery probability between the last hop and the node in question.

COPE defines a constant value G for the lowest acceptable decoding probability (in [56], authors suggest a default value of $G = 0.8$). When evaluating the feasibility of adding a native packet P_i (addressed to neighbor i) to a coded message, if the decoding probability for the potential coded message is below G for any of the potential receivers (members of the set $NextHop \cup \{i\}$), packet P_i is considered unfeasible and disregarded for coding (at least, for this transmission). Otherwise, the algorithm codes P_i with the other already selected packets, adds it to the $Native$ set, and adds i to the $NextHop$ set.

The option for giving priority for coding packets of the same size class seeks to maximize the coding gains. When two packets with very different sizes are coded together, the resultant message becomes much larger than the smaller native packet (the resultant size will always be that of the largest native packet). Therefore, the gains obtained by this type of coding will not be as expressive for the smaller packet as they would, had it been coded with another small packet. On the other hand, coding packets of different sizes still results in gains (*i.e.*, the total time needed to transmit the complete set of native packets is larger than the time needed to transmit a single coded packet, regardless of their sizes). For this reason, once all possible coding partners of similar size have been evaluated, COPE repeats the process for the other size class (lines 18 to 25).

Katti *et al.* state that this algorithm is scalable because its asymptotic time complexity is linear with respect to the number of neighbors of the node that executes it. However, in the worst case (when all neighbors have a packet selected for coding), the complexity of the algorithm becomes a cubic function of the number of neighbors (the conditions tested at lines 11 and 20 are, actually, two nested repetitions dependent of the number of neighbors due to the usage of the operator *for all* and the nature of the computation of the decoding probability). In any case, assuming the number of neighbors

of a node in a multihop wireless network is typically small (*e.g.*, not larger than 20), the practical time complexity of this algorithm is not very relevant.

One important detail is that the algorithm used by COPE is not optimal. This sub-optimality is due to the fact that, at a given execution, it only evaluates the first packet of each virtual queue for each neighbor. It is possible, however, that better native packet sets feasible for coding could be found if the algorithm would check all packets in the buffer. On the other hand, an algorithm based on an exhaustive search would certainly result in a higher complexity, possibly causing the processing overhead to become an issue.

In the experiments shown by Katti *et al.* in [55, 56], authors report gains of 70% for TCP flows and up to 4 times for UDP flows, in terms of aggregated throughput, with the usage of COPE (with respect to the same scenario without the usage of the architecture). The experiments were performed in a real wireless mesh network, which brings further confidence in the feasibility of the employment of network coding in practice.

One limitation of the work by Katti *et al.* is the absence of a modeling that considers multiple transmission rates. Authors do not approach this issue with their proposal and, during their experiments, network nodes had their transmission rates fixed at 6 Mb/s, both for experiments with COPE and without it. Hence, the gains reported by Katti *et al.* show the superiority of COPE in a network without automatic rate adaptation. Authors do not discuss whether their proposal would result in gains if compared with a network without network coding but capable of using higher transmission rates.

2.3.2 Coding Aware Routing Protocols and Metrics

After the work by Katti *et al.* [56], the usage of network coding in multihop wireless networks became more feasible. The results reported by that work have shown that this paradigm can, indeed, result in considerable gains in terms of capacity in this kind of network. This has motivated the appearance of a new line of research related to network coding: the coding aware routing protocols.

Although COPE results in high gains in terms of aggregated throughput, it just explores coding opportunities that arise in the network. In other words, the coding opportunities explored by COPE are random (from COPE's point of view).

There are two factors that determine the occurrence of coding opportunities in a network: the traffic demands and routes used by each flow. In general, there is no way to control the traffic demands. They result from the usage patterns of the network by the

users. However, the route used for each network flow is determined by the choices made by the routing protocol.

In the work by Katti *et al.*, the routing protocol is treated as a completely orthogonal mechanism, being totally unaware of the existence of COPE. However, if the routing protocol was aware of the coding capability of the network, it could manipulate its routing choices in order to force flows to "cross" in certain regions of the network topology, thus provoking more coding opportunities. With more coding opportunities, one could expect the gains obtained by opportunistic network coding mechanisms to be improved.

Routing protocols usually choose routes based on traditional performance metrics, such as the loss probability or the end-to-end delay [17]. The manipulation of routes with the objective of increasing the number of coding opportunities has the side effect of resulting in suboptimal choices according to these traditional metrics. Hence, the challenge of this kind of protocol is to guarantee a reasonable trade-off between the traditional routing metrics and coding opportunities.

In [87], Sengupta *et al.* perform a theoretical evaluation of the gains obtained by opportunistic network coding methods for multihop wireless networks, such as COPE. The authors describe a mathematical model for such networks, in the form of a linear programming problem with the goal of maximizing the aggregated throughput. Through this analysis, authors conclude that routing choices have a relevant impact in the performance of opportunistic network coding.

One of the issues with the evaluation performed by Sengupta *et al.* is the simplicity of the model of the medium access layer considered in this work. This model assumes, for instance, that two neighbors can transmit simultaneously granted that there is no intersection between the sets of receivers for each transmission and that the receivers of one transmitter are not within the interference range of the other. In medium access protocols based on the CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) technique, such as the ones found in the many versions of the IEEE 802.11 standard [49], two neighbors should not transmit simultaneously, even under those conditions.

Although one could try to use the linear programming model proposed by Sengupta *et al.* to find the best routes for a given network (according to their model), this is not practical. The first reason is the fact that the model assumes the knowledge of many parameters regarding the network itself and the data flow demands. Another issue is the relatively high complexity of the model, which demands high computational power and long execution times to be solved.

In [61], authors present a proposal of a practical coding aware routing protocol. The protocol, referred to as DCAR (Distributed Coding-Aware Routing), is a variation of the traditional DSR (Dynamic Source Routing) routing protocol [52].

When a new flow is detected, a process of route discovery is started with the broadcast of a RREQ (Route Request) packet. Upon receiving such a packet, a node includes in it a list of its "good" neighbors (*i.e.*, neighbors for which it has a link with a high delivery probability). Authors suggest that a good link could be defined as one with a delivery probability higher or equal to 0.8 (not coincidentally, the default value for the constant G used by COPE). The motivation for including this list in the RREQ packet is that these neighbors would be able (with high probability) to overhear transmissions of this flow (assuming the node is actually chosen as part of the route), possibly allowing network coding to happen in the next hop of the path. Another information added to the RREQ is the node identifier, stored so that the path traversed by the RREQ is known at the end of the process.

Eventually, one would expect the destination node for the new flow to receive one or more RREQ packets. For each received RREQ, the destination generates an RREP (Route Reply) packet, transmitted in unicast by the reverse of the path traversed by the RREQ. The RREP contains the list of nodes of the found path. When an RREP is received by an intermediate node, it calculates a routing metric defined by the authors, called CRM (Coding-aware Routing Metric). The CRM metric of a link l is defined as:

$$CRM_l = \frac{1 + MIQ_d(l)}{1 - p_l} \tag{2.9}$$

On this equation, p_l is the packet loss probability for the link l. The reciprocal of $1 - p_l$ is an estimate for the expected number of retransmissions needed until the first successful transmission of a frame through link l. The term $MIQ_d(l)$ is defined by Le *et al.* [61] as the sum of the current sizes of the queues for all nodes that belong to the same collision domain as the link l, considering possible coding opportunities with the new flow.

The idea of the authors is to consider the queue size for each node as part of the routing metric. When a node computes the metric for a link, it checks which packets currently in its buffer could be coded with packets from the new flow. Packets that could be coded with a packet from the new flow are decremented from the current queue size, since one can consider their transmission as being "free", given that it would be performed

along with the transmission of a packet from the new flow. The node must also consider the coding opportunities among other packets currently in the buffer, reducing even more the effective queue length. The result of this computation is an estimate for the number of transmissions needed to empty the node's current buffer. The sum of this value for all nodes belonging to the same collision domain results in the value $MIQ_d(l)$.

Once the receiving node of an RREP packet has computed the CRM metric for its link, it adds this information to the packet and transmits it to the next hop of the reverse path. Eventually, the RREP packet is received by the source of the new flow. After waiting for a predefined amount of time in order to possibly receive multiple RREPs, the source node chooses the path with the lowest sum of the CRM metric for all its links.

There are a number of issues with the proposal presented by Le *et al.* The authors present a method for computing the value $MIQ_d(l)$. This method builds a graph in which each vertex represents a flow that currently traverses the node, while each edge indicates that packets from two flows (vertices) can be coded together. Authors argue that the best set of flows to be coded together is represented by the maximum clique in the graph. However, they state that finding such cliques would not be feasible in practice, since the maximum clique problem is NP-hard. For this reason, Le *et al.* propose an approximative method for computing $MIQ_d(l)$. The problem with this method is that it involves the solution of a sub-problem that, by itself, it is also NP-Hard [32]. Authors argue that the maximum number of flows that can be coded, in practice, is small, which would render the computational cost of the proposed approximative method irrelevant. It is not clear, however, why the same argument is not valid for the exact method (based on solving the clique problem).

The approximative method also has a serious flaw. Authors consider that packets from flows with edges for the new flow can be all subtracted from the computation of MIQ_d, even if this set of flows do not form a clique. However, if a clique is not formed, one can not code all the packets together because the resultant message would not be decodable by the receivers. Only a subset of flows (for which a clique is indeed formed) should be chosen. Therefore, the value MIQ_d could be lower than what it should represent.

Le *et al.* also do not take into account the possibility of multiple transmission rates being available at the link layer. The model presented by authors does not deal with this issue at any point. Although not explicitly indicated by the authors, simulation results presented in this work suggest that the transmission rate for the nodes was fixed at 1 Mb/s.

Finally, there is the issue of the reception probability for a coded packet. The CRM message uses only the individual reception probability for the link l, even if it considers the coding of many packets for different receivers.

A second proposal of a practical coding aware routing protocol is presented by Fan *et al.* in [30]. The authors propose a protocol called HLCR (Heuristic Load-balanced Coding-aware Routing), very similar to DCAR. Among the differences are a mechanism for controlling the flooding of RREQ messages and the routing metric.

The proposed routing metric is called WETCC (Weighted Expected Transmission Count with Coding). Like CRM, WETCC takes into account three aspects: the load of the nodes composing a path, the link quality and network coding opportunities. The load of a node is represented by an expression proposed by the authors called *Activity Level* (AL) defined for a node a as:

$$AL(a) = 1 + \frac{QL(a) \times |N_a|}{\sum_{b \in N_a} QL(b)}, \tag{2.10}$$

where $QL(a)$ represents the length of the node's queue and N_a represents the set of neighbors of a. Intuitively, the value $AL(a)$ represents the current queue length of a in comparison to the average size of the neighbors' queues.

Under those conditions, the WETCC metric for a link $a \rightarrow b$ is defined by the authors using the following expression:

$$WETCC_{a \rightarrow b} = \begin{cases} AL(a) \times (ETX_{a \rightarrow b} - min(ETX_{a \rightarrow b}, ETX_{a \rightarrow c} \times d_{a \rightarrow b})) & \text{if } a \rightarrow b \text{ is a coding link} \\ AL(a) \times ETX_{a \rightarrow b} & \text{otherwise} \end{cases} \tag{2.11}$$

In this expression, $ETX_{a \rightarrow b}$ denotes the value of the Expected Transmission Count metric for the link $a \rightarrow b$. The value $d_{a \rightarrow b}$ represents the delivery probability of the link $a \rightarrow b$. Link $a \rightarrow b$ is considered a coding link if there is another link $a \rightarrow c$ from node a used in the path of a previously established flow such that the packets for the currently evaluated flow can be coded together.

The meaning of the expression for the WETCC metric can be explained as follows. If $a \rightarrow b$ is not a coding link, (*i.e.*, if the new flow passing through this link cannot be coded

with other flows), the cost is simply the product of the node load ($AL(a)$) and the quality of the link ($ETX_{a\rightarrow b}$). In this case, there are no coding effects to be considered. On the other hand, if $a \rightarrow b$ is a coding link, it is possible that not all packets of the new flow will need transmissions, because some of them will possibly be sent coded with packets from other flows. To quantify how many transmissions will be saved due to network coding, WETCC multiplies the ETX of the link $a \rightarrow c$ (*i.e.*, the number of transmissions, on average, used for sending each packet through the link $a \rightarrow c$) by the probability that each of the transmissions is successfully received by node b. This value is then subtracted from the ETX of the link $a \rightarrow b$. However, if the link $a \rightarrow c$ is much worse than link $a \rightarrow b$, it is possible that the resultant value is negative. In this case, WETCC considers that the amount of transmissions needed for the new flow is 0.

When the RREP packets traverse the inverse path towards the source of the new flow, each intermediate node computes the WETCC metric for its link. As with the CRM metric, the total cost of a path in the WETCC metric is given by the sum of the costs of the component links.

A disadvantage of the WETCC metric with respect to the CRM metric is the fact that the former does not consider coding opportunities involving more than two flows. Fan *et al.* assume in their work that packets are transmitted natively (*i.e.*, without coding) or coded with a single other packet. This assumption limits somewhat the scope of the WETCC metric.

A deficiency common to both CRM and WETCC is the fact that neither takes into account the arrival rate of flows to be coded together at a given node. For instance, suppose that a given flow f_1 uses some node a as a relay and the routing protocol wishes to evaluate the cost of a route using node a for a given flow f_2. In this case, the CRM metric always considers that there will be pairs of packets from flows f_1 and f_2 to be coded together. On the other hand, metric WETCC evaluates the possibility of coding packets from f_1 with f_2 only based on the ETX of link $a \rightarrow c$ and the success probability $d_{a\rightarrow b}$. However, it is possible that the arrival rates for each flow at node a are considerably different due to different generation rates at the sources or available throughput in the subpaths leading to a. In any case, differences in those arrival rates result in imperfect pairing of the packets for coding and the necessity of sending some of the packets natively.

Ding *et al.* [27] propose a coding aware routing metric which has the goal of being applicable to traditional routing mechanisms (protocols and path selection algorithms). The metric is defined as the number of transmissions needed so that the packets of the

new flow are sent from the source to the destination. To this end, authors consider that network coding only occurs when two flows share a sequence of, at least, three nodes in their respective paths, traversed in opposite directions. In this case, considering the complete path between the source s and destination t, the reduction in the number of transmissions for a flow $s \Rightarrow t$ caused by the usage of network coding is defined by authors as:

$$R[\mathcal{P}_{s\Rightarrow t}] = \sum_{(i\rightarrow j\rightarrow k)\in\mathcal{P}_{s\Rightarrow t}} \frac{min(T_{s\Rightarrow t}, T(f_{k\rightarrow j\rightarrow i}))}{d_{i\rightarrow j} + d_{j\rightarrow k} - d_{j\rightarrow k}d_{i\rightarrow j}}. \tag{2.12}$$

In this equation, $\mathcal{P}_{s\Rightarrow t}$ denotes the path being evaluated, while i, j and k are any three consecutive nodes on this path. $T_{s\Rightarrow t}$ is defined by the authors as the amount of traffic from the new flow (for example, in a window of one second). $T(f_{k\rightarrow j\rightarrow i})$ denotes the amount of aggregated traffic that currently traverses the subpath $i \rightarrow j \rightarrow k$. The terms $d_{i\rightarrow j}$ and $d_{j\rightarrow k}$ denote, respectively, the frame delivery probabilities for links $i \rightarrow j$ and $j \rightarrow k$. Notice that in this work, Ding *et al.* consider all links to be symmetrical, *i.e.*, $d_{i\rightarrow j} = d_{j\rightarrow i}$ for any two nodes i and j.

Assuming that the events of receiving a packet through links $i \rightarrow j$ and $j \rightarrow k$ are independent, the denominator of the fraction presented in Equation (2.12) represents the probability that at least one of the destinations receive the coded packet. The reciprocal of this value is the expected number of retransmissions until the packet is received by at least one of the destinations. The numerator indicates the amount of traffic that can be coded at node i (with respect to flow $s \Rightarrow t$). The product of these values, therefore, is the expected amount of transmissions for a coded packet, but requiring only the reception by one of the destinations. Is not clear in [27] why the authors consider this equivalent to the reduction in the total number of transmissions for flow $s \Rightarrow t$ due to network coding.

In any case, given the value $R[\mathcal{P}_{s\Rightarrow t}]$, Ding *et al.* define the total cost of a path for a flow $s \Rightarrow t$ as:

$$w[\mathcal{P}_{s\Rightarrow t}] = \sum_{(i\rightarrow j)\in\mathcal{P}_{s\Rightarrow t}} \frac{T_{s\Rightarrow t}}{d_{i\rightarrow j}} - R[\mathcal{P}_{s\Rightarrow t}]. \tag{2.13}$$

In other words, the cost of a path for the flow is given by the total number of transmissions needed to send all packets for this flow without network coding, subtracted from the number of transmissions saved due to network coding.

An interesting contribution of this work is the adaptation of the proposed metric to

traditional routing mechanisms. Ding *et al.* argue that the proposed metric does not have the mathematical property of isotonicity. Therefore, traditional algorithms for the problem of shortest paths in graphs cannot guarantee the optimal solution.

In the specific case of the metric presented by Ding *et al.*, the lack of isotonicity results from the fact that the cost of adding a new link to a path varies according to whether or not the current node is intermediate. If a node is a terminal in the path of a flow, it cannot perform network coding with packets from this flow. On the other hand, if the node is intermediate, it might be able to code the flow, reducing its the contribution for the total cost of the path.

To avoid this problem, Ding *et al.* propose a scheme for mapping the graph of the real network topology to a graph of a virtual topology. In this virtual topology, each real node is mapped to many virtual nodes. For instance, a node a is mapped to, at least, two virtual nodes a^+ and a^-. In the virtual topology, node a^+ is used only for data flows originated at the real node a. Similarly, node a^- is responsible for data flows addressed to node a. When a is used as an intermediate node in a path, more virtual nodes are created. For each type of virtual node, authors present a set of rules to create edges with their respective costs.

The virtual topology graph created through this mapping process can be used as an input for a shortest path algorithm, such as Dijkstra's algorithm. In this case, it is possible to guarantee that the algorithm is able to find the optimal solutions for the problem.

A more recent proposal is the Coding-aware Path Transmission Time metric (CPTT), introduced by Yue *et al.* [104]. The idea of CPTT is to assign as the cost for a path the throughput of its bottleneck link. In order to evaluate the throughput of each link, Yue *et al.* employ the concept of conflict graph and use the cliques to estimate an upper bound. The possibility of coding is taken into account by extending the concepts of links and conflict graph into hyperlinks (links with single sources and multiple destinations) and hyperlink conflict graphs. For a given real link, the throughput is approximated as the maximum between its throughput upper bound and the maximum throughput of any hyperlink containing it.

Although Yue *et al.* show good results when comparing CPTT with other coding-aware metrics, authors do not address important practical aspects of their proposal. Namely, they do not explain how an algorithm for finding the best routes would work with their metric. CPTT is a non-isotonic metric and thus it requires non-traditional path finder algorithms or the employment of virtual graphs. Even more serious, though,

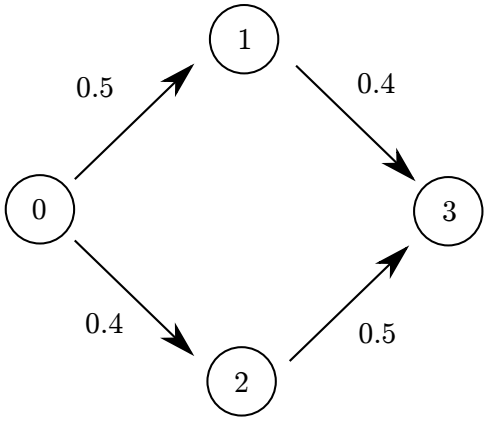


Figure 2.5: Example of a topology that may favor opportunistic routing.

is the lack of a complexity analysis for computing the costs for each link. This computation requires all maximal cliques to be found for each link in the conflict graph, a problem which is known to be NP-Hard [32].

2.3.2.1 Opportunistic Routing

Another branch of the coding aware routing protocols uses the concept of *Opportunistic Routing* [14]. In this kind of routing, the idea is to try to take advantage of the broadcast nature of the wireless medium to increase the delivery probability of the data packets.

Instead of determining a single node as the next hop for a given packet, an opportunistic routing protocol determines a set of neighbors that can efficiently act as relays for that packet, *i.e.*, a set of next hop candidates. When a node wishes to send a packet, it chooses a subset of nodes among its neighbors that are closer to the destination node than itself (according to some routing metric). The packet is then sent in broadcast and any neighbor that belongs to that subset may perform the forwarding of the packet. The probability that at least one of the candidates receive the packet and is able to continue the process of forwarding is always greater or equal to the probability that only the best one does it.

For instance, consider the topology illustrated in Figure 2.5. In this figure, the value associated with each edge shows the delivery probability for the respective link. Suppose node 0 wishes to send a packet to node 3. However, 0 is not able to reach 3 directly. It is necessary that one of the intermediate nodes 1 and 2 is used as a relay. In the traditional routing, node 0 would choose one of the intermediate nodes and send the packet to it. In the opportunistic routing, node 0 only includes nodes 1 and 2 among the

next hop candidates. In this case, the probability that at least one of the intermediate nodes receives the packet is $[1 - (1 - 0.5) \times (1 - 0.4)] = 70\%$.

Generally speaking, a transmitting node associates a forwarding priority with each next hop candidate. In other words, the closer a candidate is to the destination, the higher its priority to forward the packet. This priority is used so that lower quality candidates are only used if the better candidates do not receive the packet. In the case illustrated by Figure 2.5, the highest priority would be given to node 2, assuming the criterion was the link delivery probability.

Previous work, such as the one by Biswas and Morris [14], show that opportunistic routing can bring gains in terms of throughput, if compared with traditional routing. Specifically, in dense topologies with low quality links, opportunistic routing can result in increased throughput by reducing the packet loss probability (since a packet needs to be lost by all candidates to be considered definitely lost). There are, however, still open questions such as the coordination of the candidate nodes. In other words, how can one avoid that two or more candidates transmit the same packet redundantly? This situation can be illustrated in Figure 2.5 assuming, for instance, that nodes 1 and 2 are not within communication range from one another. In this case, even if node 2 has the forwarding priority over node 1, it is not possible to avoid node 1 from making a possibly useless transmission since this node is never able to detect that node 2 has already forwarded the packet.

The first proposal of an opportunistic coding aware routing protocol was MORE [18]. The idea of this proposal is to combine opportunistic routing with random network coding. The source node for a data flow divides the flow into blocks of k packets. The source then proceeds with the transmission of the first block, by sending multiple random linear combinations of the packets that compose the block. Each coded packet contains the information of the coefficients used for the combination and the list of the candidate forwarding nodes. However, differently from the traditional opportunistic routing, MORE does not associate priorities with each candidate. Whenever a node receives a packet and verifies it is among the forwarding candidates, it checks whether this received linear combination is independent from the others it has already received. If that is the case, the node generates another random linear combination and broadcasts it for its own set of forwarding candidates. The source only stops transmitting new linear combinations for the current block when the destination node sends a confirmation that the block was decoded successfully.

The main advantage of MORE with respect to the traditional opportunistic routing is the fact that this protocol does not need to deal with the priorities for the candidates. In traditional opportunistic routing protocols, such priorities were handled by modifying the MAC layer by imposing a fixed scheduling for using the wireless medium, which would reduce the spatial reuse of the spectrum [18].

Koutsonikolas *et al.* proposed an optimization for MORE (which could be adapted to any other similar mechanism), based on acknowledgment packets between intermediate nodes [58]. In the original proposal of MORE, Chachulski *et al.* [18] suggest that for each new linear combination received, an intermediate node (assuming it belongs to the set of forwarding candidates) should generate and transmit a number of new linear combinations that is a function of the delivery probability of this node to the nodes in its own set of candidates. Koutsonikolas *et al.* [58] argue that this strategy can result in an excessive amount of retransmissions, rendering the wireless medium unnecessarily busy. For this reason, authors suggest the usage of hop-by-hop ACKs.

The idea is that each candidate sends an ACK to the previous hop letting it know that it already has enough linear combinations (assuming that each block is composed of k packets, ideally an intermediate node should receive k independent combinations). When a node receives ACKs from all its forwarding candidates, it ceases to transmit.

The main contribution of the work by Koutsonikolas *et al.* however, is not the capability of candidates to ask the previous hop to stop transmitting. Besides this information, ACKs can also contain data regarding already received linear combination. By receiving back ACKs containing information regarding known combinations by each of its forwarding candidates, a node is able to generate and transmit more useful linear combinations, reducing the number of necessary transmissions. An interesting aspect of the scheme proposed by Koutsonikolas *et al.* is the fact that the ACKs do not need to contain the actual coefficients for all combinations received by a candidate. Instead, it is sufficient that a node includes only a vector z_a that satisfies the following property:

$$z_a \cdot v = 0, \forall v \in B, \tag{2.14}$$

where B denotes the set of all known linear combinations (represented by their respective coefficient vectors) and "$\cdot$" denotes the operation of dot product. The advantage of this method is the reduction of the overhead caused by the ACK. Instead of sending a potentially large set of coefficient vectors, a node only needs to send a single vector that satisfies this property.

Radunović *et al.* [84] also present a coding aware opportunistic routing protocol [84]. Similarly to MORE, this work uses random network coding. However, this proposal tries to better control the diffusion of the linear combinations of the packets for each block by using a system of credits. In this proposal, one credit corresponds to a permission to transmit one linear combination of a given block. When an intermediate node receives a credit referring to a given block, it becomes authorized to generate and send a new linear combination of the packets of the block. The proposed protocol uses the concept of back pressure to assign the credits of a block among network nodes.

One of the limitations of the work by Radunović *et al.* are the strong assumptions adopted by the authors to simplify the execution and analysis of the proposal. One such assumption is the existence of an independent communication channel between network nodes used to transfer credits. Authors assume this channel is ideal in the sense that there are no losses or significant delay. If losses are allowed in this channel, credits may be lost, which would possibly cause the number of linear combinations delivered to the destination node to be insufficient for decoding.

Although most proposals of coding aware opportunistic routing protocols employ random network coding, there is at least one exception in the literature. Yan *et al.* propose a protocol called CORE [101]. CORE uses hop-by-hop network coding with packets from different flows, similarly to COPE. The set of forwarding candidates is chosen according to the criterion of geographical proximity: all neighbors that are closer to the destination than the current node are included in the candidate set. Once the packet is received by a set of candidates, the priority selection is done based on the coding opportunities. To this end, each candidate calculates a backoff time which is inversely proportional to the number of packets that can be coded together with the received packet.

An interesting aspect of the CORE protocol is that it uses an algorithm for selecting packets to be coded together that is different from the one proposed with COPE. While COPE looks for coding opportunities only among the first packets of each virtual queues associated with each neighbor, with CORE this approach makes little sense, since in opportunistic routing there is not a single defined next hop for a given packet. By using opportunistic routing, each native packet has a set of receivers (all nodes from the forwarding candidate set). Hence, the algorithm for selecting packets for coding must take into account if all forwarding candidates of a native packet are able to decode a potential coded message. Incidentally, Yan *et al.* define that their algorithm searches the first k packets of the queue looking for coding opportunities.

Although authors do not present a formal description of their packet selection algorithm, they state that the proposed method has an asymptotic complexity of $O(2^k D)$, where D is the mean degree of the network nodes. Authors argue that, for practical purposes, this complexity is not much different from the complexity of the algorithm used by COPE, since it is parametrized by parameter k, which can be adjusted. However, the term 2^k grows very fast even for low values of k. Unfortunately, authors do not provide an evaluation of how the processing overhead affects the performance of the proposal. The evaluation presented in [101] is based only on simulations, which makes the presented results oblivious to this overhead.

2.4 Discussion

There is a large literature on traditional routing in multihop wireless networks. While protocols and metrics aware of interference or network coding exist, the traditional oblivious mechanisms (*i.e.*, those that do not consider neither interference or network coding) are still the most widely used in practice [68, 92, 69]. Most traditional metrics are based on ETX and ETT (which, itself, is based on ETX).

In terms of interference-aware metrics and protocols, most proposals assume the existence of multiple network interfaces in each node [28, 91, 90]. In this case, the problem becomes somewhat easier due to the fact that a metric may simply explore the diversity of channels in its paths. Assuming the available channels are non-interfering, this procedure effectively reduces the intra-flow interference.

Among the interference aware proposals that consider the scenario with a single channel, a common problem is the lack of a complete vision of the network considering all existent flows. Those proposals tend to consider flows individually, either by keeping the same route choices for older flows once new flows start [6], or by assuming flows exist, even when they do not, leading to an unnecessary usage of a worse path that is not going to avoid interference [102].

Based on the revision of the literature on the theme of coding aware routing protocols, it was also possible to identify some specific deficiencies which are common to many of the proposals. These deficiencies can be summarized in the following points:

1. over-simplified medium access models;

2. disregard for the arrival rate of packets from each flow to a coding node;

3. disregard for the MAC layer transmission rate;

4. impossibility or high delay for rerouting previously established flows in case of the appearance of new flows; and

5. assumption of independence or total dependence in the events of reception of a coded packet by many receivers.

The excessive simplicity of the models for the medium access is noticeable in the proposals that employ mathematical programming [87]. Generally speaking, these simplifications allow events that are impossible in practice, such as the transmission of two packets simultaneously by nodes that belong to the same collision domain.

The arrival rate for packets of different flows is commonly ignored by proposals of practical routing protocols [61, 30]. The metrics used by such protocols, in general, implicitly assume that two or more flows coded together always have packets available for coding. In other words, if there is a packet from a flow to be transmitted in a given moment in the buffer of a node, there are also correspondent packets from other flows to be coded together. This assumption is only true if the arrival rates for the different flows at the node match one another.

Another common issue for most of the proposals is the availability of multiple transmission rates in the link layer. The proposals usually assume that the transmission rate is fixed and equal for all nodes [61]. In practice, however, it is common that wireless network devices are able to operate at different transmission rates. Notice that part of the reason why transmission rates are not considered by those protocols and metrics is due to a difficulty regarding network coding itself. In a network that supports multiple rates, when a coded packet is to be transmitted in the link layer addressed to two or more neighbors, it is necessary to choose which transmission rate to use. This choice, however, may not be trivial, especially if the links associated with each intended receiver have very different qualities. While there are simplistic approaches, such as simply choosing the lowest transmission rate among those used for unicast transmissions for each of the receivers, the literature on network coding still lacks a deeper study in that regard.

Most of the practical proposals employ the framework of a reactive routing protocol [61]. Whenever a new flow is detected, the routing protocol triggers a process of route discovery. During this process, the existence of other flows transmitting packets through their previously established routes is taken into account for deciding the route for the new flow. However, those protocols are not capable of rerouting those previous flows in order

to find better overall routes, assuming one might exist. For this reason, the set of routes selected by such protocols might not be globally optimal.

Finally, the last identified point refers to the process of transmitting a coded packet for a set of receivers. The proposals found on the literature intrinsically assume the independence or complete dependence of the reception events in each of the receivers [61]. In practice, however, as shown in Appendix B, this assumption is not always valid [76, 74].

Chapter 3

Proposed Routing Algorithm

In this chapter we describe the core contribution of this thesis: a novel framework of routing algorithms that seek to find a set of paths that maximizes the aggregated network throughput taking into account the self-interference among flows and, possibly, resorting to the usage of network coding. From this general framework, we derive two specific routing algorithms: IAR and ICAR.

Algorithm 3.1 shows the general idea for our proposed framework. This algorithm is based on two auxiliary functions: *EstimateThroughput* and *GenerateCandidates*. The idea is to generate a list of candidate solutions (*i.e.*, a list of sets of paths for the currently active network flows) and loop through this list evaluating the resultant aggregated throughput for each candidate. This evaluation is done by the auxiliary function *EstimateThroughput*. After traversing the complete list of candidates, the candidate that yields the highest aggregated throughput is returned.

This algorithm receives as an input a graph $G = (V, E)$ representing the topology of the network, as well as a set F of the currently active flows. As will be further discussed in the following sections, the *EstimateThroughput* function requires the information of the delivery probability to be associated with each edge on G. We assume this information is provided by the routing protocol, as it would be for common routing metrics, such as ETX. We also assume, however, that these delivery probabilities are good estimates for the actual delivery probabilities of the data packets. Since data packets tend to be large – with sizes similar to the MTU (Maximum Transmission Unit) of the interface – a possible implementation is to use large probes for estimating such delivery probabilities.

In the next sections, we further detail each of those functions, as well as some project decisions that were made. Those sections are organized so that we first present an exact version of the algorithm and then present heuristic mechanisms to reduce the complexity

Algorithm 3.1 Framework for the proposed route selection algorithms.

```
1: function RouteSelection(Graph G, Flow set F)
2:     CandidateList ← GenerateCandidates(G, F)
3:     BestCandidate ← First element of CandidateList
4:     BestThroughput ← EstimateThroughput(G, F, BestCandidate)
5:     while CandidateList is not empty do
6:         NextCandidate ← Next element of CandidateList
7:         NextThroughput ← EstimateThroughput(G, F, NextCandidate)
8:         if NextThroughput > BestThroughput then
9:             BestCandidate ← NextCandidate
10:            BestThroughput ← NextThroughput
11:        end if
12:    end while
13:    Return BestCandidate.
14: end function
```

of the solution, allowing it to be employed for real time route selection.

3.1 Link State *versus* Distance Vector

An important decision when designing a routing solution is whether to deploy a protocol based on link state or distance vector. Distance vector protocols have the advantages of requiring less state information to be stored in each node and of avoiding flooding the network with control packets. Nodes only need to store information regarding the known routes and control traffic is only sent locally. The processing complexity for computing routes using distance vectors is also lower than in the link state case, due to the employment of a distributed version of the Bellman-Ford algorithm [11].

All these characteristics make the distance vector approach interesting, especially considering the implementation on resource constrained devices. Nevertheless, one advantage of the link state approach is the availability of a global network view for the routing algorithm. While maintaining this consistent network view is more expensive, it provides more information for the route selection algorithm. In the specific case of the proposal presented in this thesis, it requires this global network view due to the necessity of computing an estimate for the aggregated network throughput, as depicted in Algorithm 3.1. Moreover, differently from other approaches listed in Chapter 2 that choose routes in a more incremental way, the proposal presented in this thesis allows previously established flows to be rerouted easily, in case that leads to a better solution when a new flow is started. For these reasons, we assume the usage of a link state routing protocol.

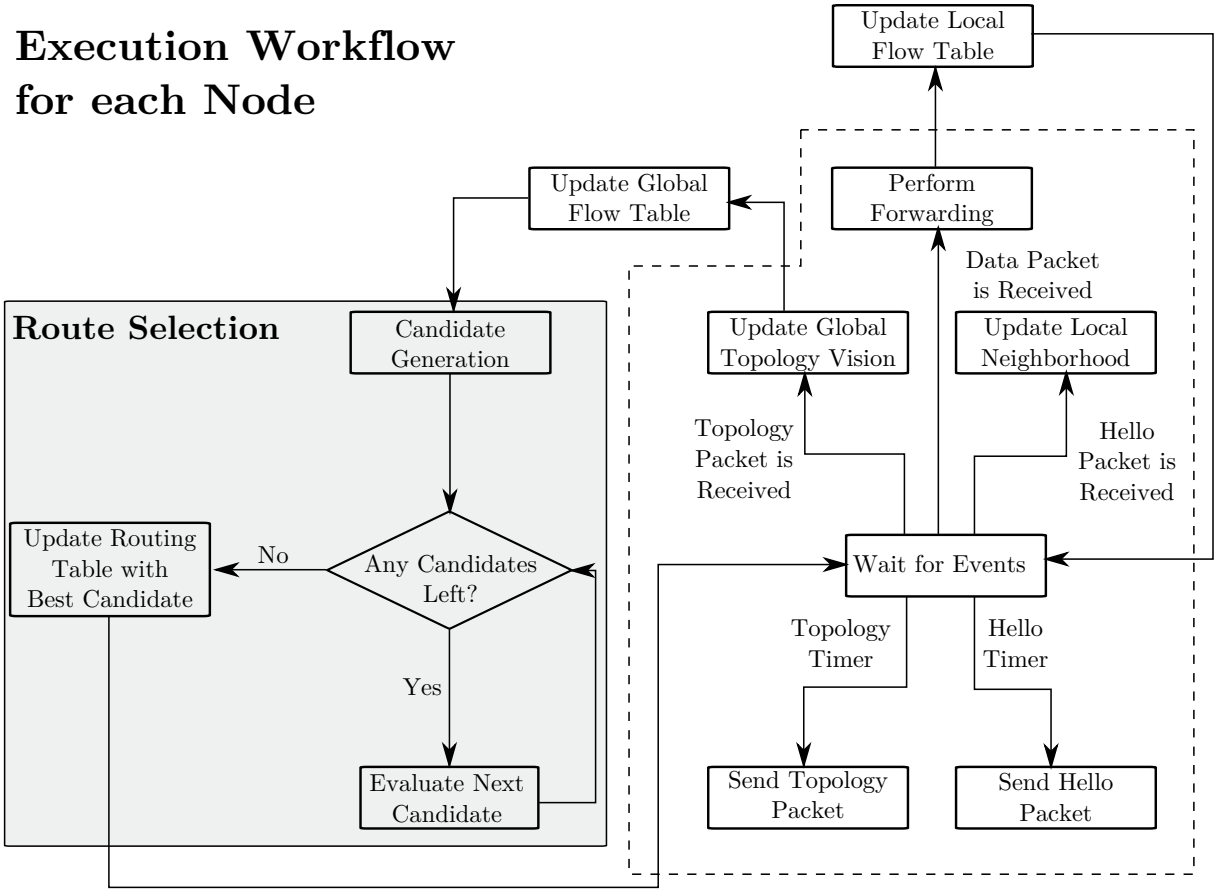


Figure 3.1: Proposed workflow for the execution of the link state routing protocol used with IAR and ICAR. The functions delimited by the dashed line represent basic functionalities found on any link state routing protocol. The gray box illustrates the routines concerning route selection.

Figure 3.1 illustrates the workflow for the routing protocol executed in each network node. All functionalities enclosed by the dashed line are typical routines for a link state routing protocol, such as maintaining local and global visions of the network topology and periodically sending control packets. To those basic functionalities, we also added routines for maintaining two flow tables: a local flow table (containing information regarding flows originated at the local node) and a global flow table (which stores information about all current network flows). The manipulation of these flow tables is explained in more detail in Chapter 4. A considerable portion of the workflow is highlighted in gray. This box represents the route selection procedure. In a classical link state routing protocol, this box would contain a call to a traditional shortest path algorithm, such as Dijkstra algorithm. In this case, the routines represent the same procedure shown in Algorithm 3.1.

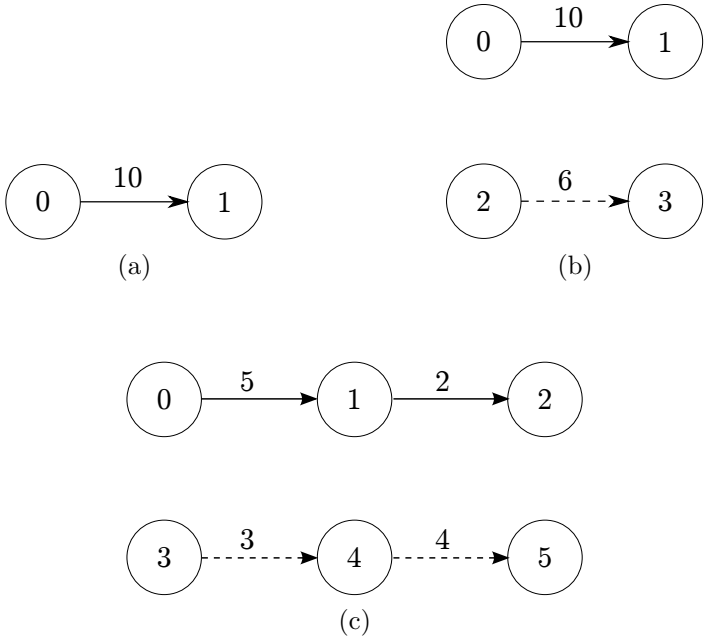


Figure 3.2: Examples of simple scenarios that help to illustrate the definitions of steady state of a simulation, transient state of a simulation and steady state cycle length. Figure (a) shows a scenario with a single flow, while the two other figures illustrate scenarios with two flows (differentiated by different kinds of arrows).

3.2 Aggregated Throughput Estimation Function

Function *EstimateThroughput*, called by Algorithm 3.1, receives three arguments: the graph G representing the network topology, the set of currently active flows F and a candidate solution, *i.e.*, a path set containing paths for all flows in set F. The goal of this function is to return an estimate of the aggregated throughput that would be achieved if the specified path set is employed.

To this end, function *EstimateThroughput* performs a simulation of the sequence of events (transmissions, packet queuing, dequeuing and discards) that would occur in the network by employing the paths of the candidate solution. During this simulation, the function tries to find a good estimate for the *long term aggregated throughput*. By long term, we refer to the average network throughput once the transient effects at the beginning of the flows are over. For instance, in Chapter 1, a simple scenario with two flows was presented, illustrated in Figure 1.2. In that scenario, assuming a given schedule of transmissions in the MAC layer, the delay of the first packet to traverse one of the paths (Path 1) was 11 ms, while the following packets would be delivered with intervals of 9 ms. If one simulates that scenario and considers the interval from $t = 0$ to $t = 20$, the average throughput would be of $2/20 = 0.1$ packets per unit of time. As the sampling interval

increases, throughput asymptotically approaches $1/9 \approx 0.111$ packets per unit of time. By adopting the long term aggregated throughput, we are implicitly assuming network flows to have a long duration.

In that specific scenario, considering the deterministic scheduling used in the example (where a packet always has priority for using the medium over the ones generated later), immediately after $t = 11$, network events start to repeat themselves in cycles. As a result, the intervals for the arrival of the packets to the destination node also become repetitive. With that characteristic in mind, we present the following definitions.

Definition 3.1 *Steady State of a Simulation. Given a network topology, a set of flows, a set of paths used by the flows, and a deterministic medium access schedule, the simulation is said to be in steady state if and only if all events from that point on repeat themselves for later packets in the same order and time intervals.*

Definition 3.2 *Transient State of a Simulation. Given a network topology, a set of flows, a set of paths used by the flows, and a deterministic medium access schedule, the simulation is said to be in transient state if and only if it has not yet reached a steady state.*

Definition 3.3 *Steady State Cycle. In a simulation that has reached the steady state, the cycle is the shortest sequence of events that repeats itself indefinitely.*

Definition 3.4 *Steady State Cycle Length. In a simulation that has reached the steady state, the cycle length is the time difference between the occurrences of correspondent events in consecutive occurrences of the cycle.*

Figure 3.2 shows a number of simple scenarios that help illustrating these definitions. In all figures, the weights of the edges represent the transmission delay for the respective link. In Figure 3.2a, at time $t = 0$, the first packet of the flow $0 \Rightarrow 1$ starts to be transmitted. After 10 units of time, it arrives at node 1, while the second packet starts to be transmitted (assuming the flow is backlogged at node 0). This cycle repeats itself indefinitely. Hence, in Figure 3.2a the steady state starts at $t = 0$, which means there is no transient state, and the length of the steady state cycle, which is composed of a single transmission event, is equal to 10 time units.

In Figure 3.2b two different interpretations can be made, depending on whether links $0 \rightarrow 1$ and $2 \rightarrow 3$ are interfering or not. Assuming that they do interfere with each other,

at time $t = 0$ one of the two source nodes obtains the right to access the medium, say node 0. Node 0, then, proceeds to start the transmission, while node 2 waits its turn. This transmission lasts 10 units of time, after which there is another dispute for the usage of the wireless medium. Assume the schedule of transmissions is perfectly fair, meaning that now node 2 gains the right to transmit. This new transmission takes 6 units of time, after which node 0 can transmit again. Assuming the links will continue to be used in a perfectly alternate manner, this scenario presents a cycle length of 16 time units, with the steady state starting at $t = 0$, meaning again that there is no transient state. If one assumes, however, that both links $0 \rightarrow 1$ and $2 \rightarrow 3$ can be used simultaneously, than the following events will occur:

1. At $t = 0$, both links $0 \rightarrow 1$ and $2 \rightarrow 3$ start to transmit.
2. At $t = 6$, the first packet of flow $2 \Rightarrow 3$ arrives at the destination, while the second packet starts to be transmitted at link $2 \rightarrow 3$. Link $0 \rightarrow 1$ is still transmitting its respective first packet.
3. At $t = 10$, the first packet of flow $0 \Rightarrow 1$ arrives at the destination, while the second packet starts to be transmitted at link $0 \rightarrow 1$. Link $2 \rightarrow 3$ is still transmitting its respective second packet.
4. At $t = 12$, the second packet of flow $2 \Rightarrow 3$ arrives at the destination, while the third packet starts to be transmitted at link $2 \rightarrow 3$. Link $0 \rightarrow 1$ is still transmitting its respective second packet.
5. At $t = 18$, the third packet of flow $2 \Rightarrow 3$ arrives at the destination, while the fourth packet starts to be transmitted at link $2 \rightarrow 3$. Link $0 \rightarrow 1$ is still transmitting its respective second packet.
6. At $t = 20$, the second packet of flow $0 \Rightarrow 1$ arrives at the destination, while the third packet starts to be transmitted at link $0 \rightarrow 1$. Link $2 \rightarrow 3$ is still transmitting its respective fourth packet.
7. At $t = 24$, the fourth packet of flow $2 \Rightarrow 3$ arrives at the destination, while the fifth packet starts to be transmitted at link $2 \rightarrow 3$. Link $0 \rightarrow 1$ is still transmitting its respective third packet.
8. At $t = 30$ both the fifth packet of flow $2 \Rightarrow 3$ and the third packet of flow $0 \Rightarrow 1$ arrive at their respective destinations. The transmissions of the sixth packet of flow $2 \Rightarrow 3$ and seventh packet of flow $0 \Rightarrow 1$ start.

Notice that at time $t = 30$, transmission events start to repeat the same cycle that started at $t = 0$. Therefore, in this case, the steady state is reached in time $t = 0$, with cycles of 30 time units.

The example illustrated in Figure 3.2c is the most complicated of the three. For this example, assume that links $0 \rightarrow 1$ and $4 \rightarrow 5$ can be used simultaneously. Similarly, links $3 \rightarrow 4$ and $1 \rightarrow 2$ do not interfere with each other. For any other combination of links, simultaneous transmissions are impossible. At time $t = 0$, both nodes 0 and 3 compete for the wireless medium. Assume that in this case, node 0 has the priority over node 3. Node 0 will, then, perform its transmission to node 1, which will last for 5 units of time. At that moment, three nodes will have packets to transmit, nodes 0, 1, and 3. Since node 3 has been waiting for the longest time, assume it receives the right to transmit. Since link $1 \rightarrow 2$ does not interfere with link $3 \rightarrow 4$, node 1 can transmit as well. After two more units of time (at $t = 7$), the first packet of flow $0 \Rightarrow 2$ arrives at its destination. The link layer transmission from node 3 to node 4 is only completed at time $t = 8$, though. At that point, nodes 0, 3, and 4 will compete for the right to transmit. Again, lets assume the node waiting for the longest time (node 0) is awarded. Since links $0 \rightarrow 1$ and $4 \rightarrow 5$ are non-interfering, node 4 can transmit as well. At $t = 12$, the first packet of the flow $3 \Rightarrow 5$ arrives at its destination. One time unit later, at $t = 13$, the transmission from node 0 to node 1 is over and now there are 3 nodes trying to access the medium: 0, 1, and 3, with node 3 being the one waiting for the longest time. Notice that this situation is exactly equal to the situation at time $t = 5$ (after the first transmission performed by node 0). Assuming the medium access schedule is deterministic and stays the same, the same sequence of transmissions will be repeated, culminating in the same *simulation state* at time $t = 21$. According to the previously presented definitions, the simulation has reached the steady state at $t = 5$, with a cycle length of 8 units of time. Differently from the previous examples, all events that happen before $t = 5$ constitute a transient state for this simulation.

Another definition that will be useful for the remainder of this chapter is the definition of *Steady State Throughput*:

Definition 3.5 *Steady State Throughput. In a simulation that has reached the steady state, the steady state throughput is the average aggregated network throughput considering the time interval between the moment when the steady state was reached until a time* $t_f \rightarrow \infty$. *In other words, it is the average aggregated network throughput for a infinitely long simulation, disregarding the initial transient state.*

With regard to this last definition, the following proposition can be stated and demonstrated:

Proposition 3.6 *The steady state throughput is equal to the average throughput of a single steady state cycle.*

Proof Let the moment the simulation reaches the steady state be denoted by t_s. Let L denote the length of one steady state cycle. Finally, let D be the number of packets delivered for their respective destination nodes within each steady state cycle and $T_c = D/L$ the throughput during a single steady state cycle. For a given, finite value of t_f (the end of the simulation), there are two possibilities: either $t_f - t_s$ is a multiple of the steady state cycle length L or it is not.

If $t_f - t_s$ is a multiple of L, then this interval is only composed of complete steady state cycles. In this case, exactly $D \times (t_f - t_s)/L$ packets were delivery during the steady state and the steady state throughput T_s is given by:

$$T_s = D \times \frac{t_f - t_s}{L} \times \frac{1}{t_f - t_s} = \frac{D}{L} = T_c \tag{3.7}$$

On the other hand, if $t_f - t_s$ is not a multiple of L, then the last cycle within that time interval is incomplete. In other words, there is a residue at the end of the interval during which nothing can be stated in the general case. Let T_r denote the unknown throughput during this residue and $L_r = (t_f - t_s) \bmod L$ be the residue length. Under these circumstances, the steady state throughput is given by the weighted average:

$$T_s = \frac{L_r}{t_f - t_s} \times T_r + \frac{t_f - t_s - L_r}{t_f - t_s} \times T_c \tag{3.8}$$

Since L_r is always lower than L_c, as $t_f - t_s \to \infty$, the first term arbitrarily approaches 0, while the second approaches T_c. Therefore, the steady state throughput is equal to the throughput of a single steady state cycle. □

In this thesis, we argue that the steady state throughput (assuming the simulation ever enters such state) is a better (fairer) metric for evaluating the aggregated network throughput resultant from the usage of a given path set than the throughput considering any other arbitrary time interval starting at $t = 0$. If the network flows are sufficiently long, the transient state might not represent well the resultant network throughput because it can be either much higher or much lower. On the other hand, assuming flows

Algorithm 3.2 Macro vision of the aggregated throughput estimation function.

1: **function** EstimateThroughput(Graph G, Flow set F, Path set C)
2: Initialize variables $t \leftarrow 0$, $AL \leftarrow \emptyset$, and $WT \leftarrow \emptyset$
3: **for all** flow $f \in F$ **do**
4: Add packet first packet of flow f to the buffer of node $src(f)$
5: $WT \leftarrow WT \cup \{src(f)\}$
6: **end for**
7: **for all** node $i \in WT$, sorted according to medium access priority **do**
8: $P \leftarrow$ first packet in the queue of node i
9: $f \leftarrow$ flow of packet P
10: $l \leftarrow$ first hop of path in C for the flow f
11: **if** l is not blocked by any link with higher priority **then**
12: $AL \leftarrow AL \cup \{l\}$
13: $TransmissionEnd_l \leftarrow t +$ Transmission delay of l
14: **if** the queue of node i is now empty **then**
15: $WT \leftarrow WT - \{i\}$
16: **end if**
17: **end if**
18: **end for**
19: **while** simulation steady state is not reached **do**
20: $t \leftarrow \min_{l \in AL} TransmissionEnd_l$
21: **for all** link $l \in AL$ s.t. $TransmissionEnd_l = t$ **do**
22: $P \leftarrow$ packet just carried by link l
23: $f \leftarrow$ flow of packet P
24: $AL \leftarrow AL - \{l\}$
25: **if** $dst(f) \neq dst(l)$ **then**
26: Add packet P to the buffer of node $dst(l)$
27: $WT \leftarrow WT \cup \{dst(l)\}$
28: **end if**
29: **if** $src(f) = src(l)$ **then**
30: Add next packet of flow f to the buffer of node $src(f)$
31: $WT \leftarrow WT \cup \{src(f)\}$
32: **end if**
33: **end for**
34: Repeat actions from steps 7 through 18
35: **end while**
36: Return the throughput of a cycle
37: **end function**

are sufficiently long, the overall network throughput (including the transient state), converges to the steady state throughput with time. Although in this thesis we do not assume knowledge of the actual duration of the flows, we do assume that they are long in comparison with the length of the transient state, so that the steady state throughput is a good approximation for the overall network throughput.

It is, therefore, the goal of our aggregated throughput estimation function to find and return the steady state throughput. To this end, this function performs the macro steps listed in Algorithm 3.2. This function uses three main control variables, namely t, the simulation time, AL, the set of currently active links, and WT, the set of nodes that want to transmit (*i.e.*, that currently have packets in their buffers).

After initializing the main control variables, the function proceeds by generating the first packet of each flow and adding it to the correspondent source node buffer (lines 3 through 6). Once those packets are generated, the function triggers the initial transmissions (lines 7 through 18). This is done by traversing the set WT according to the medium access priority taking the first packet from each node's buffer and checking whether the required link can be used. If a link l is not blocked (*i.e.*, if it can be used), the function computes the transmission delay for that link, adds it to the current time and stores the result on variable $TransmissionEnd_l$. If the packet just removed from the correspondent's node buffer was the only packet currently there, the node no longer wants to transmit, and therefore must be removed from the set WT. Notice, however, that this is not always the case even at the beginning of the simulation, since multiple flows may share the same source node.

At this point, the function reaches the main loop (lines 19 through 35). Each iteration of this loop can be divided in three tasks: updating the simulation time (line 20), consolidating the transmissions that are currently ending (lines 21 through 33), and triggering new transmissions (line 34).

To update the simulation time, it suffices to find the lowest value of the variables $TransmissionEnd_l$ among all active links l. Since all possible transmissions were scheduled previously, no other network event is possible until at least one active link finishes its transmission.

In order to consolidate the transmissions currently ending, the function loops through all links of the AL set, checking if their respective $TransmissionEnd_l$ is equal to the value of the control variable t. Whenever that is the case (and it has to be the case for at least one link), link l is removed from the AL set, and if the destination of the link is not

the final destination of the packet, the packet is added to the buffer of the next hop. A special case is when the link just carried a packet from the source of the respective flow. In that case, once the packet has just traversed its first link, the next packet from the flow must be created on the source node's buffer (thus, maintaining the source backlogged).

Finally, all possible new transmissions (if any) are triggered by executing the same steps already described in lines 7 through 18.

The main loop ends when the function detects that the simulation has reached the steady state. At that point, the function discovers what is the length of the steady state cycle and the number of packets delivered within a cycle so that it can compute the steady state throughput. This value is then returned.

Notice that this is a very preliminary description of the aggregated throughput estimation function, in which many details (*e.g.*, how the medium access priorities work, and how does the algorithm detects that the simulation has reached the steady state) have been left out for the sake of readability. On the next subsections, these details will be explained.

3.2.1 Conflict Graph and Link Blocking

Line 11 (and, consequently, line 34) of Algorithm 3.2 performs a test to verify if a given link can currently be used. This test is put in place in order to avoid activating two links that *block* each other. By blocking, we mean two links that, for some reason, cannot or should not be used together. In this thesis, we consider that two links l_1 and l_2 cannot or should not be used simultaneously in the following cases:

- both links share the same source node;
- both links share the same destination node;
- the destination for one of the links is among the neighbors of the source node for the other link; or
- the source node for one of the links is a neighbor of the source node for the other link.

The first two cases are obvious, since a node cannot receive or transmit two packets simultaneously. The third case stems from the fact that, given the shared nature of the

Algorithm 3.3 Procedure that builds a conflict graph for a network.

```
1: function BuildConflictGraph(Graph G, Path set C)
2:     Initialize conflict graph CG with all links used in C as nodes, but no edges.
3:     for all path P_i ∈ C do
4:         for all link l_i ∈ P_i do
5:             for all path P_j ∈ C do
6:                 for all link l_j ∈ P_j do
7:                     d ← delivery probability for link src(l_i) → src(l_j)
8:                     if d > NeighborhoodThreshold then
9:                         Add edge between l_i and l_j in CG
10:                    else
11:                        d ← delivery probability for link src(l_j) → src(l_i)
12:                        if d > NeighborhoodThreshold then
13:                            Add edge between l_i and l_j in CG
14:                        else
15:                            d ← delivery probability for link src(l_i) → dst(l_j)
16:                            if d > NeighborhoodThreshold then
17:                                Add edge between l_i and l_j in CG
18:                            else
19:                                d ← delivery probability for link src(l_j) → dst(l_i)
20:                                if d > NeighborhoodThreshold then
21:                                    Add edge between l_i and l_j in CG
22:                                end if
23:                            end if
24:                        end if
25:                    end if
26:                end for
27:            end for
28:        end for
29:    end for
30:    Return CG
31: end function
```

wireless medium, if a node starts transmitting while one of its neighbors is currently receiving another frame, the two signals will mix, probably resulting in a collision (*i.e.*, the receiving node might fail to receive its intended packet). Finally, the fourth case disallows two neighbor nodes to transmit simultaneously. While there are no physical constraints on this kind of event, there are important MAC layer protocols based on the CSMA/CA technique, which forbids both transmissions to happen simultaneously (one of the nodes would sense the medium as being busy, postponing its transmission[1]).

[1]There is one exception to this rule which happens when both transmitters are completely synchronized. In that case, both can sense the wireless medium as being idle at the exact same time, leading them to transmit simultaneously. However, this total synchronization is usually considered an undesirable casual event for CSMA/CA.

It is a common practice in the literature the usage of an *interference range* [87]. The interference range is the region within which a node's transmission may disturb receptions by other nodes. Most works adopt an interference range in terms of a radius centered at the node and it is common for this radius to be considered twice the size of the one for the transmission range [87]. It is not difficult, however, to find examples of how this strategy can fail to model the actual interference caused by a node. One such example is the possible existence of multiple transmission rates at the link layer. In that case, the transmission range is dependent on the used transmission rate [75]. If a node uses a higher transmission rate, it might decrease its effective transmission range. However, as long as the transmission power is kept constant, its ability to interfere with other nodes' receptions is maintained. Due to this definition being somewhat arbitrary, in this thesis we opted to employ a more concrete set of rules, considering only neighbors to be affected by the transmissions of a node.

In any case, given the cases in which two links cannot be used simultaneously, it is possible to build a conflict graph [50]. As explained in Chapter 2, this data structure represents the relationships between network links in terms of whether they can be used simultaneously or not in the form of a graph. In this graph, each network link is represented by a node, while an edge between two nodes (links) exists if, and only if, these links cannot be used together.

Before starting the simulation, function *EstimateThroughput* actually builds such a conflict graph using the steps depicted in Algorithm 3.3. Although the conditions for simultaneous usage could be tested during the simulation only when needed, having this information cached in the form of a conflict graph helps decreasing the complexity of the simulation when the number of simulated events grows. The algorithm tests the cases listed in the beginning of this section by using the information of the delivery probability, which must be associated with each edge of graph G. For a given pair of links, lines 8 and 12 check whether both source nodes are neighbors. We implicitly assume that the delivery probability from a node to itself is always 1, which make those two conditions also cover the case of both links sharing the same source node. Lines 16 and 20 test whether the source for one link may interfere with the receiver of the other link. Notice that this includes the case of both links sharing the same destination node, since, in that case, the source of a link is a neighbor of the destination node of the other link.

Notice that this procedure does not build a conflict graph for all network links. Instead, only links that belong to one of the paths in the currently tested path set are

added to the conflict graph, because those are the only links that will be needed during the simulation. It is also important to notice the employment of a parameter called $NeighborhoodThreshold$ to evaluate if two nodes are neighbors. We only consider two nodes to be neighbors if the delivery probability between them is higher than this threshold. This helps avoiding that very bad links impact the decision of whether two links interfere with each other, specially when large windows are used to compute an estimate for the delivery probability.

3.2.2 Medium Access Priorities

During a simulation, ideally, whenever a node has packets to transmit, it would be granted access to the wireless medium to perform its next transmission. However, since the usage of a link may block the usage of other network links, the *EstimateThroughput* function is required to employ a well-defined policy to choose a subset of unblocked links for transmission at each new iteration. This is done through the establishment of a set of rules for determining the order of priority for each node to access the wireless medium.

In this thesis, one requirement for this set of rules is that the long term characteristics of the medium access would resemble those of real wireless MAC protocols, so that the results of the simulations can be representative of the actual network performance. Since there are many different wireless MAC protocols being used in practice [49, 47, 48, 83], we choose to focus on the one defined by the IEEE 802.11 standard [49]. Notice that the IEEE 802.11 MAC protocol uses CSMA/CA with a random exponential backoff, which inserts a probabilistic component into the medium access sharing policy. For the purposes of our route selection algorithm, we cannot resort to probabilistic components since it is important that the algorithm yields the same results given the same inputs (we assume the usage of a link state routing protocol, in which each node must have a complete network view so that they can employ the same route selection algorithm and reach the same results). Since the IEEE 802.11 MAC protocol tends to achieve an even distribution of the medium usage in the long term [56], we propose that function *EstimateThroughput* uses a set of deterministic rules that guarantee an uniformly distributed medium access share for all network nodes.

With that goal in mind, the *EstimateThroughput* function employs the steps described in Algorithm 3.4 for choosing the nodes that will be allowed to transmit in a given moment. This transmitter selection procedure takes into account four main variables: the precomputed conflict graph CG, the set of nodes that want to transmit WT, and two

Algorithm 3.4 Procedure that chooses which nodes can use the wireless medium at a given moment.

```
1: function ChooseTransmitters(Conflict Graph CG, Set of Nodes that Want to
   Transmit WT, Set of Blocked Links BL, Set of Priority Blocked Links PBL)
2:     for all nodes i ∈ WT, sorted by how long they have been waiting do
3:         P ← first packet in node i's buffer
4:         l ← link needed by packet P
5:         if l ∉ BL and l ∉ PBL then
6:             Output node i as a new transmitter
7:             for all link l_j s.t. an arrow between l and l_j exists in CG do
8:                 BL ← BL ∪ {l_j}
9:             end for
10:        else
11:            for all link l_j s.t. an arrow between l and l_j exists in CG do
12:                PBL ← PBL ∪ {l_j}
13:            end for
14:        end if
15:    end for
16: end function
```

sets of blocked links, BL and PBL.

Nodes are given priority based on how long they have been waiting to transmit. To this end, whenever nodes are added to the WT in Algorithm 3.2, they are also implicitly added to a linked list, so that the order in which they requested the usage of the wireless medium can be preserved. Whenever this transmitter selection procedure is called, it traverses this list analyzing the feasibility of allowing each node to transmit. To do so, it retrieves the first packet in the node's buffer and the necessary link l for the transmission. It then checks whether l belongs to either of the sets BL or PBL. If it does not, the procedure allows the node to transmit and blocks all links that are connected to l in the conflict graph, by adding them to the set BL. Otherwise, the node is not allowed to transmit, but, still, the procedure adds l to the auxiliary set PBL.

The set BL is the set of links that are currently blocked because another conflicting link is already in use. On the other hand, links on the set PBL are blocked because there is a high priority node that could not gain access to the wireless medium that would have used a conflicting link. The set PBL is needed because otherwise the medium access schedule might result in priority inversion for certain nodes, *i.e.*, they would be denied access because they would be blocked by nodes with lower priorities.

Figure 3.3 illustrates this situation. Consider a network (or part of the topology) with nodes distributed as shown in Figure 3.3a. There are three nodes initially trying

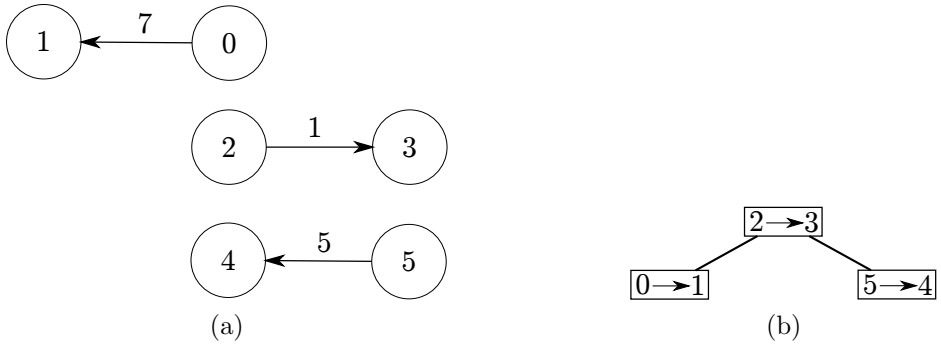


Figure 3.3: Example of a simple scenario that illustrates the problem of priority inversion that could happen without the employment of a priority block policy. The leftmost figure shows an hypothetical topology with three transmitting links, while the rightmost figure shows the relevant portion of the conflict graph.

to access the wireless medium: nodes 0, 2, and 5. Assume the relevant links form the conflict graph shown in Figure 3.3b and that all transmitters are backlogged. Assume also that the node which has been waiting for a transmission opportunity for the longest time is node 0, followed by nodes 2 and 5 (in that order). By employing the steps listed in Algorithm 3.4, node 0 is granted access to the wireless medium, which blocks the access from node 2. However, if we ignore the contents of the set PBL, node 5, which had a lower priority, is granted access as well. After five units of time, node 5 finishes its transmission and function *EstimateThroughput* reevaluates the medium access schedule. Node 5 will again compete with node 2 for using the medium, and it will win again, despite having a lower priority (node 2 is still blocked by the transmission of node 0). When node 0 finally finishes its transmission, node 2 has another opportunity to compete for the medium, but it will lose again because it is now blocked by node 5, while node 0 is not. In fact, in this example, node 0 will transmit 5 times, while node 5 will transmit 7 times before node 2 has a chance to transmit.

By employing a second blocking system, based on priorities, we effectively avoid that priority inversion issue, creating a fairer schedule for all nodes. One could argue that, in an actual IEEE 802.11 based network, node 2 would indeed receive less opportunities to transmit than the other two nodes because of its position. While that is true, it is necessary to bear in mind the fact that function *EstimateThroughput* only simulates complete link layer transmissions (comprising all retransmission attempts). In reality, nodes compete for the medium for every retransmission attempt, which mitigates this problem, reducing the time a node (such as node 2 in the example) would have to wait for an opportunity.

Notice that the procedure shown in Algorithm 3.4 relies on the maintenance of the value of set BL between executions (a link that is blocked will remain blocked until all conflict links stop transmitting). The same is not true with the set PBL: this set is emptied between executions of this procedure.

One final note regarding the medium access priorities used by function *EstimateThroughput* is the initial order of the priority link. At the beginning of the simulation, all nodes that are source of any flows are added to the WT set and, therefore, are also inserted in the linked list that maintains the order that defines the priority of each node. We implicitly assume that all flows are given an index before function *EstimateThroughput* and the order defined by that index defines the order in which nodes are traversed initially for insertion on the WT set.

3.2.3 Link Transmission Delay

One important question regarding the simulation performed by function *EstimateThroughput* is how to compute the transmission delay of a given link. Whenever a link is chosen to transmit at a certain point of the simulation, function *EstimateThroughput* computes its transmission delay in order to schedule the moment when the transmission will be over (line 13 of Algorithm 3.2).

As explained in the beginning of this chapter, we assume that the topology graph G contains the information of the delivery probability for all network links. Since in this work we also assume that all nodes transmit at the same fixed transmission rate, we adopt the expected number of link layer retransmissions as the metric for evaluating the transmission delay for a link. Although the idea is similar to the ETX metric of a link, in this work we resort to a different approach to compute this value.

The IEEE 802.11 standard defines the use of the ARQ technique. Whenever a node attempts to transmit a frame at the link layer, it waits for a predefined period for an acknowledgment frame, indicating that the transmission was successful. If this acknowledgment is not received, the sender tries to retransmit the original frame. The standard, however, defines a limit to the number of retries: 4 times for "large" packets (above 100 bytes), and 7 times for "small" packets (100 bytes or less). The original ETX metric disregards this limit, *e.g.*, if the ETX for a link is found to be 15, that value will be used regardless of the retry limit at the link layer.

In the routing algorithm presented in this thesis, we take into account the fact that

no frame can be retransmitted more than k times and we compute the expected number of retransmissions accordingly. To this end, the transmission delay for a link $a \rightarrow b$ is defined by the following expression:

$$TD_{a\rightarrow b} = \sum_{i=1}^{k} i \cdot (d_{a\rightarrow b} \cdot d_{b\rightarrow a}) \cdot [1 - d_{a\rightarrow b} \cdot d_{b\rightarrow a}]^{(i-1)} + k \cdot [1 - d_{a\rightarrow b} \cdot d_{b\rightarrow a}]^{k}, \qquad (3.9)$$

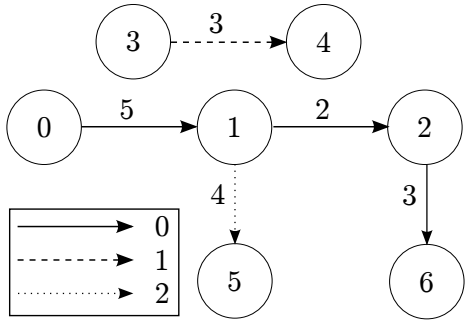
where $d_{a\rightarrow b}$ and $d_{b\rightarrow a}$ denote, respectively, the delivery probabilities for links $a \rightarrow b$ and $b \rightarrow a$. This expression is basically an average of the $k+1$ cases that can happen (success in $1, \ldots, k$ transmissions, and failure after k attempts). Specifically, for the performance evaluation presented in Chapter 4, we adopt $k = 4$, in order to model a IEEE 802.11 network, considering "large" packets.

Limiting the maximum transmission delay for a link is important because in our route selection algorithm it affects not only that particular link, but also any link that conflicts with it. Suppose for example that function *EstimateThroughput* is trying to evaluate a set of paths that make use of a link $a \rightarrow b$ with $d_{a\rightarrow b} = d_{b\rightarrow a} = 0.25$. The ETX of this link is, then, 16. If this value was to be used in the simulations, not only it would impact the performance for link $a \rightarrow b$ (due to the transmission requiring 16 units of time), but it would also mean conflicting links would be unusable for 16 units of time. In practice, though, after the fourth retry, the MAC layer would give up (regardless of whether the packet was successfully transmitted), allowing other nodes to gain access to the wireless medium. Our expression, thus, is capable of representing this.

3.2.4 Steady State Cycle Detection

One final functionality needed by function *EstimateThroughput* is the capability of detecting that the simulation has reached the steady state and discovering the size of the steady state cycle and the number of packets delivered during a cycle. According to Definition 3.1, the steady state of a simulation is reached when the transmission events start to repeat themselves in the same sequence indefinitely. One initial approach for detecting whether the simulation has reached the steady state, then, is to monitor all transmissions and try to detect repetitions in the transmission sequences.

One problem with that approach is how to guarantee that the next expected transmission events will follow knowing only that the last few transmission events were repeated. Consider, for example, the situation illustrated in Figure 3.4. The figure shows a simple



Transmission Events	
t = 0: 0→1	t = 34: 2→6
t = 5: 3→4	t = 37: 0→1
t = 8: 1→5	t = 42: 3→4
t = 12: 0→1	t = 45: 1→2
t = 17: 3→4	t = 47: 0→1
t = 20: 1→2	t = 52: 3→4
t = 22: 0→1	t = 55: 1→2
t = 27: 3→4	t = 57: 2→6
t = 30: 1→5	t = 60: 0→1

Queue State for Node 1	
t = 0: 1	t = 34: 0, 0, 1
t = 5: 1, 0	t = 37: 0, 0, 1
t = 8: 1, 0	t = 42: 0, 0, 1, 0
t = 12: 0, 1	t = 45: 0, 0, 1, 0
t = 17: 0, 1, 0	t = 47: 0, 1, 0
t = 20: 0, 1, 0	t = 50: 0, 1, 0, 0
t = 22: 1, 0	t = 55: 0, 1, 0, 0
t = 27: 1, 0, 0	t = 58: 1, 0, 0
t = 30: 1, 0, 0	t = 60: 1, 0, 0

Medium Access Priority	
t = 0: 0, 3, 1	t = 34: 2, 0, 3, 1
t = 5: 3, 1, 0	t = 37: 0, 3, 1
t = 8: 1, 0, 3	t = 42: 3, 1, 0
t = 12: 0, 3, 1	t = 45: 1, 0, 3
t = 17: 3, 1, 0	t = 47: 0, 3, 1, 2
t = 20: 1, 0, 3	t = 52: 3, 1, 2, 0
t = 22: 0, 3, 1, 2	t = 55: 1, 2, 0, 3
t = 27: 3, 1, 2, 0	t = 57: 2, 0, 3, 1
t = 30: 1, 2, 0, 3	t = 60: 0, 3, 1, 2

Figure 3.4: Example of the simulation of a simple scenario with three flows. Different arrows are used with each different flow. The top-right box shows the sequence of transmission events, while the bottom-left and bottom-right boxes represent, respectively, the state of the queue of node 1 at each moment in time and the evolution of the medium access priority list.

network with three active flows (denoted by the three different types of arrows). The box to the right of the network shows a possible sequence of transmission events, assuming all used links conflict with each other. Those transmission events were generated using the medium access rules described previously in this chapter. The figure also shows the evolution of the queue state for node 1, as well as the evolution of the medium access priority list. If we start keeping track of the transmissions from time $t = 0$ looking for repeated patterns, it is possible to notice that the fourth and fifth simulation events match the first and second, respectively. This might suggest that the simulation is in steady state **and** the steady state cycle is formed by the three first simulation events. However, the sixth simulation event does not match the third, which means either that the simulation has not reached steady state or that the original steady state cycle is different from the one originally predicted (in fact, in this example, the steady state is not reached for any of the shown events).

This method fails because it misses an important information when trying to predict whether events will repeat themselves: the current state of the nodes' buffers. Specifically, in the example of Figure 3.4 the third and sixth events do not match because the state of the buffer of node 1 was different in at each of those moments. While at $t = 8$ the buffer

contained a packet from flow $1 \Rightarrow 5$ followed by a packet from flow $0 \Rightarrow 6$, at $t = 20$ the first packet in node one's buffer is from flow $0 \Rightarrow 6$, causing link $1 \rightarrow 2$ to be used instead.

Another relevant information when trying to assess the steady state is the current order of priority for nodes trying to access the wireless medium. Consider for example, the situation at time $t = 34$ with the situation at time $t = 12$. At both moments, the last three transmission events match (transmissions using links $0 \rightarrow 1$, $3 \rightarrow 4$, and $1 \rightarrow 5$). At time $t = 12$, the transmission event that follows is the usage of link $0 \rightarrow 1$. Notice that the queue state of node 0 at time $t = 34$ matches that of time $t = 12$ (in both cases, the queue contains a single packet from flow $0 \Rightarrow 6$). However, the event that follows at $t = 34$ is a transmission through the link $2 \rightarrow 6$. That happens because the medium access priority list was different at those two moments, with node 2 being the next in line to gain access to the medium in the latter.

The example shown in Figure 3.4 suggests three relevant information for determining whether or not the transmissions events in a simulation will actually repeat themselves indefinitely: the sequence of transmissions, the state of the queue for each node, and the medium access priority list. Based on that, we present the following definition that will be useful for the remainder of this section:

Definition 3.10 *State of a Simulation. The state of a simulation (or simulation state) is formed by the currently ongoing transmissions with their respective remaining times, the content of the buffer for all nodes that are traversed by any active path on the evaluated path set (including the source for a flow, but not including the destination) and the current medium access priority list.*

Notice that the word "state" is employed in a somewhat more specific sense in this definition than for the definitions of steady state and transient states. In the case of the state of a simulation, we refer to the set of characteristics that, along with the topology graph and the set of paths currently being evaluated, completely determines a point within the simulation. In other words, given the topology graph, the current path set and any state of a simulation, it is possible to continue the execution of that simulation from that point in time, without having to process all previous events.

Various examples of states of a simulation are given in Figure 3.4. For each time t shown by the boxes on that figure, we have information regarding ongoing transmissions, queue state and medium access priority. In that case, though, the presented state is only partial, because it is missing the queues for nodes 0, 3, and 2, as well as the remaining

time for each ongoing transmission.

Not all combinations of values for the fields of a simulation state are feasible. For instance, on the example presented in Figure 3.4 it would not be feasible to have a simulation state in which both links $0 \rightarrow 1$ and $1 \rightarrow 2$ are active. It is also impossible to have a simulation state in which node 1 stores a packet from flow $3 \Rightarrow 4$, since the path being evaluate for that flow does not contain node 1. With that in mind, we reach the following definitions:

Definition 3.11 *Valid Simulation State. We say that a simulation state is valid if, and only if, the following conditions are met:*

- *the list of active links does not contain links that interfere with each other;*
- *the remaining time of any active transmission is not greater than the transmission delay of its respective link;*
- *for each flow, packets only exist in the buffers of nodes that are part of the respective path being currently evaluated;*
- *each node appears, at most, once in the medium access priority list; and*
- *the medium access priority list contains only nodes with non-empty queues.*

Definition 3.12 *Initial Simulation State. The initial simulation state for a given simulation has all links inactive, a single packet of each flow placed on their respective source nodes' buffers, and a medium access priority list containing exactly the nodes that are source for any of the flows appearing at exactly once.*

An initial simulation state is clearly a valid simulation state. Function *EstimateThroughput* always generates an initial simulation state as its first state. At each iteration, it proceeds applying a transition based on the characteristics of the network and the path sets, which generates other valid simulation states. A sequence of iterations, thus, generates a sequence of valid simulation states. One final definition presented on this section is the following:

Definition 3.13 *Reachable Simulation State. A simulation state is said to be reachable if, and only if, it is produced by function EstimateThroughput after zero or more iterations, starting with a given initial simulation state.*

In simple terms, reachable states are simply the states that are produced during the execution of the function *EstimateThroughput*. Notice that, for a given simulation, there can be valid states that there are, nevertheless, unreachable.

These definitions are relevant because they allow us to demonstrate that it is possible to detect the steady state within a simulation performed by function *EstimateThroughput* by simply checking whether a repeated simulation state was generated. To this end, we first demonstrate that, starting at a given simulation state, function *EstimateThroughput* will always generate the same sequence of simulation states (Propositions 3.14 and 3.15). As a consequence, whenever a repeated simulation state is found the sequence of generated states that lead to the repetition will repeat itself indefinitely (Corollary 3.16).

Proposition 3.14 *Let t_0 and t_1 be two simulation times within a simulation performed by function EstimateThroughput. Let S_{t_0} and S_{t_1} denote their respective simulation states. Finally, let $S_{t_0}^{+1}$ and $S_{t_1}^{+1}$ denote the next states generated by function EstimateThroughput immediately (with one more iteration) after S_{t_0} and S_{t_1}, respectively. If $S_{t_0} = S_{t_1}$, then $S_{t_0}^{+1} = S_{t_1}^{+1}$.*

Proof At both times (immediately after states S_{t_0} and S_{t_1}), function *EstimateThroughput* advances until the moment when the current active transmission with the lower remaining time (there might be multiple) will be over. Since states S_{t_0} and S_{t_1} are equal, the time offset will be equal in both cases and the set of transmissions that will be over will also be the same. Since the finished transmissions will be the same, the same packets will be added to the end of the same buffers in both cases. Since the buffers were also equal in both S_{t_0} and S_{t_1}, the final state of every node's buffer will be equal as well, causing all possible changes (additions to the medium access priority list) to be the same.

Given that the same time offset occurs in both cases, the links that remain active from S_{t_0} to $S_{t_0}^{+1}$ and from S_{t_1} to $S_{t_1}^{+1}$ will have their remaining times decreased by the same value. Since the states of the nodes' buffers, the medium access priority list, and the currently blocked links are all the same, the same set of new active links will be added to both states $S_{t_0}^{+1}$ and $S_{t_1}^{+1}$, which, in turn, will result in the same modifications to the buffers and medium access priority list. Therefore, both states $S_{t_0}^{+1}$ and $S_{t_1}^{+1}$ are identical. □

Proposition 3.15 *Let t_0 and t_1 be two simulation times within a simulation performed by function EstimateThroughput. Let S_{t_0} and S_{t_1} denote their respective simulation states.*

Finally, let $S_{t_0}^{+k}$ and $S_{t_1}^{+k}$ denote the next states generated by function EstimateThroughput with k more iterations after S_{t_0} and S_{t_1}, respectively (for any value of k). If $S_{t_0} = S_{t_1}$, then $S_{t_0}^{+k} = S_{t_1}^{+k}$.

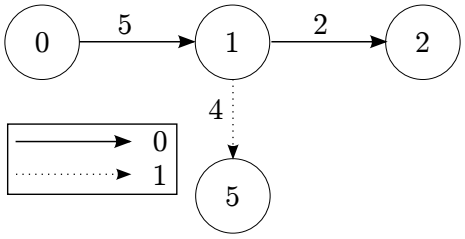
Proof This proof follows directly from applying Proposition 3.14 k times. We start with states S_{t_0} and S_{t_1} and apply Proposition 3.14 once, guaranteeing that states $S_{t_0}^{+1}$ and $S_{t_1}^{+1}$ are identical. We proceed a second time, but now applying Proposition 3.14 to $S_{t_0}^{+1}$ and $S_{t_1}^{+1}$, obtaining states $S_{t_0}^{+2}$ and $S_{t_1}^{+2}$, also guaranteed to be identical. After k steps, we reach states $S_{t_0}^{+k}$ and $S_{t_1}^{+k}$, which are also certainly equal to each other. □

Corollary 3.16 *Let t_0 and t_1 be two simulation times within a simulation performed by function EstimateThroughput. Let S_{t_0} and S_{t_1} denote their respective simulation states. If $S_{t_0} = S_{t_1}$, then the simulation has reached steady state. Moreover, if no other state S_{t_i} correspondent to a simulation time t_i such that $t_0 < t_i < t_1$ is equal to S_{t_0}, then the steady state cycle length is $t_1 - t_0$.*

Proof If $S_{t_0} = S_{t_1}$, Proposition 3.15 guarantees that, for any k, the states $S_{t_0}^{+k} = S_{t_1}^{+k}$ generated after k iterations of the function *EstimateThroughput* will be equal to each other. Since S_{t_1} is an achievable state generated after S_{t_0}, there is a number k' of iterations such that $S_{t_0}^{+k'} = S_{t_1} = S_{t_0}$. Therefore, state S_{t_0} will be generated indefinitely many times, along with all intermediate states between S_{t_0} and S_{t_1}, which means the simulation has reached the steady state.

If there is no other repetition of state S_{t_0} between t_1 and t_0, then the sequence of transmission events that happen between these two moments must be the shortest sequence that repeats itself indefinitely. In other words, this sequence must be the steady state cycle. By definition, its length is $t_1 - t_0$. □

These last results are at the core of the method used by function *EstimateThroughput* to detect whether the steady state has been reached in a simulation. Basically, function *EstimateThroughput* keeps track of every simulation state it generates at the end of each iteration. Whenever a new state is generated, function *EstimateThroughput* searches this history, checking whether this state has already been generated before. If a repeated state is found, the simulation is declared to be in steady state. Since function *EstimateThroughput* stops at the first occurrence of a repeated state, then the difference between the current time and time of the first occurrence of the repeated state is the cycle length. Similarly, the difference between the total number of packets delivery for all



Transmission Events	
t = 0: 0→1	t = 37: 1→2
t = 5: 1→5	t = 39: 0→1
t = 9: 0→1	t = 44: 1→5
t = 14: 1→2	t = 48: 0→1
t = 16: 0→1	t = 53: 1→2
t = 21: 1→5	t = 55: 0→1
t = 25: 0→1	t = 60: 1→2
t = 30: 1→2	t = 62: 0→1
t = 32: 0→1	t = 67: 1→2

Queue State for Node 1	
t = 0: 1	t = 37: 0, 1, 0, 0
t = 5: 1, 0	t = 39: 1, 0, 0
t = 9: 0, 1	t = 44: 1, 0, 0, 0
t = 14: 0, 1, 0	t = 48: 0, 0, 0, 1
t = 16: 1, 0	t = 53: 0, 0, 0, 1, 0
t = 21: 1, 0, 0	t = 55: 0, 0, 1, 0
t = 25: 0, 0, 1	t = 60: 0, 0, 1, 0, 0
t = 30: 0, 0, 1, 0	t = 62: 0, 1, 0, 0
t = 32: 0, 1, 0	t = 67: 0, 1, 0, 0, 0

Medium Access Priority	
t = 0: 0, 1	t = 37: 1, 0
t = 5: 1, 0	t = 39: 0, 1
t = 9: 0, 1	t = 44: 1, 0
t = 14: 1, 0	t = 48: 0, 1
t = 16: 0, 1	t = 53: 1, 0
t = 21: 1, 0	t = 55: 0, 1
t = 25: 0, 1	t = 60: 1, 0
t = 30: 1, 0	t = 62: 0, 1
t = 32: 0, 1	t = 67: 1, 0

Figure 3.5: Example of the simulation of a simple scenario with two flows and unlimited buffers. By continuing to execute function *EstimateThroughput* on this scenario, node one's buffer grows without bounds, indefinitely generating new different states.

flows at each occurrence of the repeated state is equal to the number of packets delivered during a steady state cycle. With these two values at hand, function *EstimateThroughput* can compute the steady state throughput (according to Proposition 3.6).

In terms of implementation, since a simulation state can be a large structure, it is important to employ efficient methods for comparing and storing it. In terms of storage, bitmaps can be employed for representing active links, reducing the size of the structure. Hashing can be used to speedup the comparison process, avoiding comparing complete states unnecessarily. Notice that the states must be stored along with their respective occurrence time, as well as the number of packets delivered up to that point, so that function *EstimateThroughput* is able to retrieve that data in order to compute the throughput.

3.2.4.1 Limited *versus* Unlimited Buffers

One question that arises, though, is whether any instance provided to function *EstimateThroughput* necessarily constitutes a simulation scenario that eventually reaches the steady state. In other words, are there cases in which function *EstimateThroughput* enters an infinite loop looking for a repetition of states that will never happen?

We will tackle that question by looking at a correlated issue: given a generic instance

provided for function *EstimateThroughput*, how many reachable states are there? The answer lies in the structure of the simulation state. As discussed before, a simulation state is comprised of three components: currently active links, the current medium access priority list, and the state of the nodes' buffers. The length of the list of active links is clearly bounded by the number of network links. In fact, since a reachable state is also a valid state, then this number is bounded by the size of the largest independent set of the conflict graph. The medium access priority link is also bounded by the number of nodes that are part of some path in the evaluated path set. While the sizes of these two first components are clearly finite, the state of the nodes' buffers is not limited by any of the definitions presented so far. Under these conditions, it is possible to demonstrate the following proposition:

Proposition 3.17 *Let G, F, and C denote, respectively, a graph, a flow set, and a path set used as an input for the function EstimateThroughput. If the simulation allows nodes to have unlimited buffers, there may be an infinite number of reachable states.*

Proof We prove this proposition by presenting an example. Consider the scenario illustrated in Figure 3.5. It represents a simpler version of the scenario in Figure 3.4 with only two flows and less nodes. In the initial state, the queue for node 1 has a single packet, the first packet of flow $1 \Rightarrow 5$. As the simulation progresses, the queue's length increases, because node 1 needs more bandwidth than node 0 (it relays two flows), but it receives the same number of transmission opportunities. Since the rules that govern the medium access priorities do not change and there are no other elements to the simulation (flows or nodes), this trend is maintained and node 1 keeps receiving more packets than it is capable of forwarding.

Specifically, for every opportunity node 1 has of transmitting a packet of flow $1 \Rightarrow 5$, the number of packets from flow $0 \Rightarrow 2$ waiting on its buffer is increased by 1: 1 at $t = 5$, 2 at $t = 21$, 3 at $t = 44$, and so on. Therefore, as long as the simulation continues, new different states are generated, resulting in an infinite number of reachable states. □

Corollary 3.18 *If a simulation allows unlimited buffers, there are instances in which function EstimateThroughput enters an infinite loop.*

Proof This proof follows directly from the fact that Proposition 3.17 guarantees the existence of instances with an infinite number of reachable states. For those instances,

since new states continue to be reached, simulation never reaches steady state. Thus, the main loop of function *EstimateThroughput* never ends. □

From these two last results, it becomes evident that a simulation with unlimited buffers is not viable for the purposes of this thesis. In order to be useful for employment in a route selection algorithm, the simulation must have a guaranteed finite execution time. The question, thus, is whether limiting the size of the buffers for the simulation guarantees this finite execution time regardless of the instance. For this reason, we present the following result:

Proposition 3.19 *Let $G = (V, E)$, F, and C denote, respectively, a graph, a flow set, and a path set used as an input for the function EstimateThroughput. Assume G has a finite number of edges and vertices. If the simulation employs a hard limit for the maximum number of packets that can be stored in each node's buffer, there is always a finite number of reachable states.*

Proof Let QL_{max} denote the maximum length allowed for the queue of a node in the simulation in packets. For a given node, the total number of possible states for its buffer is limited by $(|F| + 1)^{QL_{max}}$. Since a simulation cannot have more than $|V|$ nodes, the number of different states considering all simulation nodes must be less or equal to $\left[(|F| + 1)^{QL_{max}}\right]^{|V|}$.

Let AL_{max} denote the maximum number of simultaneously active links for the path set C. Then, the following inequality holds:

$$AL_{max} <= |E| \tag{3.20}$$

In that case, the maximum number of different lists of active links in a given simulation is bounded by $2^{|E|}$.

Let PL_{max} denote the maximum length of the medium access priority list. Then, the following inequality must be true:

$$PL_{max} <= |V| \tag{3.21}$$

Similarly to the list of active links, the number of different medium access priority lists cannot exceed $2^{|V|}$.

Under these circumstances, the maximum number of different valid simulation states is bounded by $2^{|V|} \times 2^{|E|} \times \left[(|F| + 1)^{QL_{max}}\right]^{|V|}$, which must be a finite number. □

Corollary 3.22 *If a simulation only allows limited buffers, function EstimateThroughput always finishes in finite time.*

Proof Since Proposition 3.19 guarantees that the number of states is always finite, then function *EstimateThroughput* must eventually find a repeated state, at which point the steady state of the simulation is reached, and the function returns. □

Therefore, as long as we fix a maximum length for the buffers of the nodes, function *EstimateThroughput* is guaranteed to find the steady state throughput in finite time. For the purposes of this thesis, we always consider the maximum size of the nodes' buffer to be twice the maximum number of flows that pass through a node in the given path set.

3.2.4.2 Throughput and Packet Loss

As explained in Section 3.2.3, function *EstimateThroughput* requires that the graph G of the network provides information regarding the delivery probability of each link, so that it is possible to compute an estimate for the link transmission delay. However, the information of the delivery probabilities is also employed by function *EstimateThroughput* for a different purpose.

An obvious drawback of using a hard limit for the estimate of the link transmission delay is the fact that links with very different qualities (in terms of delivery probability) can be assigned similar costs. Consider as an example a link $a \to b$ with $d_{a\to b} = d_{b\to a} = 0.4$. According to Equation 3.9, the transmission delay for that link is 3.14. Now suppose there is another link $c \to d$ with $d_{c\to d} = d_{d\to c} = 0.2$. For this link, the transmission delay will be 3.76. While there is obviously a difference between the values of transmission delay, it is not a large difference considering that one delivery probability is half of the other (for comparison, under the ETX metric, those costs would be 6.25 and 25, respectively).

The most important difference between those two links is not the expected transmission delay, though. Both links are expected to have delays close to 4, given that both have low delivery probabilities and 4 is the link layer retransmission limit. However, the probability that a packet transmitted through each of the links is actually received at the other end with, at most, 4 transmissions is much different for each link. For link $a \to b$, this probability is $1 - (1 - 0.4)^4 = 0.87$, against $1 - (1 - 0.2)^4 = 0.59$ for link $c \to d$.

Differently from other works, in this thesis, we argue that transmission delay and packet loss rates must be accounted for separately. While the delivery probability of a

link has influence over the transmission delay (and, in the case of the routing algorithm presented in this thesis, over the amount of time a link interferes with other links), it also causes a separate effect on the actual probability that the packet is received. Classical routing metrics tend to consider only one of those effects: either they model the transmission delay (such as ETX and ETT), or the packet loss rates (such as ML).

To cope with this secondary effect of the delivery probabilities, function *EstimateThroughput* computes, for each path of the candidate path set, an estimate for its end-to-end packet delivery probability by applying the following expression:

$$p[\text{packet is lost on path } \mathcal{P}] = 1 - \prod_{a \to b \in \mathcal{P}} 1 - (1 - d_{a \to b})^4 \tag{3.23}$$

Notice that Equation 3.23 only uses the link delivery probability in the forward direction. In other words, we do not account for the reception of the acknowledgment frame for each link. The reason is that, for a given link $a \to b$, if node b receives the packet in one of the transmission attempts, it will proceed with the forwarding (or reception) of the packet regardless of whether node a received the acknowledgment frame. If by any chance, node a does not receive an acknowledgment frame relative to a successful transmission attempt, the only consequence is that a will (possibly) perform a retransmission. That consequence, however, is already captured in the calculation of the link transmission delay.

During a simulation, once a packet is delivered to its final destination, function *EstimateThroughput* increments a counter to keep track of how many packets have already been delivered (which is used once the steady state is found for computing the throughput). Instead of simply adding 1 to the number of delivered packets, function *EstimateThroughput* increments this counter by the end-to-end delivery probability of the respective path, as given by the complement of Equation 3.23. This is equivalent to multiplying the number of packets delivered for each flow by their respective end-to-end delivery probability (*i.e.*, the expected number of delivered packets for that flow) and summing the resultant values.

3.2.5 Adding Network Coding to the Model

So far, all the details presented about the function *EstimateThroughput* consider only the case of simulating a network without network coding. Nevertheless, the structure of our proposed solution lends itself well to the incorporation of the network coding case, as will

be discussed in this section.

In order to add network coding to the model developed so far, it is necessary to alter only three aspects of function *EstimateThroughput*: the selection of packets for transmission (whenever a node gains access to the wireless medium), the computation of the transmission delay for coded packets, and the computation of the end-to-end packet delivery probability.

The selection of packets for transmission follows the same basic algorithm used by COPE [56]. Whenever a node gets the opportunity to transmit, the first overall packet in its buffer is selected for transmission. Function *EstimateThroughput*, then, looks for coding partners among the remaining packets. If there are no packets that can be coded together (according to the requirements specified by COPE), the packet is transmitted natively, respecting all the details previously presented in this chapter.

Assuming, however, that there are coding partners, it is necessary to model the transmission of a coded packet. While Chapter 4 discusses some alternative methods for the transmission of such packets, in this chapter we limit the scope to a basic method known as *Simple Broadcast*. In the Simple Broadcast, a coded packet is mapped to a broadcast frame at the link layer. This frame is transmitted in broadcast for all neighbors that should filter whether or not the coded packet was originally addressed to them.

Notice that a packet transmitted using the Simple Broadcast method has always a fixed transmission delay of 1. Since there are no acknowledgment frames to report the success of the transmission, there are also no retransmission attempts. Therefore, if a node transmits a coded packet, function *EstimateThroughput* schedules the end of the transmission to one unit of time ahead, instead of relying on Equation 3.9.

Another difference between the transmission of a native packet and of a coded packet using Simple Broadcast is the probability that the packet is received at the other end of the link. While in a unicast transmission of a native packet there are multiple retries, increasing the probability of success, with Simple Broadcast there is always a single attempt, making the probability of the coded packet being received by a given node equal to the delivery probability of the respective link. For example, suppose node a codes together two packets P_b and P_c, which have as next hops nodes b and c, respectively. The probability that node b receives the coded packet is, then, given by $d_{a\to b}$, while for node c this probability is $d_{a\to c}$.

Moreover, it is still necessary to take into account whether nodes b and c are ac-

tually capable of decoding the message to retrieve their respective originally addressed native packets. Each packet coded together with the packet addressed to each node, must be known by that node so that decoding is possible. Given the probabilities of a node knowing each of the packets necessary for decoding, the product of these values can be computed as the probability of that node being able to decode the message. Function *EstimateThroughput* employs the same methodology used by COPE to compute the decoding probability for each receiver of a coded transmission (as explained in details in Chapter 2).

Under these circumstances, the probability of a coded packet being successfully transmitted from a node a to a node b is given by:

$$p[\text{coded packet is received by } b \text{ from } a] = d_{a \to b} \times p[\text{b can decode}] \tag{3.24}$$

Notice, however, that, during the course of a simulation, packets from a given flow can traverse a certain hop of the path currently being evaluated both natively and coded with other packets. Since the success probabilities involved in each case may be different, there is not a single end-to-end success probability associated with each path. Therefore, when a packet is delivered to its final destination it would be necessary to evaluate in what conditions each of the hops of the path were traversed in order to compute the end-to-end probability for that specific packet.

To cope with that issue, function *EstimateThroughput* appends to each simulated packet a delivery probability that is initialized at 1 when the packet is created. As the packet traverses the evaluated path, for each link $a \to b$, that probability is multiplied by the expression:

$$p[\text{packet is received by } b] = \begin{cases} 1 - (1 - d_{a \to b})^4 & \text{if native transmission} \\ d_{a \to b} \times p[\text{b can decode}] & \text{otherwise} \end{cases} \tag{3.25}$$

When a packet reaches its final destination, its delivery probability field contains the delivery probability for the specific conditions it has faced during the traversal of the path. This value can be added directly to the total number of packets delivered, in order to be used for computing the steady state throughput.

For the remainder of this thesis, we will consider two separated versions of our routing

selection algorithm: one that does not consider network coding, hereinafter referred to as IAR (Interference Aware Routing), and another that does incorporate network coding in its model, hereinafter called ICAR (Interference and Coding Aware Routing). During our evaluations, we will consider both the cases of coding enabled and disabled networks.

3.2.6 Asymptotic Complexity Analysis

One important characteristic of both IAR and ICAR is that they do not employ the traditional Dijkstra algorithm for path selection. Instead, they use a completely novel algorithm that has the advantage of evaluating paths for all flows together, instead of looking at each flow separately. As expected, though, there is a trade-off in doing so, as this novel algorithm is more expensive in computational terms than the classical Dijkstra. While this chapter still has not presented all components of IAR and ICAR (Section 3.3 discusses function *GenerateCandidates*), a core element of these algorithms (function *EstimateThroughput*) was already described in all its details. If the execution of function *EstimateThroughput* is not viable in practice due to high computational costs, then the routing selection algorithms as a whole are not feasible as well. For this reason, in this section we analyze the asymptotic complexities involved in executing function *EstimateThroughput*.

The function, described in Algorithm 3.2, can be divided in two main stages: an initialization and a main loop. We start by analyzing the time complexity of the initialization.

During this first stage, the main control variables (time t, the set of active links AL, and the set of nodes that wish to transmit WT) are initialized. This step clearly takes constant time. From that point until line 6, the initial packets for each flow are generated, placed on the buffers of the source nodes which, in turn, are added to the set of nodes that wish to transmit. All those steps take time proportional to the number of flows $|F|$. Another initialization that is done implicitly is the construction of the conflict graph, which is done by the procedure depicted in Algorithm 3.3. This procedure checks every link of every path of the currently evaluated path set against each other, looking for conflicts. In the absolute worst case, all network links will be used in the path set, leading this procedure to have a worst case complexity of $O(|E|^2)$.

At that point, the initial state of the simulation has already been build. From lines 7 to 18 the second state is built by the scheduling of the first transmissions. To that end, set WT is traversed (actually, the structure that is traversed is a linked list that is sorted

according to the medium access priority) and, for each node, it is necessary to check whether the required link l is blocked. Assuming the worst case, in which all network links are in use and l conflicts with all of them, even a naive implementation would result in a complexity of $O(|E|^2)$. If a link is not blocked, it is necessary to compute the transmission delay of the link and, eventually, remove the node from the set WT, both actions which can be performed in constant time. A caveat at this point, though, is that, if network coding is used, we also have to look for a coding partner for the packet. Since we employ the same basic procedure proposed by Katti *et al.* [56] for COPE, the time complexity is a cubic function of the number of neighbors of the node (for which the node has packets to transmit). In the simulation that value is bounded by the number of flows, resulting in a worst case complexity of $O(|F|^3)$ for that case. By considering the complexities of all those steps, the overall asymptotic complexity of the repetition executed between lines 7 and 18 is $O(|V| \cdot |E|^2 \cdot |F|^3)$.

By the time function *EstimateThroughput* reaches line 18, the initialization stage is finished, having accumulated an asymptotic complexity of $O(|V| \cdot |E|^2 \cdot |F|^3)$. At the next line, the function begins its main loop, which is composed of two nested loops: one for managing packets that are arriving, and another for scheduling new transmissions.

The loop that manages packets that are arriving iterates through all active links, checking which ones are scheduled to finish their transmissions at the current time. For each link under this condition, a series of simple set and buffer manipulations are performed (*e.g.*, packets are placed at nodes' buffers, nodes are added to or removed from the WT set). By using adequate data structures, each iteration can, thus, be executed in constant time, resulting in a total worst case time complexity for the complete loop of $O(|E|)$ (assuming all network links are currently under use and scheduled to finish at the same time).

The loop that is responsible for scheduling new transmissions executes the same steps of lines 7 through 18, resulting in the worst case time complexity: $O(|V| \cdot |E|^2 \cdot |F|^3)$.

Another component that introduces complexity to the main loop is the test of the stop criterion. Implicitly, that test is done by assembling the current network state (gathering information regarding the active transmissions, the current state of nodes' buffers and the current medium access priority list) and comparing it to previous states. The assembly of the current state can be done in worst case time $O(|E| + |V| + |F| \cdot |V|)$. The comparison, though, is more complicated to evaluate since it depends on the quality of the hash function used to store the states, as well as the number of states generated until the

steady state is found. It is also important to notice that the number of iterations of the main loop is also equal to the number of states generated through the simulation. Let GS denote the number of generated states until the steady state is reached in a simulation. The total asymptotic time complexity of the main loop is bounded by $O(|V| \cdot |E|^2 \cdot |F|^3 + (|E| + |V| + |F| \cdot |V|) \cdot GS + GS^2)$, which is clearly the overall dominant component of the time complexity of the complete algorithm of function *EstimateThroughput*.

It is possible to put that complexity only in terms of $|V|$ (the number of nodes in the graph) and GS, by using $|V|^2$ as an upper bound for both $|E|$ and $|F|$. In that case, we obtain $O[|V|^{11} + (|V|^3 + |V|^2 + |V|) \cdot GS + GS^2]$. Notice that we have not employed particularly tight upper bounds for deriving this complexity expression, which possibly resulted in a higher exponent for the term $|V|$ than actually necessary. The goal of this section, though, is simply to demonstrate that, if the number GS of generated states is polynomial on the number of network nodes, so will be the overall asymptotic time complexity of function *EstimateThroughput*.

Regarding the value of GS, during the proof of Proposition 3.19 an upper bound was provided for the maximum number of different valid simulation states. While the number of reachable simulation states tends to be lower than the number of valid simulation states and the bound provided for that proof was also not particularly tight, it is worth noticing that the expression found includes exponential functions on $|V|$. Indeed, we performed preliminary experiments with function *EstimateThroughput* and found that, for some instances, the number of generated states was high, leading to execution times in the order of seconds, rendering its application on real time protocols unfeasible. For this reason, in the next section we present optimizations and heuristics for reducing the number of generated and stored states during the execution of function *EstimateThroughput*.

3.2.7 Optimizations and Heuristic Stop Criteria

As discussed in the previous section, the challenge for the practical employment of function *EstimateThroughput* lays on the number of generated states necessary to find the steady state. In order to allow this function to be used in practice with generic instances, we propose the usage of a cycle detection procedure comprising three different stop criteria — one optimal and two heuristic. Each criterion uses a different condition to evaluate whether or not the main loop of function *EstimateThroughput* should stop. When at least one of them flags that the loop should stop, function *EstimateThroughput* returns the computed throughput. The idea is to complement the search for the steady state with

other stop criteria that allow function *EstimateThroughput* to find good approximations for the long term aggregated throughput, while maintaining the number of iterations in an acceptable level.

The optimal stop criterion is simply the one originally presented in Section 3.2.4: states are generated and stored at the end of each iteration of the main loop. Whenever a new state is generated, function *EstimateThroughput* looks for an equal state previously stored. The simulation is declared to be in steady state if this search is successful. In this case, the main loop is stopped and the exact value for the steady state throughput is returned. While our preliminary results show that this method can lead to unacceptable execution times for certain instances, they also show that in many practical cases it can rapidly detect the steady state, while providing the exact result.

This optimal criterion, however, can be improved with a slight modification. When the steady state is reached, the steady state cycle will be repeated indefinitely. Each repetition of this cycle may correspond to one or more iterations of the main loop of function *EstimateThroughput*, which is equivalent to say that one or more states will be generated for each repetition of the cycle. If we knew before hand that a given state would be generated during the steady state cycle, we could concentrate only on finding that specific state. While we do not know such a state, we do know some characteristics of some of the states that will be generated during the steady state cycle. One such characteristic is that the link correspondent to the first hop of each flow will be used in at least one of the states generated during the steady state cycle. That is true because the source nodes for the flows will always generate new packets that will eventually be transmitted through the first link of the path.

It is reasonable to assume function *EstimateThroughput* will always be called to simulate a scenario with at least one flow (otherwise, there is nothing to be simulated). Therefore, for any input received by this function, it is possible to conclude that, whatever the steady cycle may be, it will comprise at least one reachable state containing the first link of the first path of the path set beginning to transmit. With that information, function *EstimateThroughput* can store and compare states only for iterations in which that link was scheduled to start transmitting.

While that optimization does not reduce the number of iterations of the main loop (all states still need to be generated until the simulation reaches the steady state), it does reduce the number of states that are stored. This not only reduces the amount of memory used by the function, but it also reduces the complexity of checking whether a given state

was previously stored (because there are less stored states). Moreover, it reduces the number of times function *EstimateThroughput* has to perform such search.

The first heuristic criterion is based on an observation made on our preliminary results: the steady state cycle is usually comprised by at least one event of end-to-end packet delivery for each flow (in other words, usually no flows suffer starvation). With that in mind, we present the following definition:

Definition 3.26 *Delivery Cycle of a Simulation. A delivery cycle of a simulation is the shortest sequence of events within which at least one packet from each flow is delivered to its final destination.*

The delivery cycle of a simulation can be seen as the heuristic version of the steady state cycle. It is an attempt to approximate the concept of steady state cycle in order to accelerate the process of finding the steady state throughput. The idea of the first heuristic mechanism is, thus, to keep track of delivery cycles, using the throughput during those cycles to approximate the steady state throughput.

During our preliminary results, we were able to verify that, as simulations approached their steady state, the average throughput of the delivery cycles approached that of the steady state cycle. Hence, as the simulation unfolds, for each new delivery cycle found, function *EstimateThroughput* computes the average cycle throughput. That value is then applied to an exponentially weighted average of the throughputs of all delivery cycles found so far with weight α given to the newest sample. After β delivery cycles are found, this heuristic criterion declares the steady state found and the steady state throughput is approximated by the current value of the weighted average. This criterion can also be reached if the percentage difference between the current and previous values of the weighted average is less than γ. The option of using the current value of the weighted average instead of the average throughput of the last cycle has the goal of minimizing eventual variations of the delivery cycles. Although the delivery cycles tend to approach the steady state cycle, some fluctuations might occur. Hence, we consider the average to be a more reliable estimator.

Our preliminary results show that these first two stop criteria are sufficient to guarantee feasible execution times for the majority of the evaluated cases. There are, however, some special cases in which the execution times grow beyond what is acceptable for real time routing, even considering the first heuristic criterion. For that reason, we employ a second heuristic criterion that is based simply on counting the number of generated states.

When that number reaches a specified threshold λ, the function is stopped and the steady state throughput is approximated by the average throughput of the complete simulation up to that point. While this criterion uses the poorest approximation for the steady state throughput, it provides a hard limit based on a constant number of saved states, which assures that the total asymptotic time complexity of function *EstimateThroughput* is polynomial on the number of network nodes.

Hereinafter, for all results and discussions regarding function *EstimateThroughput*, we will use the values of $\alpha = 0.8$, $\beta = 100$, $\gamma = 1\%$, and $\lambda = 1000$, all chosen after preliminary experiments. While we do not provide a performance evaluation of the impact of each of these parameters on the precision and execution time of function *EstimateThroughput*, we argue that they do result in satisfactory performance in both aspects, as shown in Chapter 5.

We do, however, provide a rationale for the choice of values for each of those parameters. The choice for $\lambda = 1000$ was motivated purely by time constraints. During our preliminary experiments, this value resulted in the route selection procedure being always executed in less than one second for all evaluated instances in our computing environment. When $\beta = 100$, all flows have delivered at least 100 packets. In other words, in this case function *EstimateThroughput* would have a sample of at least 100 packets *per flow* to approximate the long term aggregated throughput. The idea of parameter γ is to evaluate whether the delivery cycles are converging. Therefore, lower values of γ result in stricter rules for declaring convergence. It is interesting to notice that we opted for using $\alpha > 0.5$, *i.e.*, we give more weight to the last sample, than to the historical average. This increases the variability of the moving average when the cycle throughput samples are also very variable. Together, the values chosen for α and γ impose a quite strict constraint for the stop criterion to be met.

3.3 Candidate Generation Method

Up to this point, this chapter has mostly discussed function *EstimateThroughput*, that receives a given path set (a candidate solution for the routing problem) and outputs an estimate for the long term aggregated throughput. However, as shown by Algorithm 3.1, IAR and ICAR also make use of a second auxiliary function called *GenerateCandidates*. This function receives as arguments the graph that represents the network topology and the set of currently active flows. It, then, outputs a list of candidate solutions to be

evaluated by function *EstimateThroughput*. This section, thus, focuses on discussing how to implement such candidate generation procedure.

A first approach to function *GenerateCandidates* would be to simply output all combinations of all possible paths for all flows. This can be implemented by the employment of Yen's algorithm [103]. This algorithm can find the k shortest loop-less paths (in terms of the sum of edges) between two given nodes of a graph with time complexity of $O(k \cdot |V| \cdot (|E| + |V| \cdot log\ |V|))$. It can also be adapted to find all possible loop-less paths for a given pair of nodes (argument k is set to infinite). In that last case, the time complexity remains the same, except for the fact that k represents the total number of available loop-less paths. By using Yen's algorithm, it is possible to write the brute force version of the function *GenerateCandidates* shown in Algorithm 3.5.

Algorithm 3.5 Brute force approach to function *GenerateCandidates*.

```
1: function BruteForceGenerateCandidates(Graph G, Flow set F)
2:     for all flow f ∈ F do
3:         pathList_f ← Yen(G, src(f), dst(f), ∞)
4:     end for
5:     Return all possible combinations with the paths of the path lists.
6: end function
```

One issue with the brute force version of function *GenerateCandidates* is that it may generate an exponential number of candidate solutions with respect to the number of network nodes (for a complete graph, for example, any two pairs of nodes are connected by approximately $e \cdot (|V| - 2)!$ paths). In that case, it will be necessary to call function *EstimateThroughput* an exponential number of times, which will render the usage of the routing algorithms unfeasible for real time routing. In the next subsection, an alternative version for the function *GenerateCandidates* will be presented.

3.3.1 Heuristic Candidate Generation

In this section, we present an heuristic version of the candidate generation procedure that greatly reduces the computational cost of IAR and ICAR. Called *Perturbation Heuristic*, this candidate generation procedure receives an initial path set and perturbs the solution in order to obtain a new candidate. To this end, this heuristic executes two different disturbing routines: one specific for avoiding interference by using traditional strategies (*i.e.*, by separating flows further apart), and another for trying to find coding prone path sets (by actively making flows cross with one another).

Algorithm 3.6 Overview of the Perturbation Heuristic candidate generation procedure.

```
1: function PerturbationHeuristicGenerateCandidates(Graph G, Flow set F)
2:     initialList ← PerturbationHeuristicNonCoding(G, F)
3:     finalList ← PerturbationHeuristicCoding(G, F, initialList)
4:     Return finalList
5: end function
```

Algorithm 3.6 presents an overview of how this heuristic works. It delegates the task of generating candidates to two sub-routines: *PerturbationHeuristicNonCoding* and *PerturbationHeuristicCoding*. The function *PerturbationHeuristicNonCoding* generates a number of candidates considering only the possibility of placing flows further apart in order to decrease interference. After it returns the list of generated candidates, this list is passed as an argument for function *PerturbationHeuristicCoding* that will perturb each candidate by forcing flows to cross, trying to create coding opportunities. By adding these newly generated candidates to the list, the final list of candidates is obtained and finally returned.

Algorithm 3.7 illustrates function *PerturbationHeuristicNonCoding*. The idea of this function is to assemble path sets finding paths for one flow at a time, but avoiding nodes that suffer much interference from other previously selected paths. To this end, it loops through a maximum number of iterations generating permutations of the flows. Each permutation establishes an order in which a new path set will be built (*i.e.*, an order in which paths will be found for each flow).

The process of building a path set starts by reseting some control variables and structures. Associated with each node of the graph G, there is a data structure that determines whether the node is *enabled* or *disabled*. When a node is disabled, all operations performed over the graph behave as if the node did not exist. For example, this data structure allows the algorithm to look for paths between given source and destination nodes without using the disabled nodes. At the beginning of each iteration, all nodes are re-enabled. The algorithm also keeps track of the *score* of each node: an integer variable that stores the number of links in the current path set that interfere with the given node (if a link is found in multiple paths in the path set, it is counted multiple times for each interfering node). Notice that, in this heuristic, we employ the concept of a link interfering with a node. We say that a link l interferes with a node a if a is not able to transmit (in any of its links) if l is active. Finally, associated with each flow there is a *tolerance level*. Whenever the algorithm tries to find a route for a given flow, it first disables all nodes with score higher than the tolerance level for that flow.

Once these main data structures are initialized, the algorithm resorts to a nested loop that will perform the actual assembly of the path set (lines 10 to 30). Every iteration of this loop refers to the search for a path for the *i-th* flow of the current permutation α. The search starts by computing the score of all network nodes, considering the current state of the path set being assembled. Once all nodes have their scores computed, nodes with score higher than the current tolerance level for the flow are disabled. The Dijkstra algorithm is then called in order to find a path for the current flow. If such path exists, it gets added to the current path set. If the path set is complete, it is added to the output list of candidates. Otherwise, the loop is repeated for the next flow in the permutation. If a path cannot be found for the current flow, the algorithm tries to relax the graph restrictions by increasing the tolerance for that flow and re-executing the loop. If the tolerance is already higher than any node's score, then it is impossible to find a path connecting the current flow.

The generation of multiple different permutations allows the algorithm to find potentially different candidates. If a single permutation was used, then the path found for the first flow would always be the optimal path according to the ETX metric. By using multiple permutations, flows are traversed in different orders, possibly resulting in multiple different candidates.

Algorithm 3.8 shows the steps executed by function *PerturbationHeuristicCoding* to generate new coding prone candidates based on the initial candidate list generated by function *PerturbationHeuristicNonCoding*. For each candidate of the original list, the function tries — up to *max* times— to generate a coding prone version of it. To this end, the function first finds the two closest nodes a and b contained in the path set, such that the nodes belong to different paths and are different. The function proceeds by finding the closest path that connects a and b (according to ETX). Finally, the middle node c is identified, *i.e.*, the intermediate node in the shortest path between a and b such that the cost from a to c in that path is closest to half the total cost of the path. The idea of finding this node c is that it represents the middle point between the two paths P_1 and P_2 that contain nodes a and b. Node c is then used as a reference for joining the two paths. This is done by finding the paths between the source nodes of each flow and c, as well as from c to the destination nodes. The prefixes and suffixes for each path are then concatenated, forming the new pair of paths.

Notice that, although Algorithm 3.8 makes multiple calls to the Dijkstra Algorithm, those can be replaced by queries to a pre-computed distance table assembled, for example,

with the Bellman-Ford Algorithm. This alternative possibly reduces the overall computational cost of the algorithm due to the large number of queries. Specifically, in the implementations of this heuristic used in the results shown in this thesis, that was the chosen strategy.

As one final note, when this heuristic is used for IAR, only the first step — *i.e.*, the execution of function *PerturbationHeuristicNonCoding* — is executed. Since IAR is oblivious to network coding, it does not make sense to generate the second set of coding prone candidates.

During the development of this thesis, other alternative heuristics were also explored. One such alternative was an heuristic that would generate only the k best paths for each flow (according to the sum of the ETX values of each link), thus resulting in a maximum of k^f candidates. The problem of this approach is that, as the number of flows increase, the total number of candidates becomes exponentially high. A second approach was an heuristic that would take the paths output by the Dijkstra Algorithm as an initial solution. At each new iteration, the solution would be modified, resulting in a new candidate. To this end, the heuristic would build a conflict graph, find an approximation for the maximum clique and would substitute the worst link that belonged to that clique for an alternative path. By breaking the worst cliques, this heuristic potentially created candidates with less interference. However, since this heuristic did not take network coding directly in consideration, it usually did not generate good coding prone candidates. Overall, our preliminary results indicated that the Perturbation Heuristic offered the best trade-off between computational complexity and quality of generated candidates.

Algorithm 3.7 Pseudocode of the *PerturbationHeuristicNonCoding* function.

```
1: function PerturbationHeuristicNonCoding(Graph G, Flow set F, Maximum
   Number of Iterations max)
2:     Zero outputList
3:     while max > 0 do
4:         α ← new permutation of the flows
5:         currentPathSet ← ∅
6:         for i = 1 → |F| do
7:             tolerance_i ← 0
8:         end for
9:         i = 1
10:        while i ≤ |F| do
11:            Compute nodes' score based on elements of currentPathSet
12:            Re-enable all nodes in graph G
13:            Disable all nodes with score higher than tolerance_i
14:            f ← α[i]
15:            P ← Dijkstra(G, src(f), dst(f))
16:            if P exists then
17:                currentPathSet ← currentPathSet ∪ {P}
18:                if i < |F| then
19:                    i ← i + 1
20:                else
21:                    Add currentPathSet to outputList
22:                end if
23:            else
24:                if tolerance_i is higher than all nodes' score then
25:                    Return null
26:                else
27:                    tolerance_i ← tolerance_i + 1
28:                end if
29:            end if
30:        end while
31:        max ← max − 1
32:    end while
33:    Return outputList
34: end function
```

Algorithm 3.8 Pseudocode of the *PerturbationHeuristicCoding* function.

function PERTURBATIONHEURISTICCODING(Graph G, Flow set F, Candidate List CL, Maximum Number of Iterations max)
  for all path sets PS in CL **do**
    for $i = 1 \rightarrow max$ **do**
      Find the two closest nodes a and b s.t. a and b belong to different paths $\mathcal{P}_1, \mathcal{P}_2 \in PS$ and $a \neq b$
      $path_{a,b} \leftarrow Dijkstra(G, a, b)$
      $targetCost \leftarrow cost(path_{a,b})/2$
      Find node c in $path_{a,b}$ s.t. cost from a to c is closest to $targetCost$
      $prefix_1 \leftarrow Dijkstra(G, src(\mathcal{P}_1), c)$
      $suffix_1 \leftarrow Dijkstra(G, c, dst(\mathcal{P}_1))$
      $\mathcal{P}_1^* \leftarrow$ concatenate $prefix_1$ and $suffix_1$ and remove any loops.
      $prefix_2 \leftarrow Dijkstra(G, src(\mathcal{P}_2), c)$
      $suffix_2 \leftarrow Dijkstra(G, c, dst(\mathcal{P}_2))$
      $\mathcal{P}_2^* \leftarrow$ concatenate $prefix_2$ and $suffix_2$ and remove any loops.
      Create PS^* with all paths from PS, replacing $\mathcal{P}_1$ and $\mathcal{P}_2$ for $\mathcal{P}_1^*$ and $\mathcal{P}_2^*$
      Add PS^* to CL, if it does not exist.
      $PS \leftarrow PS^*$
    end for
  end for
  Return CL
end function

Chapter 4

Practical Aspects

Chapter 3 presented IAR and ICAR, two novel route selection algorithms. However, that chapter only discussed theoretical aspects, such as the pseudocode for the routines involved in computing the best paths. Since IAR and ICAR take a different approach to the problem of route selection (in comparison to the traditional solutions of the literature), there are non-trivial practical issues that must be addressed in a real implementation. In this chapter we list those issues and present solutions to allow the usage IAR and ICAR in real networks.

4.1 Flow Detection

The most fundamental difference between the proposals presented in this thesis and the traditional routing algorithms is the fact that IAR and ICAR must know the set of flows currently active in the network. Traditional solutions based on classical algorithms such as Dijkstra's or Bellman-Ford are oblivious to which flows are actually active: they simply find routes connecting every possible pair of nodes in the network. On the other hand, IAR and ICAR must know exactly which flows are in use, so that paths with low inter-flow interference can be found. To this end, a practical implementation of these route selection algorithms must be able to keep track of all current flows, as well as detecting whenever a new flow is started and released.

In this thesis, we propose the following method for achieving this goal. Each node should maintain a *local flow table*, *i.e.*, a table of the currently active flows originated at the local node. Each entry of the local flow table has a timestamp associated with it, indicating the time when the last packet of that flow has been sent. Whenever a packet is routed, the node checks whether the packet was generated locally (*i.e.*, by the node

itself). If that is the case, the node adds or updated the correspondent entry in its local flow table. Periodically, the table is traversed so that old entries can be removed. An entry is considered to be old if the difference between the current time and its timestamp is greater than a configurable parameter, hereinafter called *local flow table timeout*.

Since in this thesis we assume the usage of a link state routing protocol, periodically each node generates a control packet containing information regarding the locally known network links. This packet, usually called a *topology packet*, is then diffused through the network so that all nodes know that set of links. This same control packet can be used to propagate the information contained in the local flow table for all other network nodes. Nodes then store a second table, called a *global flow table*, which contains information regarding all active flows (network wise). This table is built with the information received in the topology packets and each entry has a lifetime that is equal to the lifetime of the topology information of its correspondent packet.

Similarly to a traditional link state routing protocol, that relies on coherence of the link state information with the current network state and among all network nodes, IAR and ICAR rely as well on the coherence of the global flow table. It is, thus, susceptible to similar issues (*e.g.*, route loops) whenever that coherence is not maintained.

One aspect that deserves special attention is the order in which flows are passed to the route selection routines of IAR and ICAR. As explained in Chapter 3, it is assumed that there is some implicit order in which the flow set is traversed by those routines. In practice, the flow set is passed to the route selection routines as an array or a linked list, providing that implicit order. Notice, however, that the solutions found by IAR and ICAR can be dependent on that order, in some cases. Therefore, in order to guarantee that all nodes will find the same set of routes as the best solution (assuming they all have the same data on the state of the network), it is necessary that all nodes pass the flow set with the same order to the route selection routines. That can be easily achieved by maintaining the list of flows sorted by some deterministic criterion. One such criterion is to sort the flows by source node address, with ties solved by the destination node address.

4.1.1 Initial Routes for New Flows

Another issue related to flow detection is the setup of initial routes. Obviously, a flow can only be detected once its first packet is routed by the source node. Even if we assume the routing protocol running at the source node will immediately update its local information and run the route selection routines for IAR and ICAR, other network nodes will only

have knowledge of the new flow once the next topology packet reaches them. Since in this thesis we do not assume the usage of source routing, the first packets of each new flow would be lost until all nodes were aware of its existence.

To avoid this issue, we propose the usage of two different routing tables: a main table and a fallback table. The main table is used to store the routes yielded by IAR and ICAR, while the fallback table is built based on the traditional ETX metric and the classical Dijkstra Algorithm. For each packet to be routed, a node would first identify its flow. If the flow is found on the main table, then the respective next hop is retrieved and the packet is forwarded accordingly. Otherwise, the fallback table is queried and a fallback route is used.

The idea of this scheme with two different routing tables is to minimize the effects of the delay in propagating the information of the new flow to all network nodes. While the initial routes might not be optimal, they at least provide a viable path while the network state is updated throughout the network.

One last important detail regarding the routing tables is the format of the main table. A routing table usually is indexed only by the destination node of the packet. In the cases of IAR and ICAR, though, it is important that the main routing table is indexed by both source and destination nodes of a packet (or of a flow). That is due to the fact that IAR and ICAR allow two flows with the same destination node to use a same intermediate node, but with different path suffixes. For example, suppose that in a given network, there are two active flows $a \Rightarrow b$ and $c \Rightarrow b$. Due to the way candidate solutions are built, it is possible that IAR or ICAR would select two paths passing through a same node d, such that the subpaths from d to b are different for each flow. To allow the usage of the correct paths for each flow, node d must have in each entry of its main routing table both the source and destination nodes, in order to differentiate between the packets for flows $a \Rightarrow b$ and $c \Rightarrow b$. Notice that, while routing tables based on source addresses are less common, there are implementations readily available for them in major operating systems [46].

4.2 Coded Packet Transmission Methods

An important issue for network coding is how to actually perform the transmission of coded packets. This subject has already been briefly discussed in Chapter 3, but in this section we approach this issue more deeply. We first review two known methods

from the literature and discuss their characteristics. Then, we introduce a new method that overcomes some of the issues with the previous proposals. Finally, we discuss the consequences of using this proposed method in the models used by ICAR.

4.2.1 Methods Found on the Literature

The simplest idea to transmit a coded packet is to simply encapsulate it in a link layer broadcast frame. The broadcast frame would then be transmitted by the link layer protocol as usual. Upon the reception of such a packet, a node would check whether it belongs to the set of intended receivers. In this case, the node would try to decode the packet and, if successful, send it to the upper layers. Otherwise, the packet would simply be ignored.

This method, known as *Simple Broadcast* [56], has a very simple implementation. Nevertheless, its employment results in very different characteristics for the transmission of coded and native packets. While native (unicast) packets are usually transmitted using the ARQ technique, the employment of broadcast frames for coded packets results in a single transmission attempt. As a result, the transmission of a native packet in the link layer tends to be much more resilient to losses than its counterpart for coded packets. On the other hand, the transmission of a coded packet always occupies the wireless medium for the least amount of time, since no retransmissions are allowed and since there is no acknowledgment frame to be received.

A perhaps more interesting method found in the literature is the so called *Pseudo-Broadcast* [56]. In this method, each coded packet is mapped into a unicast frame at the link layer. Since a unicast frame needs a unicast destination MAC address, this address is chosen by Pseudo-Broadcast among the addresses of the intended receivers for the coded packet. The policy for choosing this destination MAC address is random, *i.e.*, the address is randomly selected among those of the intended receivers.

The Pseudo-Broadcast is widely considered in the literature as an improvement with respect to Simple Broadcast. The improvement stems from the possibly higher delivery probability achieved by Pseudo-Broadcast given the employment of error recovery techniques, such as ARQ, available directly from the link layer to unicast frames. While Pseudo-Broadcast tends to result in longer periods of medium occupation (due to the possibility of retransmission), it is usually assumed that the upper layers have considered this side effect when choosing the paths and that set of packets to be coded together. Under this assumption, the rationale behind Pseudo-Broadcast is that, if a higher protocol decided to code together a given set of native packets, then the link layer must make the

best effort possible to transmit that coded packet successfully for all intended receivers.

One issue with the Pseudo-Broadcast method is that it lacks a well defined criterion to choose the destination address. Suppose, for instance, that a given coded packet to be transmitted by node a is intended to two receivers b and c. Assume that the delivery probability of link $a \rightarrow b$ is 1, while the one for link $a \rightarrow c$ is 0.5. If the random selection performed by Pseudo-Broadcast results in node c being chosen as the destination MAC address for the unicast frame, then node b will certainly receive the coded packet, while node c will have a probability of $1 - 0.5^k$ of receiving it (where k denotes the maximum number of transmission attempts at the link layer). If k is 4 (as it is defined in the IEEE 802.11 standard), this probability reaches 93.75%, for example. On the other hand, if b is chosen as the destination node, only one transmission attempt will be made by node a (since b always receives successfully the attempts by a, and assuming the delivery probability for the ACK is also 1). In that case, the probability that c receives the coded packet drops to 50%.

That example, while simple, is useful to demonstrate how sub-optimal performance may result from the usage of the random policy employed by Pseudo-Broadcast (from the point of view of the delivery probability of coded packets). In the next section, we will present a proposal of a simple variation of the Pseudo-Broadcast that will maximize the performance in that regard.

4.2.2 Deterministic Pseudo-Broadcast

We argue that the adoption of a simple deterministic criterion can improve the performance of Pseudo-Broadcast. Similarly to the original proposal of the Pseudo-Broadcast method, we assume that the goal of transmitting a coded packet is that all receivers can successfully receive it. In other words, denoting by Suc_x the event of a coded packet being successfully received by node x, we would like to maximize the joint reception probability $P(Suc_a, Suc_b, \dots)$, where the intended receivers are nodes $a, b, \dots$.

When using the Pseudo-Broadcast technique, one of the intended receivers is used as the destination for the unicast frame and, thus, there is a confirmation when this node correctly receives the coded packet (assuming this occurs within the predefined limit of retransmissions). Denoting the chosen node by α, we wish to maximize the following expression:

$$P(Suc_a, Suc_b, \dots | Suc_\alpha) = \frac{P(Suc_a, Suc_b, \dots)}{P(Suc_\alpha)} \tag{4.1}$$

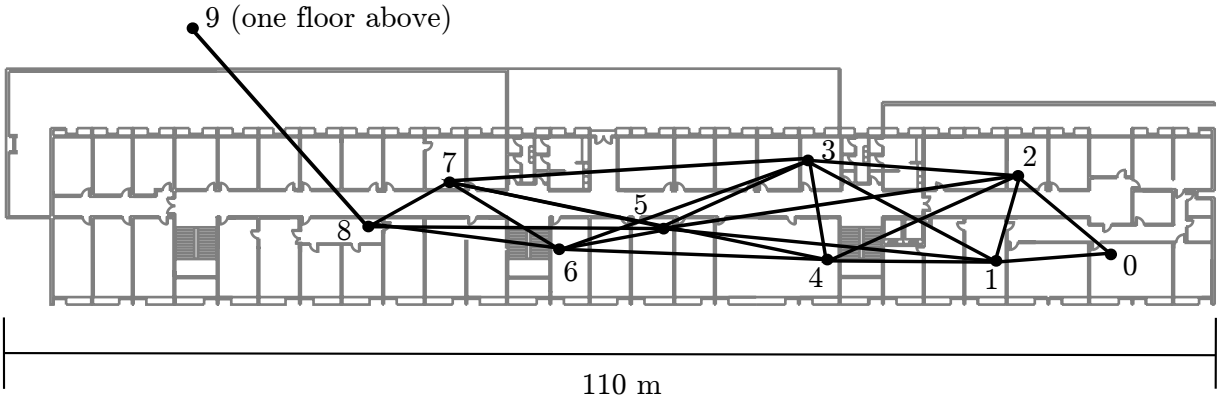


Figure 4.1: Topology of the wireless mesh network used in the experiments.

The numerator of the expression depends only on the intended receivers for the coded packet. Hence it is independent of the choice for the destination of the unicast frame. Therefore, to maximize the expression, it is necessary to minimize the value of the denominator. As a consequence, the choice that maximizes the joint reception probability for all receivers is the node with the lowest individual delivery probability with respect to the transmitter. Notice that by using this criterion, we are not artificially introducing a bad link that may be prone to channel outages and other side effects. By definition, node α is already among the receivers. Therefore, the correspondent link will be used regardless of the choice for the destination of the unicast frame. Since the individual delivery probabilities are usually already computed by most traditional routing protocols (and assumed to be available for the routing algorithm presented in this thesis), this strategy, hereinafter called *Deterministic Pseudo-Broadcast*[1], can be easily implemented.

The fact that the choice for the "worst" receiver as the destination address of the unicast frame maximizes the joint reception probability for all intended receivers is somewhat intuitive. By doing so, we are choosing the destination address that maximizes the expected number of retries in the link layer. With more retransmissions, all intended receivers have more opportunities to receive the coded packet, leading to an improved overall joint reception probability.

4.2.3 Experimental Evaluation

To evaluate the proposed mechanism, experiments were conducted in an indoor network, composed of 10 nodes deployed in two floors of a building. This network testbed has been used in previous work [17, 75]. Figure 4.1 illustrates the topology. The lines between nodes

[1]In this thesis, from this point on, we will always refer to the original Pseudo-Broadcast method as the *Random Pseudo-Broadcast* in order to avoid confusion

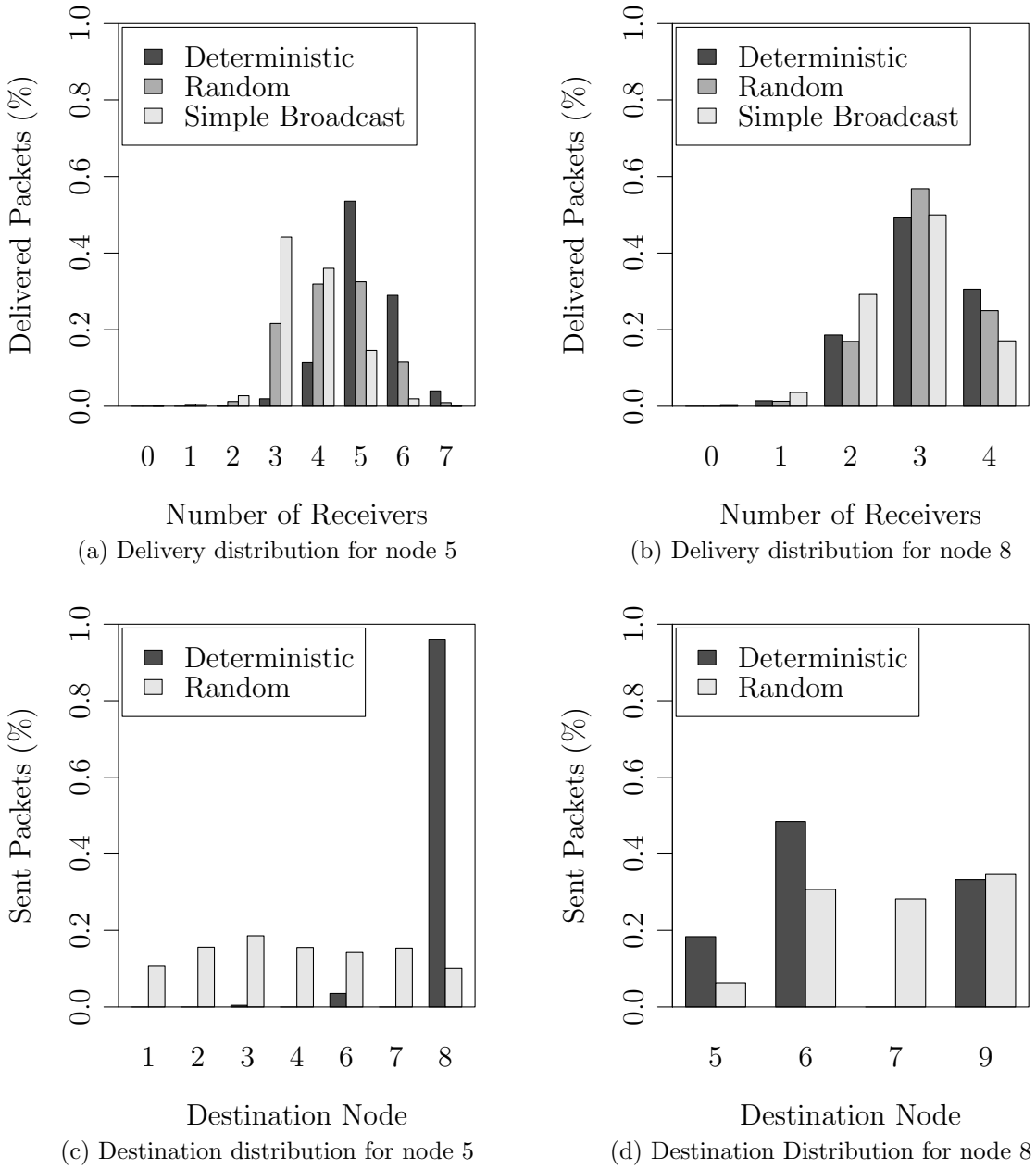
represent links frequently available in the network. Each network node is an off-the-shelf Linksys WRT54g router running the Linux-based OpenWrt operating system.

Three different mechanisms were implemented as plugins of the OLSR routing protocol [68]: the Simple Broadcast (encapsulate the coded packet in a link layer broadcast frame), the Random Pseudo-Broadcast and the Deterministic Pseudo-Broadcast.

When loaded, each plugin creates a virtual network interface, using Linux TUN/TAP support. When a packet is sent by a local application through an interface of this type, the local kernel delivers the packet to the application that owns the interface. Hence, the plugin is able to intercept packets transmitted to the subnetwork of the virtual interface and choose the best destination for the link layer frame according to the currently selected mechanism.

In this set of experiments, an application in the source node sends a sequence of packets to the broadcast address of the subnetwork of the virtual interface. Upon the reception of such a packet, the plugin generates a link layer frame with the chosen destination address. This frame is then passed to the wireless adapter which performs the actual transmission. The neighbors, on the other hand, run another application which waits for the packets and generates a log file containing the received sequence numbers. In total 3000 packets of 1500 bytes are sent with 100 ms intervals from each other. To increase the fairness of the experiment, all three compared mechanisms were executed simultaneously (our plugin is able to create three different TUN/TAP interfaces, each attached to a different transmission mechanism), guaranteeing that all mechanisms had the same network conditions.

The graphs in Figure 4.2 summarize the results obtained for two topology nodes (5 and 8) acting as sources of the data flow. Figure 4.2a shows that the Deterministic Pseudo-Broadcast achieved a delivery rate considerably superior to those of the other two mechanisms, considering 7, 6 and 5 receivers. By adding these three cases, we conclude that the Deterministic Pseudo-Broadcast could deliver the packets to at least 5 neighbors 86.5% of the time. The second best performance, in this case, was obtained with the Random Pseudo-Broadcast, which achieved only 45% of delivery, in the same conditions. Finally, the Simple Broadcast achieved a delivery rate of only 19%. By analyzing the frequency each neighbor was chosen as the destination for the link layer frame (Figure 4.2c), we notice the Deterministic Pseudo-Broadcast opted for node 8 in 96% of the time. The Random Pseudo-Broadcast, as expected, distributed its choices uniformly, ignoring the necessary retransmissions.



(a) Delivery distribution for node 5

(b) Delivery distribution for node 8

(c) Destination distribution for node 5

(d) Destination Distribution for node 8

Figure 4.2: Comparison results for the three mechanisms for transmitting coded packets. The two graphs above show the distribution of the number of neighbors that correctly received the transmitted packets. The two graphs bellow show the distribution of the destination addresses used by each mechanism for the link layer frames.

Most of the obtained results for other source nodes were quite similar in trend to the ones obtained for node 5. The only exception was node 8. As illustrated in Figure 4.2d, for this source node the Deterministic Pseudo-Broadcast mechanism presented a much even distribution in terms of destinations, approaching the choices made by the Random Pseudo-Broadcast (with the exception of node 7, which was never selected). The reason for this behavior is the high variability of quality of the output links for node 8. This variability causes the worst output link to change several times during the experiment. As a consequence, both mechanisms (Random and Deterministic Pseudo-Broadcast) had similar results in terms of delivery rate. Nevertheless the delivery rate considering 4 neighbors was higher using the Deterministic Pseudo-Broadcast (30.5% against 25% by the Random Pseudo-Broadcast).

4.2.4 Consequences for the Route Evaluation Algorithm

As previously explained, one side effect of the Deterministic Pseudo-Broadcast (and, to a lesser degree, of the Random Pseudo-Broadcast as well) is that the link layer transmissions of coded packets tend to use the medium for longer periods than with Simple Broadcast. Despite this trade-off, we argue that maximizing the joint reception probability of a coded packet is desirable, especially since the transmission of the native packet associated with the link selected by the Deterministic Pseudo-Broadcast would required the same amount of medium usage. In other words, if a coding oblivious routing protocol chooses that link to be used for a given flow, it is already considering that amount of medium usage in its decisions.

Nevertheless, it is particularly important for ICAR to know exactly how the transmission of coded packets is performed in order to evaluate the medium usage required by a give transmission, as well as the resultant delivery probability for each native packet coded together. In Chapter 3, we assumed the usage of Simple Broadcast, which resulted in the model presented then. While we present the Deterministic Random Broadcast as an alternative to Simple Broadcast and argue that it can improve the performance of network coding (assuming proper modeling is employed by the route selection algorithm), we assume such modeling is outside the scope of this thesis. We do, however, provide some insight on how that modeling can be done.

There are two components of the model employed by ICAR that would be affected by the usage of the Deterministic Pseudo-Broadcast. The first one is the medium usage time, which can be simply calculated by applying the same formula used in a native packet

transmission (see Equation 3.9) considering the link from the transmitter to the receiver α — the one with the worst delivery probability with respect to the transmitter. Notice that a similar approach can be used if the Random Pseudo-Broadcast is applied, with the difference that the final medium usage time has to be the average considering the cases when each intended receiver is chosen as the destination address.

The second affected component is more complicated, though. This component is the one that estimates the probability that a coded packet is received by a given intended receiver. With Simple Broadcast, this probability was simply the delivery probability of the receiver with respect to the transmitter. However, for Pseudo-Broadcast (both the random and the deterministic versions), multiple retries may be attempted by the transmitter. A simplified approach to this issue would be to compute the expected number of attempts (which is already done for estimating the medium usage) and simply calculate the reception probability for each intended receiver based on this number and on the respective delivery probability. The problem with this approach is that, as shown in Appendix B, the reception events for different intended receivers of a coded packet are not necessarily independent. In other words, the probability that an intended receiver a receives a coded packet that was correctly received by node b whose address was used for the unicast frame can be different from the probability of a receiving the packet in the same number of attempts regardless of what happened to node b. While the information shown in Appendix B can be used as a foundation, we do not attempt to provide a complete model in this thesis.

4.3 Collision Probability for Probe Packets

A third practical aspect that is important to consider when implementing IAR and ICAR is the collision probability of the probe packets used to estimate the delivery probability of each link. Both IAR and ICAR depend on the availability of estimates for the delivery probability information for all network links, from which other information is derived. The assumption that this information is available for the routing protocol is reasonable in practice, since most current routing metrics rely on this probability in one form or another, as shown in Chapter 2. However, those estimates may be susceptible to considerable measurement errors due to the probability of collision between probes and data packets [23]. In other words, whenever data flows start to traverse the network, the delivery probability estimates obtained by the routing protocols start to decrease, due to probe packets colliding with the increased amount of data packets.

One could argue that the effect of the measured delivery probability for a wireless link dropping under traffic is desirable. In other words, if this drop is caused by an increase in the number of collisions in the wireless medium, it reflects the true state of the link, since data packets transmitted through that link will be susceptible to the same conditions. However, the goal of many mechanisms that rely on those estimates is to measure only the wireless link quality itself, and not other phenomena that could influence its delivery probability. On [23], for example, the work that originally proposes the ETX metric, the author clearly states that the sensitivity of its probing mechanism to traffic load is a design flaw.

While the measurement errors in the delivery probabilities may affect the performance of traditional routing metrics, in this thesis, we argue that its effect on IAR and ICAR is potentially stronger. The reason for that is two folded. On one hand, IAR and ICAR derive two different values from these estimates (link transmission delay and packet receiving probability). On the other hand, they already take into account the impossibility of two interfering links transmitting at the same time by not allowing them to be scheduled simultaneously during the simulation that evaluates a given path set.

It is, therefore, important that not only the estimates for the links' delivery probabilities are as precise as possible, but also that they are as orthogonal as possible to the collision probability. In order to try to improve these two characteristics, in the next section we present a proposal for mitigating the collision component of the estimates provided by the usage of broadcast probes, hereinafter referred to as *collision probability adjustment*.

4.3.1 Proposed Method

The collision probability adjustment method is based on the following premise. If the routing protocol was able to somehow find out what the current collision probability is for all network links, it would be possible to compute an estimate for the delivery probability orthogonal to collision (denoted by d', as opposed to the traditional estimate d) using the following expression:

$$d'_{a\to b} = \min\left(\frac{s}{w \times (1 - \overline{p_{col_{a\to b}}})}, 1\right), \tag{4.2}$$

where s denotes the number of successfully received probes for a window of w probes, and $\overline{p_{col_{a\to b}}}$ is the average collision probability at node b for probes sent by node a during the

current window. Basically, what this expression does is compute the expected number of probes that are not lost due to collision and use it as the denominator for the ratio of received packets. Since this value is only an expected number, it is possible that less probes are actually lost due to collision and the total number of received probes during the window (s) is greater than the denominator. In that case, the expression becomes limited by 1 so that it represents a valid probability value.

The key to isolate the collision probability from the delivery probability, then, is to know — or to estimate — the probability $p_{col_{a \to b}}$. Generally speaking, a collision may arise from two different causes:

- two or more nodes start to transmit at the same time (or very close to that); or
- a node cannot detect that a transmission is already taking place and starts its own transmission.

The IEEE 802.11 standard, for example, tries to mitigate the first cause by employing a random backoff value chosen from an exponentially growing window [49]. The second cause, known in the literature as the hidden terminal problem [96], can be solved in infrastructured networks by the employment of RTS/CTS frames (Request To Send and Clear To Send) [54]. Notice, however, that RTS frames are never sent before broadcast frames due to the existence of multiple receivers, rendering this method ineffective against collisions involving this kind of frame. In other words, in a network that employs RTS/CTS frames, the collision probability for probe packets may be higher than that for unicast data packets. Moreover, in this work we do not assume the usage of RTS/CTS by the network nodes. For this reason, the collision probability adjustment method presented on this thesis focus on the second case.

The collisions caused by the hidden terminal problem are exacerbated by the amount of traffic on the network. For instance, assume that a given network has very low traffic usage at a given moment in time. If a node a wishes to send a packet to node b, the probability that another node c — hidden, from the point of view of a — is already transmitting is low. On the other hand, if the network, or specifically node c, is under heavy load, then the probability that node c is already transmitting while a tries to send a packet to b increases, also increasing the overall collision probability. Therefore, the collision probability caused by hidden terminals is a function of both the number of hidden terminals and their respective medium usage. Specifically, the collision probability for a link $a \to b$ due to hidden terminals is equivalent to the probability that node a tries

to transmit while any of the terminals hidden from a that can affect that transmission is already transmitting.

With that in mind, we propose the following approximation for the collision probability of a probe sent from node a to node b:

$$p_{col_{a\to b}} = \frac{1 - idle_b}{\sum_{i \in N_b} AirTime_i} \times \sum_{i \in N_b, i \neq a} AirTime_i \cdot (1 - d_{i\to a}). \tag{4.3}$$

In the expression, $idle_b$ represents the percentage of time node b has detected the wireless medium as being idle, $AirTime_i$ is the percentage of time node i has used the medium, and N_b is the set of neighbors of node b. The idea of this expression is to approximate the collision probability with the percentage of time in which potential hidden terminals are transmitting. We consider a node i to definitely be a hidden terminal for link $a \to b$ if it is neighbor of b, but not a neighbor of a ($d_{i\to a} = 0$). However, due to the existence of time-varying fading [86], we also take into account the possibility that, in a given moment, a neighbor of a node a can also act as a hidden terminal. To this end, we sum the values of $AirTime$ for all neighbors of b weighted by the complement of their respective delivery probability with respect to a. The result of that summation is them normalized so that the sum of $AirTime$ values for all neighbors of b correspond to the actual percentage of time node b has detected the wireless medium as being occupied (notice that multiple neighbors of b may transmit at the same time, resulting in that sum being greater than $(1 - idle_b)$).

Values $idle$ and $AirTime$ may not be as commonly available to routing protocols as the neighborhood information and the delivery probability for each network link. However, it is still feasible to obtain such values in a practical implementation. The $idle$ value for a node is provided by some popular drivers for IEEE 802.11 wireless cards such as [7] and can be retrieved by software through standard API calls in Linux. The $AirTime$ value can be obtained by either monitoring the total number of bytes transmitted by the wireless interface (including retransmissions), or estimated at the routing layer by monitoring each packet that leaves the queue considering the ETX of the respective link as the number of retransmissions (or 1 for broadcast frames). Each node should monitor the behavior of these parameters in samples during a given time interval (in this thesis we employ measurement intervals of 10 seconds) and broadcast its $AirTime$ value within the hello packets. Whenever a node b has to compute the adjusted delivery probability from a neighbor a to itself, it computes the collision probability of that link using Equation 4.3

based on the values of $AirTime$ it has received from its neighbors, on the unadjusted delivery probabilities calculated from the reception of the probes, and its own *idle* value.

One important detail on this method is that the estimate of the collision probability changes relatively fast with changes in the load of the network, since it uses samples for the values of *idle* and $AirTime$. On the other hand, the delivery probability itself tends to change slowly, since it results from the average of multiple samples (probe transmissions). Suppose, for instance, that each node samples $AirTime$ and *idle* every 10 seconds, while probes are transmitted every 5 seconds and the window used for the average comprises the last 100 probes. In that case, while $AirTime$ and *idle* correspond to the state of the network in the past 10 seconds (or a few more seconds because of the delay for the information to propagate through the neighbors), the unadjusted delivery probability conveys information of the last 500 seconds. To avoid this mismatch, instead of using directly the value provided by Equation 4.3 as the collision probability in Equation 4.2, the collision probability adjustment method uses a moving average of these values. The parameters of the average (*e.g.*, the number of samples) are chosen so that the period of time from which the samples of collision probability are taken matches the period of time from which the probes are considered for computing the delivery probability in Equation 4.2. Specifically, in this thesis, we opted for adopting an Exponentially Weighted Moving Average (EWMA) instead of the traditional moving average so that less state has to be stored — the EWMA requires only that the current average value is stored, in contrast to all the currently used samples. Therefore, the value used as the collision probability in Equation 4.2 is given by:

$$\overline{p_{col_{a\to b}}} = \theta \cdot \overline{p_{col_{a\to b}}} + (1-\theta) \cdot p_{col_{a\to b}}, \tag{4.4}$$

where $p_{col_{a\to b}}$ is the new sample value of the collision probability (as given by Equation 4.3) and θ is the exponent for the EWMA — hereinafter referred to as *collision probability exponent*. This exponent, therefore, has to be chosen such that the amortization rate of the two means — for the average collision probability and for the delivery probability — are similar.

4.3.2 Discussion on the Precision of the Method

The collision probability adjustment method proposed in this thesis is an approximation for the computing of the delivery probability orthogonal to the collision probability. As with any approximation, precision issues must be considered.

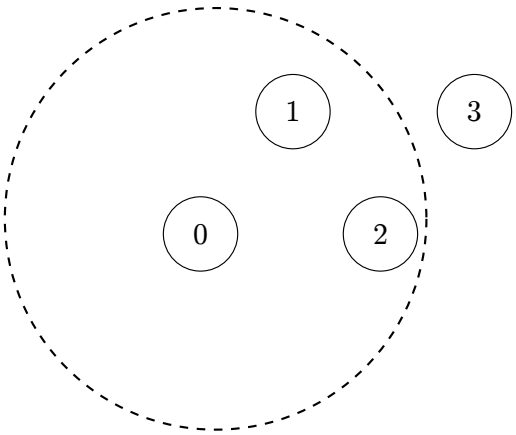


Figure 4.3: Example of a node distribution that might lead the collision probability adjustment method to underestimate the collision probability. The dotted circle around node 0 illustrates the hypothetical transmission radius of the node. Node 3 cannot overhear the transmissions by node 0, while nodes 1 and 2 can. In this example, we would like to compute the collision probability for link $3 \rightarrow 2$, assuming nodes 0 and 1 have their respective buffers backlogged.

The first source of imprecision in this method is considering only collisions that stem from hidden terminals. Although the usage of random backoff by wireless MAC layer protocols (IEEE 802.11, specifically) mitigates the occurrence of collisions between transmitters that can overhear each other, this technique is still susceptible to nodes casually synchronizing their transmissions, resulting in collisions.

Another simplification is the simple summation of the values of $AirTime$ for the potential hidden terminals. Since multiple nodes (among those potential hidden terminals) may transmit at the same time, the summation may be an overestimate of the actual medium occupation fraction. We try to account for that by normalizing this sum with respect to the complement of the actual idle time, as perceived by the receiving node, but we cannot guarantee that this procedure results in the correct value in the general case. Moreover, the usage of the unadjusted delivery probability as a weight in order to capture the possible effects of fading on the capability of the sender's carrier sense to detect an ongoing transmission may not correlate perfectly with the physical phenomenon.

Finally, there is the issue of the dependency between the events of transmission in a multihop wireless network network. Consider, for instance, the example illustrated by Figure 4.3. Suppose a multihop wireless network has that distribution of nodes and we would like to estimate the collision probability for probes sent by link $3 \rightarrow 2$, assuming nodes 0 and 1 have their respective buffers backlogged. Assume also that the carrier sense performed by node 3 is always unable to detect transmissions by node 0, but can perfectly detect transmissions by node 1. Similarly, node 1 is capable of detecting when

node 0 transmits. Under these circumstances, whenever node 0 starts to transmit, node 1 does not transmit, but the CSMA/CA protocol running on node 3 allows it to transmit simultaneously. If the transmitted packet is addressed to node 2, a collision will happen. In this case, the fact that node 0 is not a hidden terminal for node 1 introduces a bias in the collision probability by creating a dependency between the times at which node 0 is transmitting and the times at which node 3 will find the medium to be free. With more nodes distributed in random ways, more complex patterns of dependency may arise, which are not covered by the proposed collision probability adjustment method.

Despite these simplifications and sources of imprecisions, we argue that the proposed method can mitigate the effect of collisions in the delivery probability estimates, resulting in more stability for the route selection algorithms. In Chapter 5, we provide experimental data to support this claim.

4.4 Delayed Execution of the Route Selection Algorithm

While the route selection algorithms presented in Chapter 3 make use of heuristics and can be used to find routes in real time, it is still more complex, in terms of computational cost, than traditional routing algorithms. Usually, routing protocols for multihop wireless networks recompute routes any time they receive new information (through topology packets) that change their current view of the network — which might result in different routes. However, due to the relatively short intervals between the generation of topology packets — usually in the order of a few seconds — a network with a high number of nodes may result in a node receiving various different topology packets in short time intervals (less than a second), which leads the routing protocol to call the route selection algorithm many times in sequence. This problem is aggravated because some protocols generate new sporadic topology packets when they detect changes in their neighborhood. In other words, when a change happens in a given neighborhood, all affected nodes will generate new topology packets in a short time interval.

With this characteristic in mind, in this thesis we adopt a delayed execution strategy for reducing the CPU usage of network nodes. The idea is that, whenever a node receives a topology packet containing changes in the link state, it delays the execution of the route selection algorithm for a short period of time, instead of executing it immediately. During that time span, if other packets are received, the protocol's network vision is updated, but

the route selection algorithm is not called. This way, multiple changes to the topology are aggregated and only one execution of the route selection algorithm is required.

The trade-off of this strategy is clear: while it reduces the number of executions of the route selection algorithm, it also introduces a delay for nodes to find new routes when the network state changes. Nevertheless, in preliminary experiments we verified that using short delays (of one second, for example), the network was able to sustain good performance (as shown in the results presented in Chapter 5), while considerably reducing CPU usage.

Chapter 5

Performance Evaluation

This chapter describes the experiments conducted to evaluate the performance of IAR and ICAR, as well as their results. Differently from the results presented in Chapter 4, in this chapter we focus in network performance parameters (such as throughput, end-to-end delay and packet loss), considering how all parts of the proposal work under real conditions. To this end, the experiments were divided in two groups: simulations with the well-known ns-2 simulator [93] and experiments in a real testbed. While simulations allow one to easily reproduce network conditions and explore different scenarios, a real testbed provides all the characteristic complexities of the wireless medium, as well as the real-time processing requirements for the route computations.

In both cases, IAR and ICAR were implemented on top of the SLSP routing protocol [71, 75] (Simple Link State Protocol). For the real testbed, the original code from SLSP was modified to support the proposals. Afterwards, the resulting code was ported and adapted to work as a routing agent for ns-2. Besides IAR and ICAR, the ETX, ML and Hop Count route selection methods[1] are also implemented in SLSP and, thus, available for this evaluation. For all experiments (simulated or real), the protocol was configured to use the parameters listed in Table 5.1.

As of the writing of this thesis, to the best of our knowledge there are no implementations of COPE (or similar techniques) for ns-2. For this reason, we also implemented a simplified version of COPE for the simulator by creating modified versions of the Drop-Tail queuing policy and of the IEEE 802.11 MAC layer. The Drop-Tail queuing was modified to include COPE's routine for finding possible coding partners for a packet, to implement a *Packet Pool* (a buffer of native packets transmitted or received to be possibly used for decoding in the future), as well as to implement coding and decoding routines.

[1] Although ETX, ML, and Hop Count are routing metrics, in this chapter we employ the more generic term *route selection method* so that we can refer to those metrics in the same context of IAR and ICAR.

Table 5.1: Configuration parameters used during the experiments for the routing protocol. The names listed in the table represent the names of the parameters as used by the implementations of the SLSP protocol.

Name	Description	Value
`hello_interval`	Interval between Hello packets	5 s
`topology_interval`	Interval between Topology packets	10 s
`hello_life_time`	Validity of each Hello packet	510 s
`lq_window_size`	Number of Hello packets used for estimating link quality	100
`topology_life_time`	Validity of each Topology packet	50 s
`idle_interval`	Interval between samples of the IEEE 802.11 Idle Counter	10 s
`airtime_interval`	Interval between samples of queue estimate of Used Air Time	10 s
`collision_prob_exponent`	Exponent of the EWMA for the collision probability	0.975

The IEEE 802.11 MAC layer was modified to include a coding header in each packet, to store received packets not addressed to the local node in the Packet Pool and to call the decoding routines when necessary. Notice that we chose to use the `dei80211mr` [9] implementation of the IEEE 802.11 standard instead of the one that comes with ns-2 by default due to the various improvements brought by this version (such as a SINR-based packet error model and the capture model). All modified or implemented components of the simulation environment used in this evaluation are available at [72].

For the real testbed, the same simplified version of COPE was implemented as a module of the SLSP routing protocol. Since SLSP is implemented in the user space, it was necessary to create an architecture to allow coding routines to be implemented without access to the kernel space. To this end, SLSP creates a TUN interface [59], which should be used by the host applications to send and receive all network traffic. All outgoing packets are then received by SLSP through the TUN interface and forwarded to the actual physical wireless interface. By using this architecture, SLSP is capable of performing queuing for the traffic, which allows it to implement COPE's coding capabilities, as well as to monitor the transmissions in order to collect statistics for the probability adjustment technique. Since a non-standard header must be added for all packets (a coding header) and since COPE requires even frames that are not addressed to a node to be stored by it, SLSP also puts the wireless interface in monitor mode and opens a raw socket in order to receive and transmit packets. Figure 5.1 summarizes this architecture.

It is important to point out that this solution includes a greater overhead to the

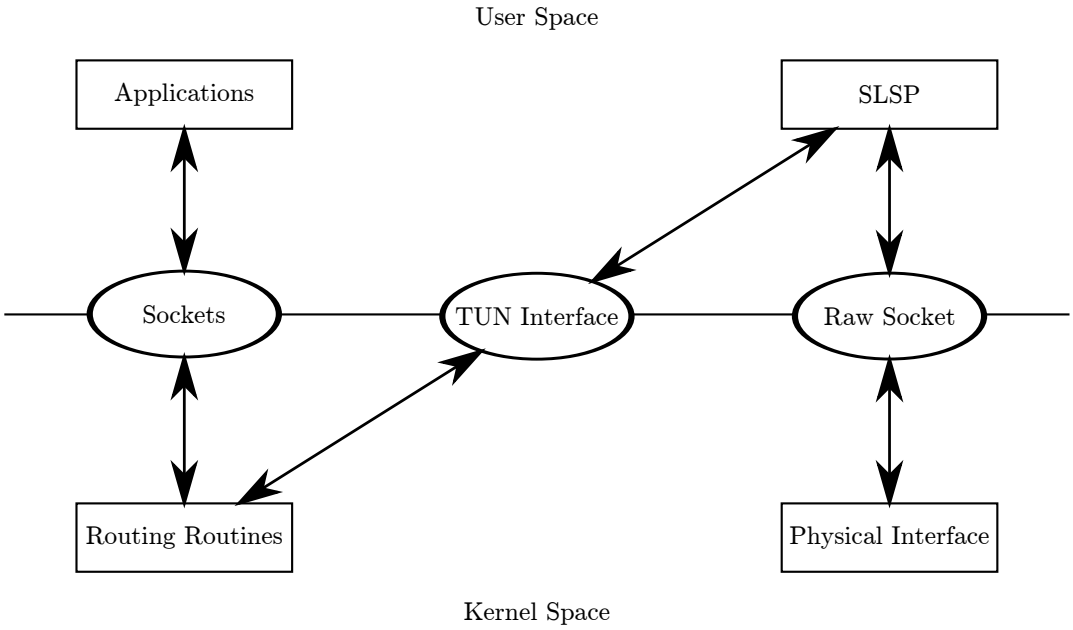


Figure 5.1: Architecture used for implementing COPE in the real testbed. Arrows indicate the directions in which data packets flow from one component to the other. Ellipses represent means of communication between the kernel and user space processes.

routing process in comparison to an ideal implementation completely within the kernel space. Most of the added overhead is due to the increase in the number of context switches (*i.e.*, from the user space to the kernel space and the other way around) needed to transmit or receive a packet. A packet transmitted by an application, for example, goes to the kernel space, comes back to the user space (to SLSP), and then is re-injected into the kernel space through the raw socket. Even for forwarding a packet, it first needs to be received by the SLSP process for later to be returned to the kernel space. Nevertheless, despite the increase in overhead, the proposed implementation is a functional prototype and we argue it is sufficient for the evaluation purposes of this thesis. The source code for the version of the SLSP protocol used for this evaluation along with the implemented modules are available at [73].

5.1 Simulations

Since simulations allow one to easily vary scenarios, in this section we tried to explore a number of different topologies covering a range of characteristics. We employed both especially designed topologies with the goal of demonstrating particular characteristics of each proposal, as well as more generic and random topologies.

Table 5.2: Propagation parameters used for all simulation scenarios.

Parameter	Value
Propagation Model	Shadowing [86]
Path Loss Exponent	4.1
Shadowing Reference Distance	1.0 m
Shadowing Standard Deviation	3.0 dB
Node's Transmission Power	19.5 dBm
Transmission Frequency	2437 MHz

A total of 6 different network topologies were created for this evaluation. Unless otherwise specified, all of them employ the same propagation parameters, summarized in Table 5.2, chosen after preliminary measurements such that the propagation environment would resemble that of the indoor wireless mesh testbed of the ReMoTE Project [75]. In all simulations, RTS/CTS frames were disabled and data rate was fixed at 1 Mb/s.

5.1.1 Basic Scenarios

For an initial evaluation, we employed four basic scenarios in order to analyze specific aspects of IAR and ICAR, as well as deficiencies of traditional route selection methods. In the next sections, we describe each one in detail, along with their respective simulation results.

5.1.1.1 Pro-Interference Scenario

The Pro-Interference scenario consists of 12 nodes deployed as depicted by Figure 5.2. When nodes 0 and 1, located at the extremes of the topology, perform a bidirectional communication, the distribution of the nodes suggests two different basic options for each flow: the "upper line" (passing through nodes 2 to 6) and the "lower line" (passing through nodes 7 to 11). Due to the distances between both lines, upper nodes do not interfere with the lower ones.

The best possible path selection is to assign flow $1 \rightarrow 0$ to one line and flow $0 \rightarrow 1$ to the other. Specifically, given the distances between nodes in each line and the propagation parameters, the best possible path is to select all nodes of a line in sequence (*e.g.*, $0 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 1$), in order to avoid links with high loss rates. By assigning different lines for each flow, they will only interfere with each other at the extreme nodes 0 and 1. If, however, one chooses to assign both flows to the same line, then the interference

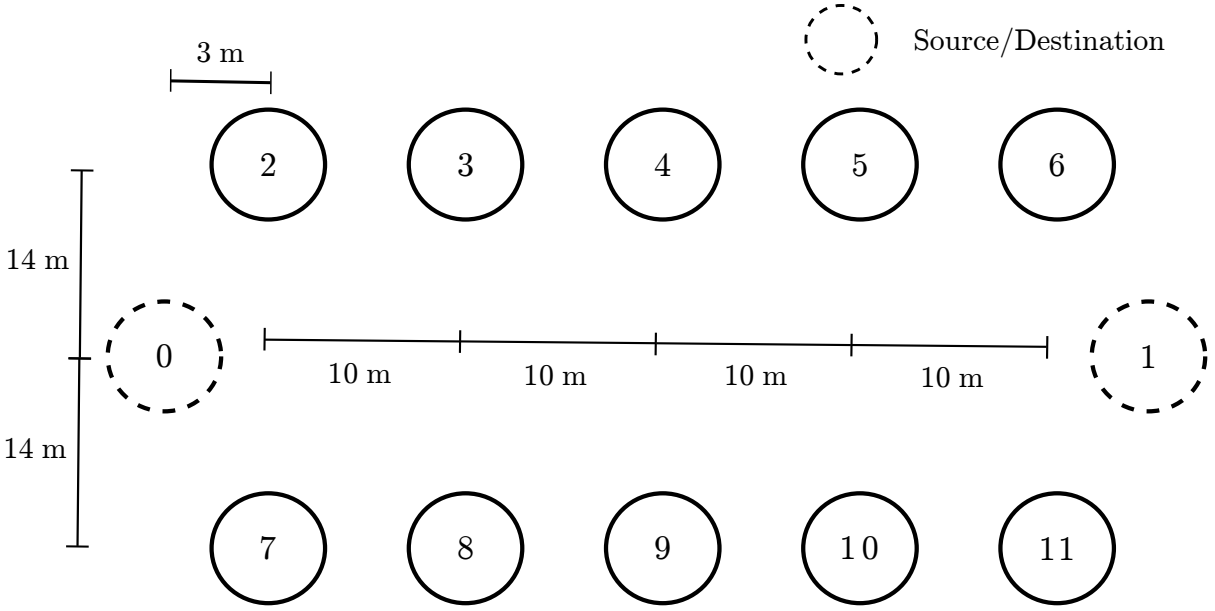


Figure 5.2: Pro-Interference Scenario: a scenario that presents a clear opportunity for avoiding interference by re-routing a flow through an alternative path.

will be much higher, reducing aggregated throughput.

Therefore, this scenario clearly favors an approach that is aware of inter-flow interference, such as IAR. To evaluate whether IAR can actually find the optimal path in this scenario, we performed a series of simulations with the two unidirectional CBR (Constant Bit Rate) flows $0 \rightarrow 1$ and $1 \rightarrow 0$ (*i.e.*, data flows both from 0 to 1 and 1 to 0). The flow consisted of 1450-byte packets sent at a rate of 200 Kb/s in each direction during 1000 seconds. This rate was chosen in order to saturate the capacity of the paths. Before the flows start, we let the routing protocol run during 500 seconds so that link quality information can converge.

Figure 5.3 shows the results obtained in terms of aggregated throughput for simulations without network coding. The bars represent the average of 5 runs with different seeds and, to avoid transients, we discard the first 100 seconds of the flows. The error bars represent the 95% confidence interval for the average. While IAR achieved more than 130 Kb/s of aggregated throughput, ML, the second best route selection method in this scenario, resulted in an average throughput of 69.77 Kb/s.

As expected, IAR was able to take advantage of the node distribution in order to choose mostly non-interfering paths. Figure 5.4 shows the fraction of time that IAR and the other evaluated route selection methods choose each of the most common sets of paths during the simulations. Specifically, the two sets of paths most frequently chosen by IAR

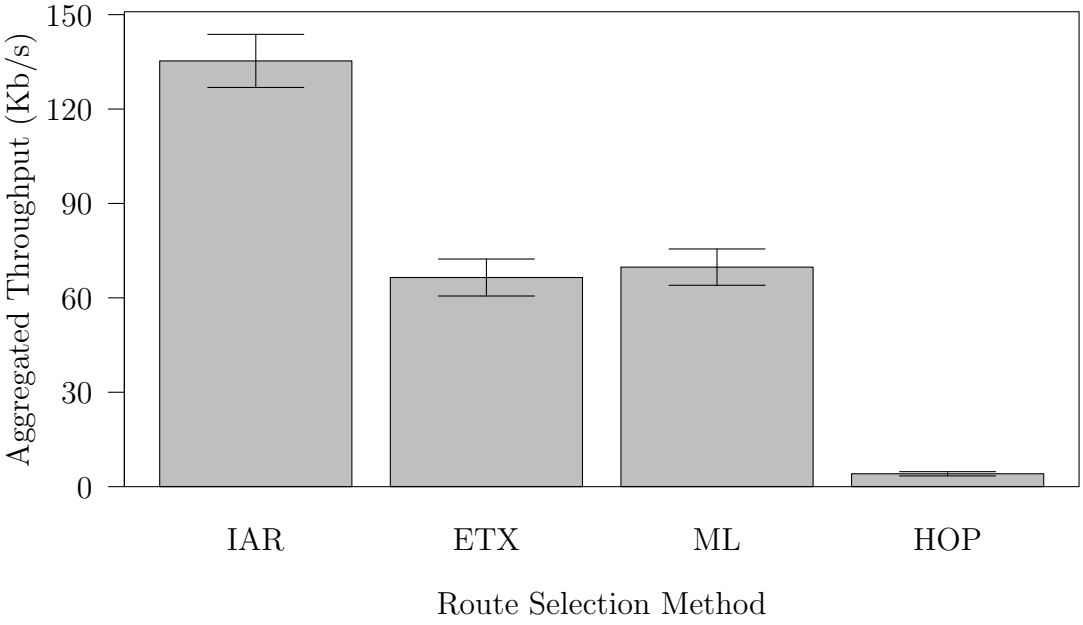


Figure 5.3: Comparison between IAR, ETX, ML, and Hop Count in terms of aggregated throughput in the Pro-Interference Scenario without network coding.

were $\{0 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 1,1 \rightarrow 11 \rightarrow 10 \rightarrow 9 \rightarrow 8 \rightarrow 7 \rightarrow 0\}$ (40.5% of the time) and $\{0 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 11 \rightarrow 1,1 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 0\}$ (25.7% of the time). On the other hand, ETX, ML and Hop Count tend to choose symmetric paths because the expression they use to attribute weights to links is identical irrespective of the direction of the link. Indeed, Figure 5.4 shows that this is the case. Both ETX and ML chose symmetric routes for more than 85% of the time, leading to roughly half the throughput obtained by IAR. With respect to Hop Count, not only the metric tends to choose symmetric paths, but it also selects very long links. In fact, Hop Count was unable to select one of the four most common sets of paths throughout the simulation, which, ultimately, led to the worst result in terms of aggregated throughput.

Figure 5.5 shows the percentage of lost packets for each route selection method, as well as a breakdown of the causes that accounted for those losses. Except for Hop Count, all route selection methods resulted in some losses due to routing loops. However, loops only caused losses for a very low percentage of the transmitted packets (the highest was 0.3% for ETX) and, thus, are not visible in the Figure. As expected, most of the losses are due to buffer overflow because the transmission rate of the flows saturated capacity of the paths, which resulted in queuing at the source nodes and, consequently, discards.

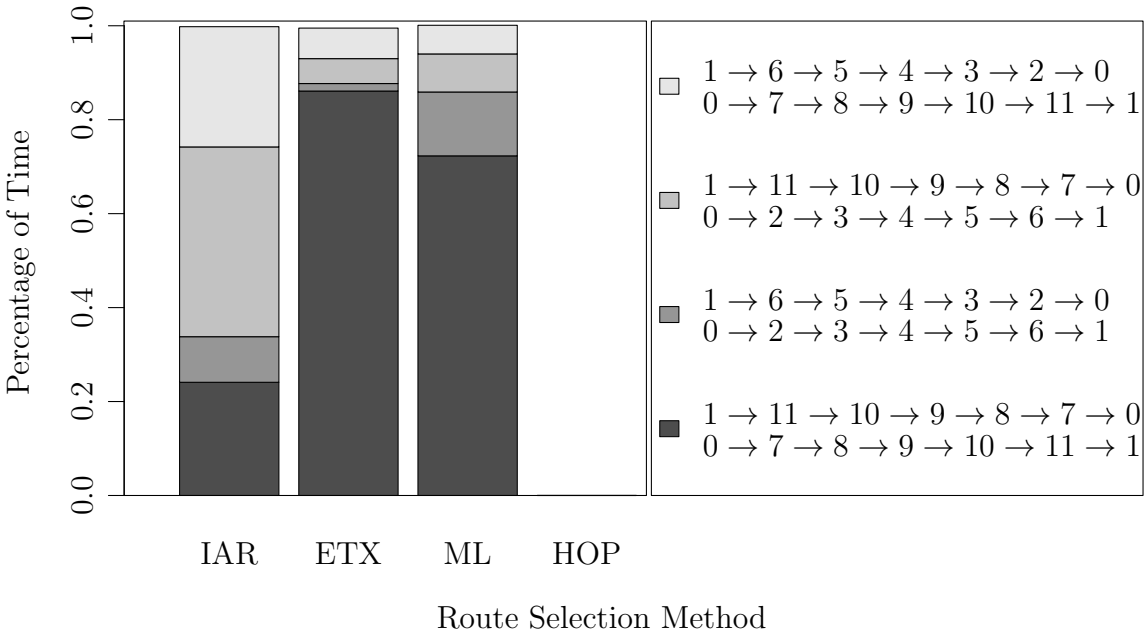


Figure 5.4: Comparison between the routes selected by IAR, ETX, ML, and Hop Count in the Pro-Interference Scenario without network coding. The graph shows the percentage of time in which each route selection method selected each of the four most common path sets.

It is interesting to notice, however, that both Hop Count and IAR had considerably less losses due to buffer overflow than ETX and ML. For Hop Count, that happens because it chooses shorter paths than the other route selection methods (in terms of hops) and worse links (so that most packets are lost in the first hop due to an excessive number of retries). These two factors combined result in more available medium access time for the source nodes to transmit, leading to less packet discards due to buffer overflow. IAR, on the other hand, tends to choose path sets with more available aggregate bandwidth, lessing the issue of queuing in the source nodes. Moreover, since IAR tends to separate the two flows, it also reduces the probability of buffer overflow in the intermediate nodes, since there is less sharing of resources (in this case, buffer space).

Another interesting point shown by this graph is the loss rate due to excessive number of retries (*i.e.*, packets that were actually lost due to phenomena of the wireless medium). IAR actually had more such losses than ETX and ML. Notice, however, that IAR also had more packets than ETX and ML that got the opportunity to be transmitted in the wireless medium, since it resulted in fewer losses due to buffer overflow. By computing the

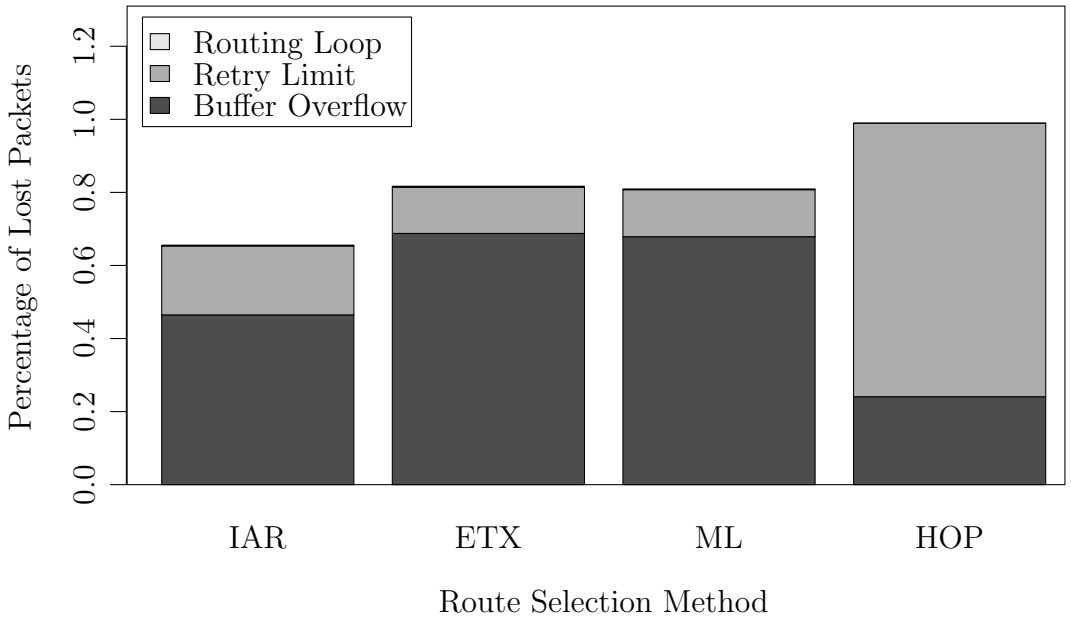


Figure 5.5: Breakdown of packet loss causes for each route selection method in the Pro-Interference Scenario without network coding.

percentage of losses during actual wireless transmissions with respect to the total number of packets that were not lost due to other causes, we conclude that IAR had a loss rate of 35.39% in this case, against 40.81% and 40.29% from ETX and ML, respectively. That is also an expected result, since, by choosing non-interfering paths for each flow more frequently, IAR tends to reduce losses due to collision. Since paths are symmetric, there is no difference in frame loss probability due to other wireless phenomena.

Figure 5.6 shows the average delay for each route selection method. IAR and Hop Count achieved the lowest averages, which is consistent with their lower levels of losses due to buffer overflow (although Hop Count does so at the expense of the packet loss rates). Both ETX and ML achieved similar averages, 73.5% and 70.3% higher than IAR, respectively. Again, as expected, this is a result of the higher levels of dispute for the wireless medium among network nodes.

Two interesting questions that arise from this scenario are how ICAR would perform if network coding was enabled and whether coding could improve the performance achieved by the other route selection methods to the levels achieved by IAR without it. To answer those questions, we repeated this set of simulations, but now we enabled network coding

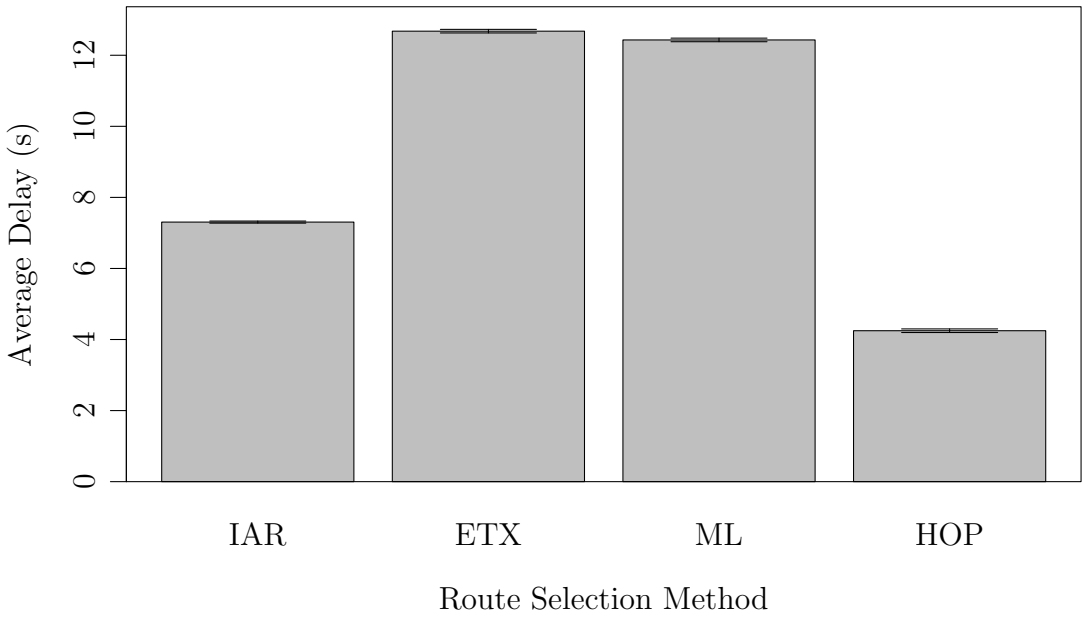


Figure 5.6: Comparison between IAR, ETX, ML, and Hop Count in terms of average delay in the Pro-Interference Scenario without network coding.

using Simple Broadcast as the method for transmitting coded packets. Notice that COPE, the employed method for network coding in this evaluation, cannot be used jointly with Hop Count since the latter does not provide the necessary link quality estimates for evaluating whether coding is possible. Therefore, for all results involving network coding, we do not consider Hop Count.

Figure 5.7 shows the average aggregated throughput obtained by ICAR and each of the other available route selection methods. For comparison, the result obtained with IAR without network coding is also presented. While the performances for ETX and ML greatly improved with respect to the case without network coding (85% and 84.2%, respectively), they still could not perform to the same level as IAR without it (without network coding, IAR achieved an aggregated throughput 10% and 5.3% higher than ETX and ML with network coding, respectively). On the other hand, ICAR with network coding achieved the best overall performance, improving IAR without network coding by 14.6%.

While network coding has lessen the issues associated with choosing interfering paths for both flows, thus increasing the performances of ETX and ML, it is still not the best

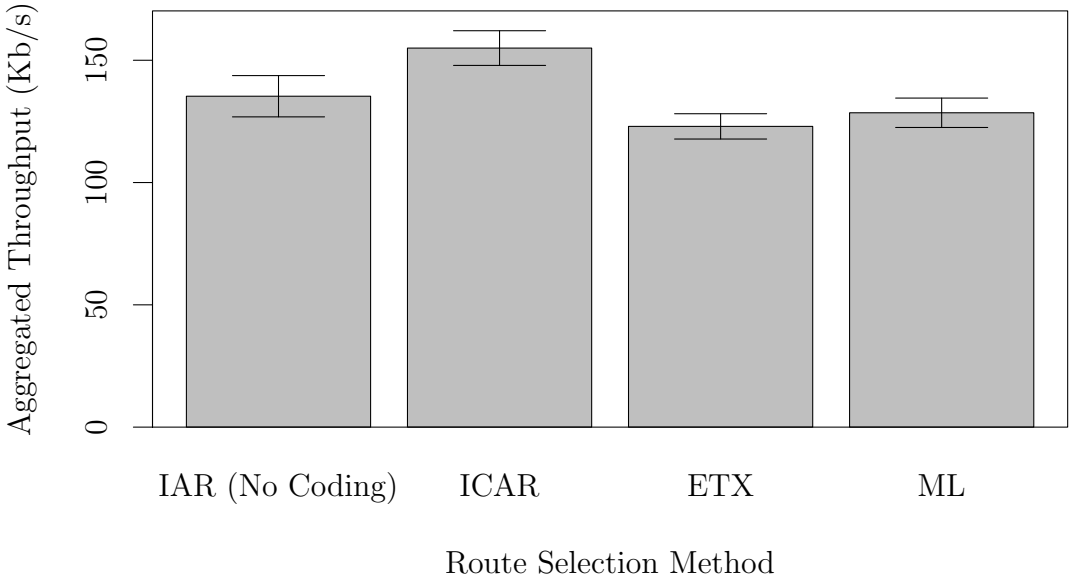


Figure 5.7: Comparison between IAR, ETX, ML, and Hop Count in terms of aggregated throughput in the Pro-Interference Scenario with network coding using Simple Broadcast. For comparison, the graph also includes the result obtained with IAR in this same scenario, but without network coding.

option for the scenario, as it increases the loss probability for packets transmitted in the wireless medium (since, with Simple Broadcast, it decreases the number of retransmission attempts to 1). In fact, the route choices by ICAR were very similar to those by IAR, as shown in Figure 5.8, indicating that even with the availability of network coding, the model predicted more often that the best option was still to separate the flows. The gain obtained by ICAR with respect to IAR is due to the fact that network coding lesses the drop in aggregated throughput whenever ICAR mistakenly evaluates that the flows should be assigned to interfering paths.

Figure 5.9 shows the breakdown of the frequencies for each reason for lost packets. All route selection methods presented considerably less losses due to buffer overflow in comparison to the results obtained without network coding. It is interesting to notice that this drop is more accentuated for ETX and ML than it is for ICAR (with respect to the results of IAR in simulations without network coding). This is due to the fact that ETX and ML choose more frequently interfering paths, and thus suffer a greater effect from network coding. Notice also that, by enabling network coding, ETX and ML achieve even

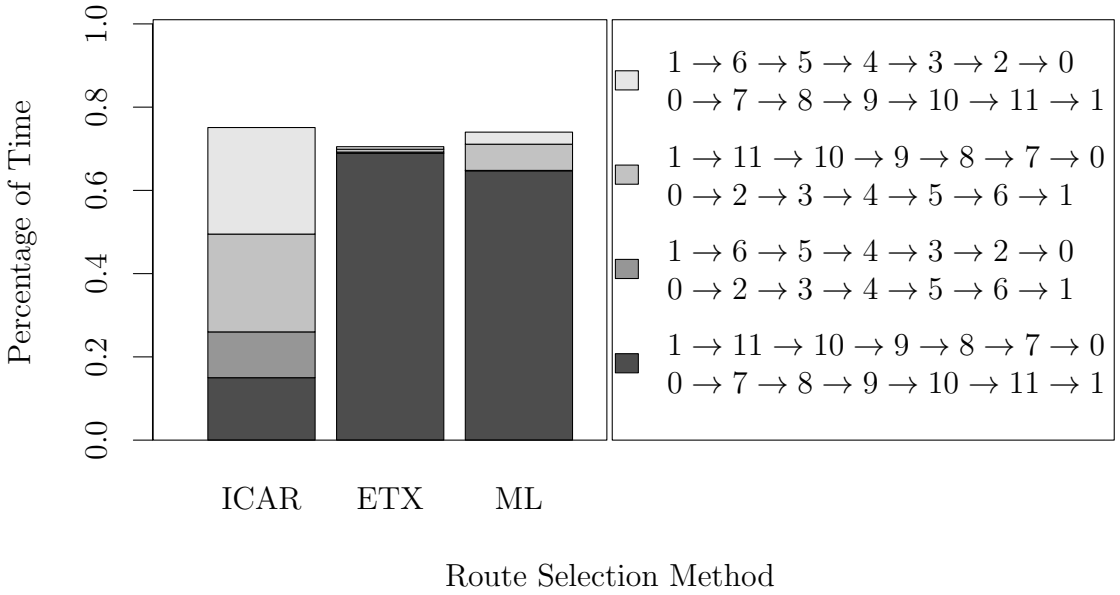


Figure 5.8: Comparison between the routes selected by ICAR, ETX, and ML in the Pro-Interference Scenario with network coding using Simple Broadcast. The graph shows the percentage of time in which each route selection method selected each of the four most common path sets.

lower levels losses due to buffer overflow than IAR did in the first set of simulations. That is due to the reduction in the medium usage with network coding, since Simple Broadcast transmits each coded packet at most one time. That, however, leads to a greater number of losses of coded packets in the wireless medium.

Figure 5.10 shows the average values of delay obtained by each route selection method. Both ETX and ML greatly reduced their average end-to-end delay with the employment of network coding. ICAR also presents a lower average end-to-end delay with respect to the previous results by IAR for the same reasons, albeit in a much smaller proportion.

5.1.1.2 Pro-Coding Scenario

The previous section discussed a scenario which presents a clear opportunity for avoiding interference by rerouting flows in order to spatially separate them. Similarly, this section presents a scenario with characteristics that force the ideal routing to reroute flows in order to bring them closer to each other, creating more coding opportunities (assuming the network has the capability of performing network coding). This scenario, depicted in

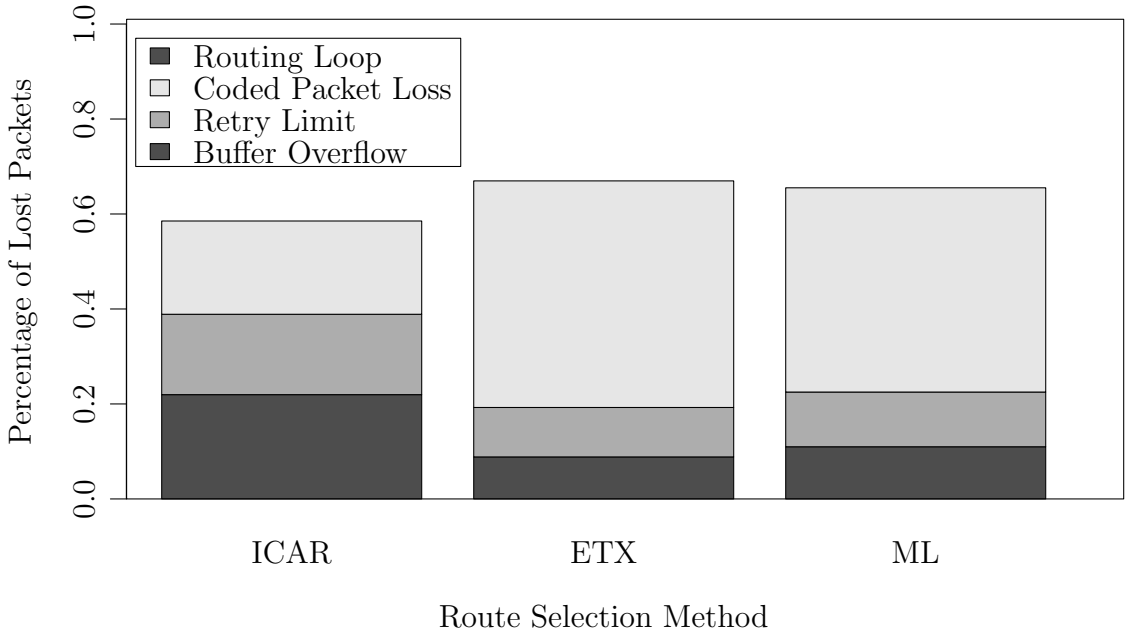


Figure 5.9: Breakdown of packet loss causes for each route selection method in the Pro-Interference Scenario with network coding using Simple Broadcast.

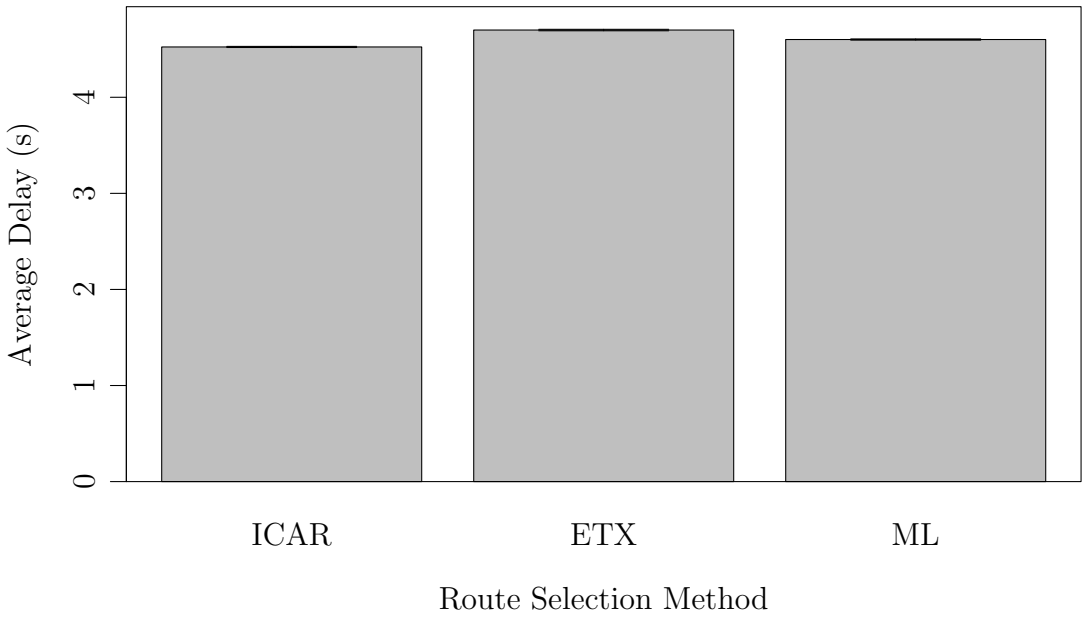


Figure 5.10: Comparison between IAR, ETX, ML, and Hop Count in terms of average delay in the Pro-Interference Scenario with network coding using Simple Broadcast.

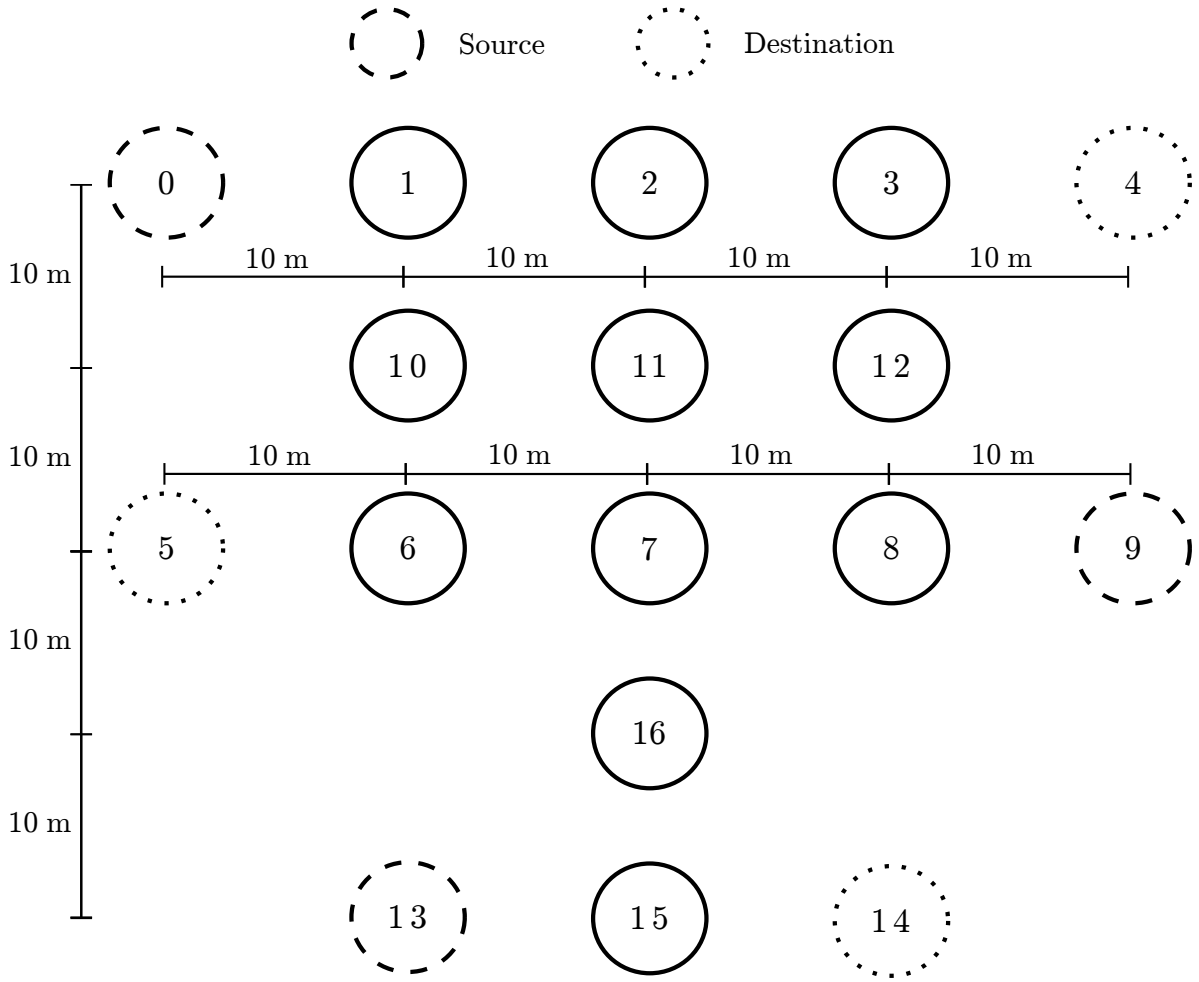


Figure 5.11: Pro-Coding Scenario: a scenario that presents a clear opportunity for avoiding interference with network coding by re-routing two flows through alternative paths, leaving more frequent transmission opportunities for nodes 13 and 15.

Figure 5.11, will be referred to hereinafter as the *Pro-Coding Scenario*.

The Pro-Coding Scenario consists of 17 nodes and three network flows: $0 \Rightarrow 4$, $9 \Rightarrow 5$ and $13 \Rightarrow 14$. The paths that result in the best individual throughputs for each flow in this scenario are $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4$, $9 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 5$, and $13 \rightarrow 15 \rightarrow 14$. However, the distance of 20 meters between nodes from the path $13 \rightarrow 15 \rightarrow 14$ to their counterparts in the path $9 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 5$ would make both flows interfere with each other, resulting in lower than expected throughput. By the same token, flow $0 \Rightarrow 4$ would also interfere with flow $9 \Rightarrow 5$.

One way of improving the aggregated throughput, assuming the network is capable of network coding, is to reroute flows $0 \Rightarrow 4$ and $9 \Rightarrow 5$ through nodes 10, 11 and 12, resulting in paths $0 \rightarrow 10 \rightarrow 11 \rightarrow 12 \rightarrow 4$, and $9 \rightarrow 12 \rightarrow 11 \rightarrow 10 \rightarrow 5$. That would cause coding opportunities to happen both deterministically (at node 11) and

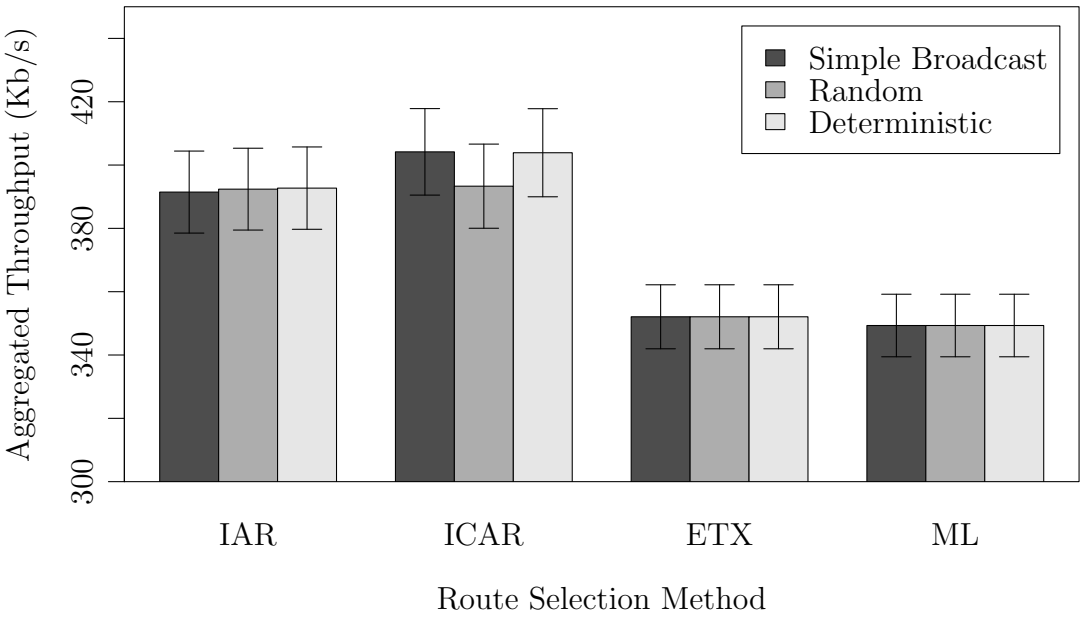


Figure 5.12: Comparison between IAR, ICAR, ETX, and ML in terms of aggregated throughput in the Pro-Coding Scenario. The graph shows results for each of the three transmission methods available for coded packets.

probabilistically (at nodes 10 and 12). Moreover, that change would also increase the distance between flow $13 \Rightarrow 14$ and flow $9 \Rightarrow 5$, decreasing the interference caused by the latter on the former. All those factors combined, result in higher aggregated throughput.

The questions we would like to answer in this scenario, therefore, are whether ICAR is able to detect this opportunity of using network coding and if it can actually conclude that deviating flows from their optimal paths (in individual terms) is better for the overall throughput. To this end, we conducted a series of simulations as follows. Flows $0 \Rightarrow 4$ and $9 \Rightarrow 5$ are generated at a rate of 500 Kb/s using 1450-byte packets. Since viable paths for flow $13 \Rightarrow 14$ tend to be shorter, in our simulations this flow is generated at 1000 Kb/s. All flows start at 500 seconds and end at 1500 seconds. The first 500 seconds are used for the routing information to converge. For each route selection method, the simulations were repeated 5 times, using different seeds. As with the previous scenario, we discarded the first 100 seconds of each flow. In all cases, network coding was enabled.

Figure 5.12 shows the results of aggregated throughput for all available route selection methods under three different methods for transmitting coded packets: Simple Broadcast, Random Pseudo-Broadcast, Deterministic Pseudo-Broadcast. The gains obtained

Table 5.3: Comparison between the routes more frequently chosen by each route selection method in the Pro-Coding Scenario for the flow 9 ⇒ 5. These values correspond to the set of simulations with network coding enabled using Simple Broadcast.

Rank	**IAR**	**ICAR**
1	9 → 12 → 11 → 10 → 5 (62.5%)	9 → 12 → 11 → 10 → 5 (52.8%)
2	9 → 8 → 7 → 6 → 5 (23.3%)	9 → 8 → 7 → 6 → 5 (25.5%)
3	9 → 8 → 12 → 11 → 10 → 5 (10.5%)	9 → 12 → 11 → 10 → 6 → 5 (8.1%)
4	9 → 8 → 12 → 11 → 10 → 6 → 5 (1.6%)	9 → 8 → 12 → 11 → 10 → 5 (6.8%)
Rank	**ETX**	**ML**
1	9 → 8 → 7 → 6 → 5 (91.0%)	9 → 8 → 7 → 6 → 5 (99.6%)
2	9 → 8 → 12 → 11 → 10 → 5 (5.5%)	9 → 8 → 12 → 11 → 10 → 6 → 5 (0.4%)
3	9 → 8 → 12 → 11 → 10 → 6 → 5 (3.5%)	– –
4	– –	– –

by ICAR, in comparison to the results by ETX and ML (14.8% and 15.7%, respectively, using Simple Broadcast), suggest that, indeed, ICAR took advantage of the scenario and managed to reduce the interference on flow 13 ⇒ 14. However, it is interesting to notice that, although not by as much as ICAR, IAR was also able to outperform ETX and ML (11.2% and 12.1%, respectively, using Simple Broadcast). That result is somewhat surprising given that IAR is not aware of the coding capabilities of the network and, therefore, should not be able to find the path set described at the beginning of this section.

The gains of IAR in this scenario with respect to the performances of ETX and ML can be explained by Table 5.3. This table shows the four most frequent routes chosen by each of the route selection methods for the flow 9 ⇒ 5 for the simulations using Simple Broadcast. As shown by the table, both IAR and ICAR rerouted flow 9 ⇒ 5 through nodes 12, 11, and 10 more than 67% of the time (IAR chose such routes even more frequently than ICAR). The difference, however, was that since IAR was not aware of the coding capabilities of the network, it resulted in flow 0 ⇒ 4 using path 0 → 1 → 2 → 3 → 4 98.8% of the time. Hence, IAR was able to find a solution that outperformed ETX and ML by reducing the interference on flow 13 ⇒ 14 without resorting to network coding. This solution, nevertheless, was not as efficient as the one found by ICAR due to the

Table 5.4: Breakdown of the throughputs obtained by each individual flow during the simulations in the Pro-Coding Scenario with network coding using Simple Broadcast. The last row of the table lists, for each route selection method, the resultant Jain's Fairness Index.

Flow	**IAR**	**ICAR**	**ETX**	**ML**
$0 \Rightarrow 4$	57.83 Kb/s	70.61 Kb/s	81.49 Kb/s	82.82 Kb /s
$9 \Rightarrow 5$	36.22 Kb/s	35.69 Kb/s	19.34 Kb/s	17.62 Kb/s
$13 \Rightarrow 14$	297.82 Kb/s	304.34 Kb/s	253.63 Kb/s	249.99 Kb/s
$\mathcal{J}\,(0 \Rightarrow 4, 9 \Rightarrow 5, 13 \Rightarrow 14)$	0.548	0.568	0.587	0.588

latter's ability to consider the possibility of coding packets together.

Another interesting aspect of the results in Figure 5.12 is the comparison between the throughput obtained by each route selection method with each method of transmission of coded packets. ETX and ML selected only paths that did not result in any coding opportunities, while IAR provoked very few of them. But since ICAR relied more heavily on network coding, it shows a greater variation between the throughput achieved by each transmission method. As Chapter 4 and the literature on network coding suggest, Simple Broadcast should result in worse performance than Pseudo-Broadcast (Random or Deterministic) because the latter uses the capability of the MAC layer to retransmit packets, thus reducing packet losses. However, our results do not show any gains in this scenario for the Pseudo-Broadcast variations with respect to the Simple Broadcast (it even shows a slight advantage in favor of Simple Broadcast if one disregards the confidence intervals).

When analyzing these results, it is important to bear in mind that ICAR assumes the usage of Simple Broadcast (*i.e.*, it always considers the transmissions of coded packets to have the duration and probability of success of a single transmission attempt). That is very different from what actually happens when Pseudo-Broadcast is used, since, in this case, the duration of the transmission can be higher than one attempt and the probability of success tends to be higher given the possibility of retries.

Since the routes selected by ICAR are biased in favor of Simple Broadcast (in the sense that the model is not appropriate for Pseudo-Broadcast), these experiments alone cannot be used to compare the performance of the three transmission methods. For the same reason, hereinafter, all simulations with network coding enabled will use only Simple Broadcast[2].

[2]In future work, we intend to extend the network coding model developed in this thesis to consider the other transmission mechanisms.

As one final result from this scenario, Table 5.4 shows a breakdown of the individual throughputs obtained by each flow with each of the available route selection methods. All results correspond to the simulations with Simple Broadcast. As expected, both IAR and ICAR have achieved higher throughputs for flow $13 \Rightarrow 14$ than ETX and ML, since, most of the time, they selected path sets that placed flow $9 \Rightarrow 5$ further away. It is interesting, however, that this move has also improved the individual throughput from flow $9 \Rightarrow 5$, since it also suffers less interference from $13 \Rightarrow 14$.

The difference between IAR and ICAR becomes obvious when one looks at flow $0 \Rightarrow 4$. While both IAR and ICAR cause this flow's performance to drop, with IAR this drop is much more pronounced, since, in this case, this flow suffers more interference from flow $9 \Rightarrow 5$. ICAR manages to mitigate this effect by also rerouting $0 \Rightarrow 4$ to provoke coding opportunities.

One issue that arises with moving flows away from their individual optimal paths (*i.e.*, the path that results in the highest throughput when the flow is the only one in the network) is what happens in terms of fairness. In other words, do ICAR and IAR cause an unfair division of the network resources among flows due to their objective of achieving the optimal aggregated throughput? In the specific case of this scenario, both IAR and ICAR improved the throughput of a high throughput flow ($13 \Rightarrow 14$) at the expense of the performance of a lower throughput one ($0 \Rightarrow 4$). On the other hand, the flow with the worst throughput of the three was improved as well.

As a tool for a more objective analysis, one can resort to Jain's Fairness Index [51], which is shown at the last row of the table for each route selection mechanism. This index can assume values in the range $\left(\frac{1}{n}, 1\right]$ (where n is the number of flows), with 1 meaning a perfectly fair division of resources. In the context of throughput of multiple flows, it is computed using the following expression:

$$\mathcal{J}\left(f_1, \ldots, f_n\right) = \frac{\left(\sum_{i=1}^{n} T_i\right)^2}{n \cdot \sum_{i=1}^{n} T_i^2} \tag{5.1}$$

As shown by the fairness values from ML and ETX, this scenario does not provide a very fair division in terms of throughput, which is expected since nodes 13 and 14 are closer to each other than 0 is of 4 or 9 is of 5. The index suggests, indeed, a drop in fairness with the usage of IAR and ICAR, but this drop is not as dramatic as one could

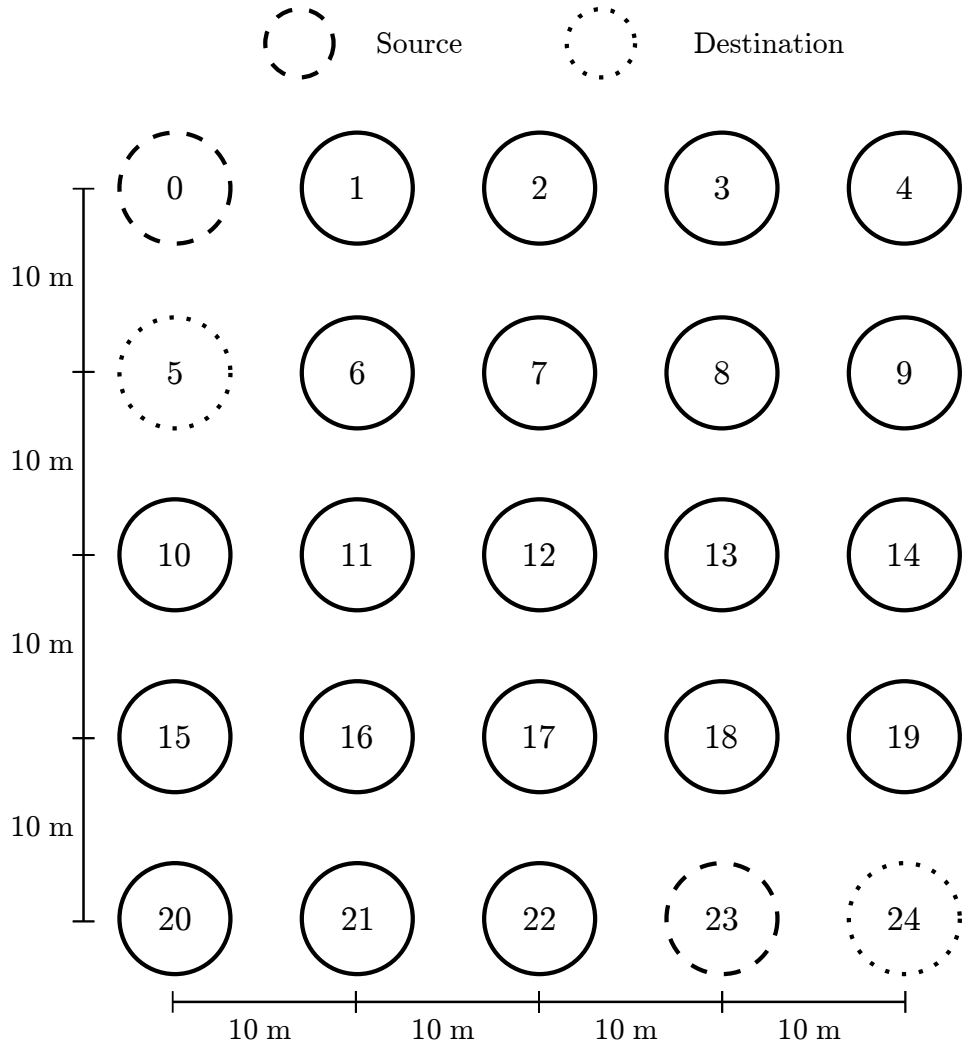


Figure 5.13: Grid Scenario: a neutral topology that presents no clear opportunity for avoiding interference, either by forcing network coding or by separating flows.

expect due to the increase in the throughput of flow $9 \Rightarrow 5$.

5.1.1.3 Grid Scenario

The first two basic scenarios used in this evaluation present clear advantages for route selection methods that are either aware of inter-flow interference or network coding. The third scenario, on the other hand, is completely neutral in this regard. It is a grid topology in which there are two flows as shown in Figure 5.13. The grid is composed by 25 nodes spaced from each other by 10 meters in each dimension.

Differently from the other scenarios used in this evaluation, this topology has slightly different propagation parameters. The path loss exponent was decreased to 3.7, while the shadowing standard deviation was increased to 5.5 dB. The idea was to create a scenario

with a greater variability in terms of links' qualities.

The two flows on this scenario are $0 \Rightarrow 24$ and $23 \Rightarrow 5$. The distance between nodes and the propagation parameters result in links between consecutive nodes in a diagonal (*e.g.*, $0 \rightarrow 6$, $3 \rightarrow 9$) having relatively low delivery probability (around 0.85). For links between non-consecutive nodes in a diagonal, that probability drops rapidly (*e.g.*, around 0.02 for link $0 \rightarrow 12$).

Under these conditions, one intuitive path set would be $\{0 \rightarrow 6 \rightarrow 12 \rightarrow 18 \rightarrow 24,\ 23 \rightarrow 17 \rightarrow 11 \rightarrow 5\}$. By empirically evaluating static path sets in our simulation environment, it was possible to verify that this path set is, indeed, the best possible in this scenario, even with network coding. This happens because this scenario does not provide good enough alternative paths so that it is worth increasing the number of hops or the packet loss probability in order to avoid interference (neither by distancing the flows nor by bringing them closer to one another and relying on coding).

Therefore, the challenge of this scenario is not to find the optimal path set, but instead to maintain that solution selected during the longest period possible. This task is made harder by the high variability in link quality measurements due to both the propagation parameters (with a high shadowing standard deviation) and the density of the network (which can lead to high collision probabilities). The reason for employing such scenario in this evaluation is to assess how sensitive IAR and ICAR are to variations on link quality measurements with respect to the other available route selection methods.

The simulations and post-processing of this scenario were conducted similarly to the ones of the previous scenarios. Both flows were CBR and configured for generating packets at a rate of 200 Kb/s. Flows started at 500 seconds and ended at 1500 seconds. For the averages, the first 100 seconds of the simulation were discarded. Each simulation was repeated 6 times using different seeds.

Figure 5.14 shows the average aggregated throughput for each route selection method with and without network coding. With network coding, both IAR and ICAR achieved considerably higher average throughputs than the other route selection methods (32.9% and 18.2% higher, respectively, than ML, the third best in this case). Without network coding, IAR resulted in good performance as well, achieving an aggregated throughput 40.9% higher than ETX. However, ICAR had a much worse performance in this case, being only 3.2% better than ETX (which is not representative, given the confidence intervals).

Table 5.5 provides some insight to the reason why ICAR performs so poorly in com-

Table 5.5: Comparison between the path sets more frequently chosen by each route selection method in the Pro-Coding Scenario. These values correspond to the set of simulations with network coding enabled using Simple Broadcast.

Rank	**IAR**	**ICAR**
1	$\left\{ \begin{array}{c} 0 \to 6 \to 12 \to 18 \to 24 \\ 23 \to 17 \to 11 \to 5 \end{array} \right\}$ (22.7%)	$\left\{ \begin{array}{c} 0 \to 6 \to 12 \to 18 \to 24 \\ 23 \to 18 \to 12 \to 6 \to 5 \end{array} \right\}$ (11.0%)
2	$\left\{ \begin{array}{c} 0 \to 1 \to 7 \to 13 \to 19 \to 24 \\ 23 \to 22 \to 16 \to 10 \to 5 \end{array} \right\}$ (9.8%)	$\left\{ \begin{array}{c} 0 \to 6 \to 12 \to 18 \to 24 \\ 23 \to 18 \to 12 \to 11 \to 5 \end{array} \right\}$ (8.5%)
3	$\left\{ \begin{array}{c} 0 \to 1 \to 7 \to 13 \to 19 \to 24 \\ 23 \to 17 \to 11 \to 5 \end{array} \right\}$ (9.2%)	$\left\{ \begin{array}{c} 0 \to 1 \to 2 \to 3 \to 9 \to 14 \to 19 \to 24 \\ 23 \to 22 \to 16 \to 10 \to 5 \end{array} \right\}$ (6.9%)
4	$\left\{ \begin{array}{c} 0 \to 6 \to 7 \to 13 \to 19 \to 24 \\ 23 \to 17 \to 11 \to 5 \end{array} \right\}$ (8.7%)	$\left\{ \begin{array}{c} 0 \to 6 \to 12 \to 18 \to 24 \\ 23 \to 17 \to 11 \to 5 \end{array} \right\}$ (6.7%)
Rank	**ETX**	**ML**
1	$\left\{ \begin{array}{c} 0 \to 6 \to 12 \to 18 \to 24 \\ 23 \to 17 \to 11 \to 5 \end{array} \right\}$ (10.6%)	$\left\{ \begin{array}{c} 0 \to 5 \to 11 \to 12 \to 17 \to 23 \to 24 \\ 23 \to 17 \to 12 \to 11 \to 5 \end{array} \right\}$ (4.0%)
2	$\left\{ \begin{array}{c} 0 \to 6 \to 11 \to 17 \to 23 \to 24 \\ 23 \to 17 \to 11 \to 5 \end{array} \right\}$ (7.2%)	$\left\{ \begin{array}{c} 0 \to 1 \to 2 \to 3 \to 9 \to 14 \to 19 \to 24 \\ 23 \to 22 \to 17 \to 11 \to 5 \end{array} \right\}$ (3.9%)
3	$\left\{ \begin{array}{c} 0 \to 1 \to 7 \to 12 \to 17 \to 18 \to 24 \\ 23 \to 17 \to 12 \to 7 \to 6 \to 5 \end{array} \right\}$ (5.7%)	$\left\{ \begin{array}{c} 0 \to 6 \to 11 \to 16 \to 22 \to 23 \to 24 \\ 23 \to 22 \to 16 \to 11 \to 6 \to 5 \end{array} \right\}$ (3.7%)
4	$\left\{ \begin{array}{c} 0 \to 6 \to 11 \to 12 \to 18 \to 24 \\ 23 \to 18 \to 12 \to 11 \to 5 \end{array} \right\}$ (5.4%)	$\left\{ \begin{array}{c} 0 \to 1 \to 2 \to 7 \to 12 \to 17 \to 23 \to 24 \\ 23 \to 17 \to 12 \to 7 \to 6 \to 5 \end{array} \right\}$ (3.2%)

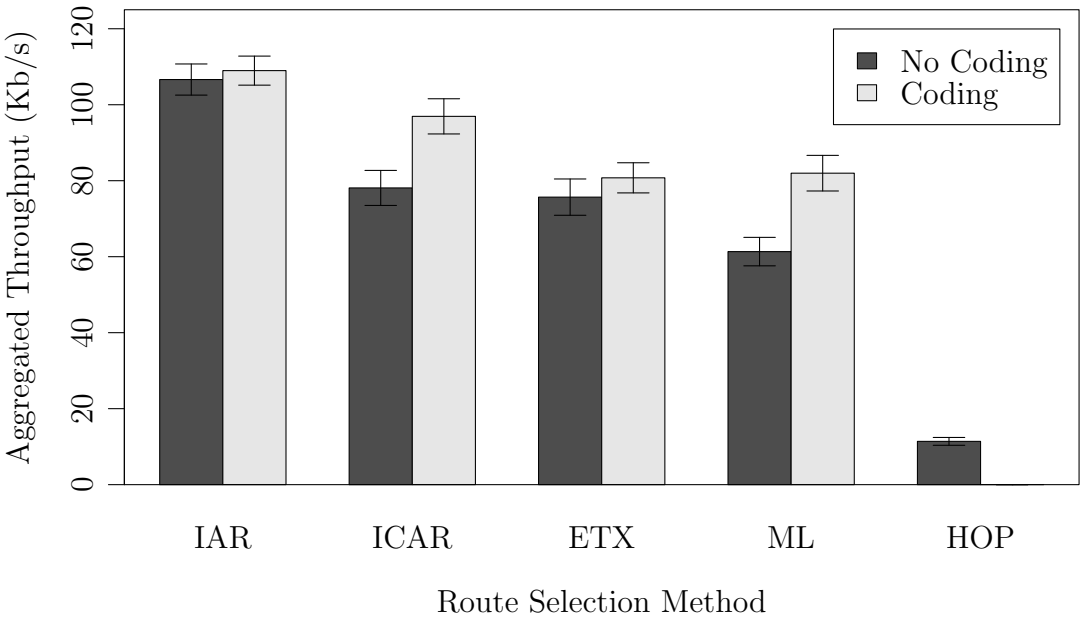


Figure 5.14: Comparison between IAR, ICAR, ETX, ML, and Hop Count in terms of aggregated throughput in the Grid Scenario. Results are shown for both coding-enabled and coding-disabled simulations. In the coding-enabled cases, the Simple Broadcast was used.

parison to IAR when network coding is enabled in this scenario. The table shows the four most used path sets chosen by each route selection method. As expected, all route selection methods suffered from the great variability of this scenario, with the four most frequent routes chosen by ML representing only 14.8% of the time, for example. Nevertheless, it is possible to see that IAR could sustain the optimal choice for the longest period. ICAR, on the other hand, has chosen the optimal path only 6.7% of the time. In fact, the optimal route was not even the most frequently chosen route for ICAR. Instead, its two most frequent selections were two routes that involved network coding. In the simulations in which network coding was available, ICAR actually had good performance, in comparison to ETX and ML (although it was already considerably lower than the throughput obtained by IAR), as shown by Figure 5.14. However, when network coding was disabled, the choice for those paths deeply affected the performance of ICAR.

The fact that ICAR had a worse performance when network coding was disabled is not unexpected, given that its model assumes a coding capable network. However, the fact that ICAR could not identify the optimal path set as being actually optimal as frequently

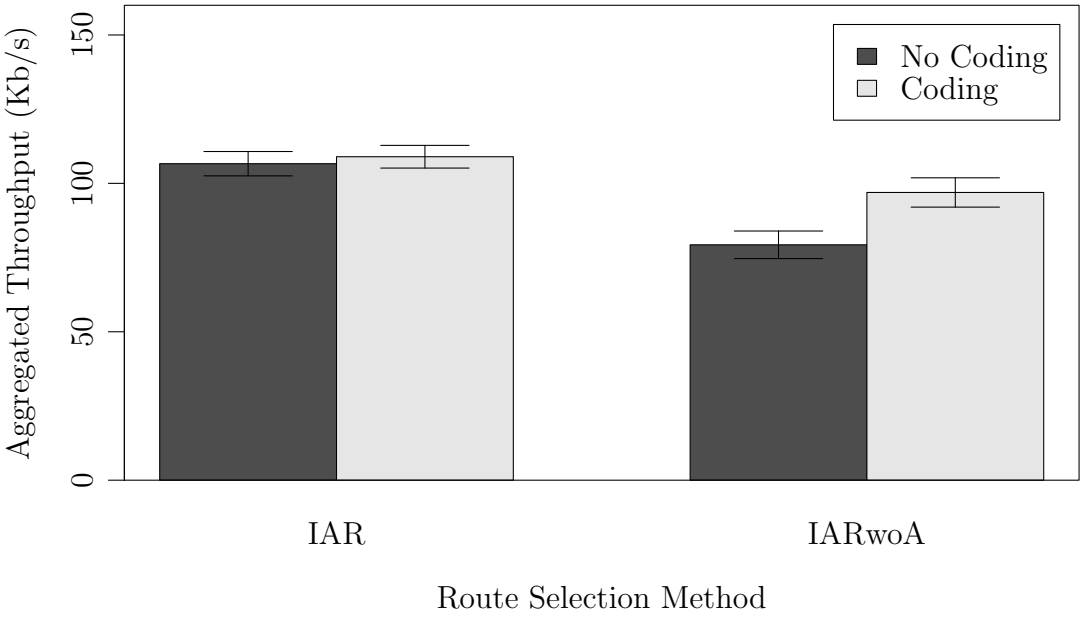


Figure 5.15: Comparison between IAR with and without the probability adjustment technique. The graph shows results of aggregated throughput with and without network coding enabled.

as IAR suggests that the former is more sensitive to variations and errors in link quality measurements than the latter. This is somewhat expected, given the greater complexity of the model used by ICAR, as well as the larger pool of candidates generated by its heuristic for evaluation.

In any case, the results show that both IAR and ICAR (when the network actually supports network coding) could sustain better performance than the other route selection methods in a scenario with great variability. We argue that one of the reasons for such result is the employment of the collision probability adjustment technique, presented on Chapter 4. Figure 5.15 provides data to support such claim. The graph compares the results obtained by IAR in the Grid Scenario with and without network coding against the same setup, but without the collision probability adjustment technique (identified as IARwoA). As the graph shows, the performance of IAR drops considerably without the collision probability adjustment technique: 26.6% without network coding and 11% with it. Although these values are still on par with the other route selection methods (or even better, in the coding enabled case), they suggest the collision probability adjustment technique brings more stability, as expected.

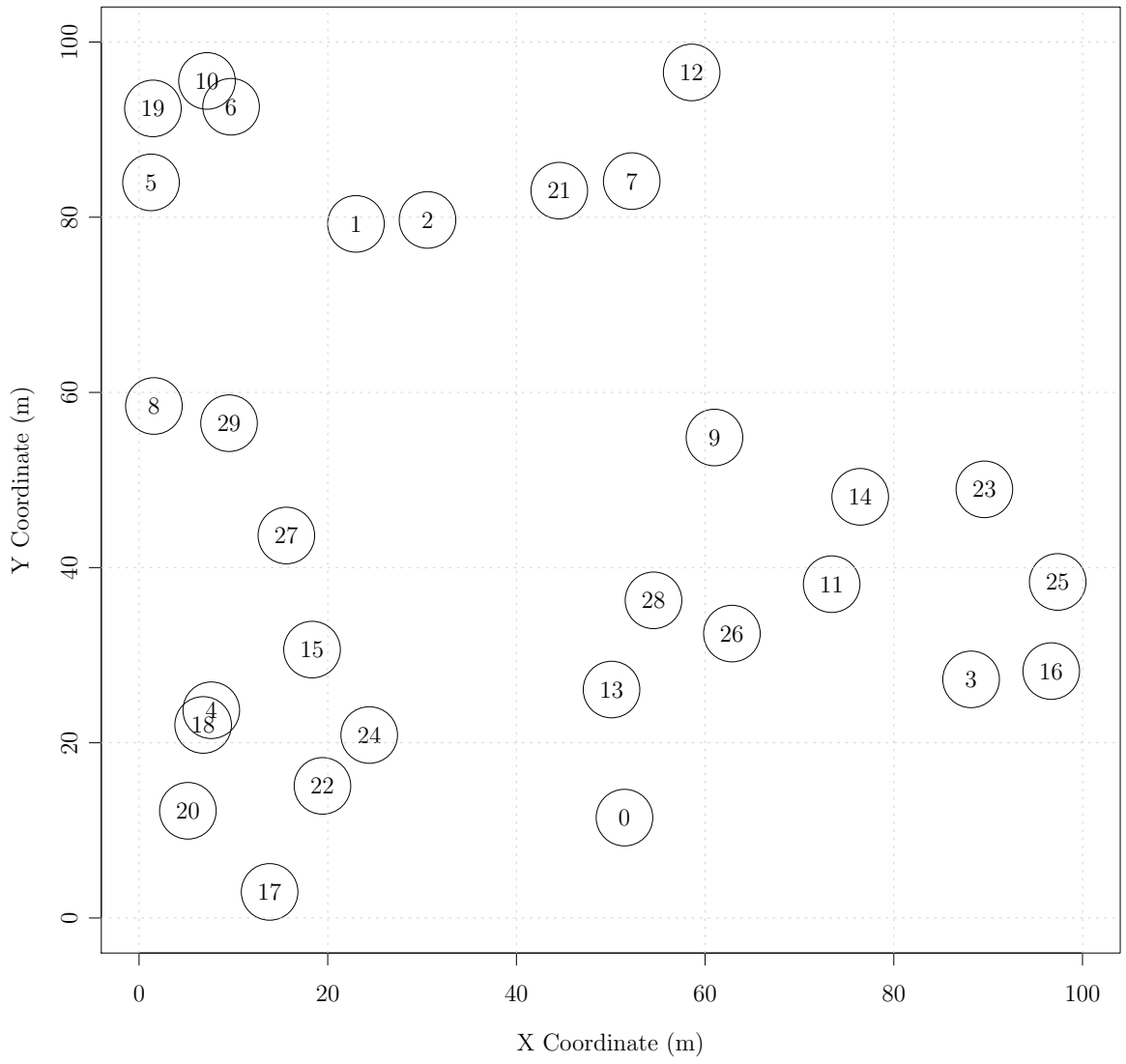


Figure 5.16: Representation of Random Topology I. All 30 nodes are distributed in an area of 100 m X 100 m.

5.1.2 Generic Scenarios

All the scenarios presented so far can be considered simple and one could argue they are not common in practice. Although all those scenarios have been useful for evaluating the performance of IAR and ICAR under certain specific conditions, the question of whether IAR and ICAR can provide gains in more practical scenarios remains.

For this reason, in this section we present simulation results of IAR and ICAR in less structured scenarios, designed to represent more closely the characteristics of a real multihop wireless network. The goal of this section is, thus, to demonstrate that even in more generic scenarios, not specially crafted for presenting specific interference characteristics, there are opportunities for IAR and ICAR to find alternative path sets (*i.e.*, path sets that are not considered optimal by traditional route selection methods) that lead to performance improvement.

To achieve this goal, we generated topologies using the well-known tool `setdest`, which is provided as part of the standard distribution of ns-2. This tool randomly generates wireless network topologies based on simple parameters such as the dimensions of the scenario and the number of nodes. It can also generate mobility patterns based on the random waypoint model. The topologies generated for this evaluation, however, are all static, since mobility is beyond the scope of this thesis.

Two random topologies were generated, from this point on referred to as *Random Topology I* and *Random Topology II*. Both topologies were generated with the same parameters, namely: 30 nodes within an area of 100 m X 100 m. The resultant topologies can be seen in Figures 5.16 and 5.17.

Although randomly generated, Random Topology I actually presents an interesting distribution. Nodes can be somewhat grouped in three clusters. The first cluster comprises nodes of the top half of the topology (above node 1). The second includes nodes from the bottom-left of the scenario (bellow node 8 and to the left of node 24). The remaining nodes compose the third cluster. While Random Topology II does not present such obvious clustering properties, it does have an interesting feature: there is a circular "gap" around coordinates (60, 40), *i.e.*, a circular region of considerable area without any nodes.

The following sections discuss how IAR and ICAR handle unidirectional and bidirectional CBR flows in these two topologies.

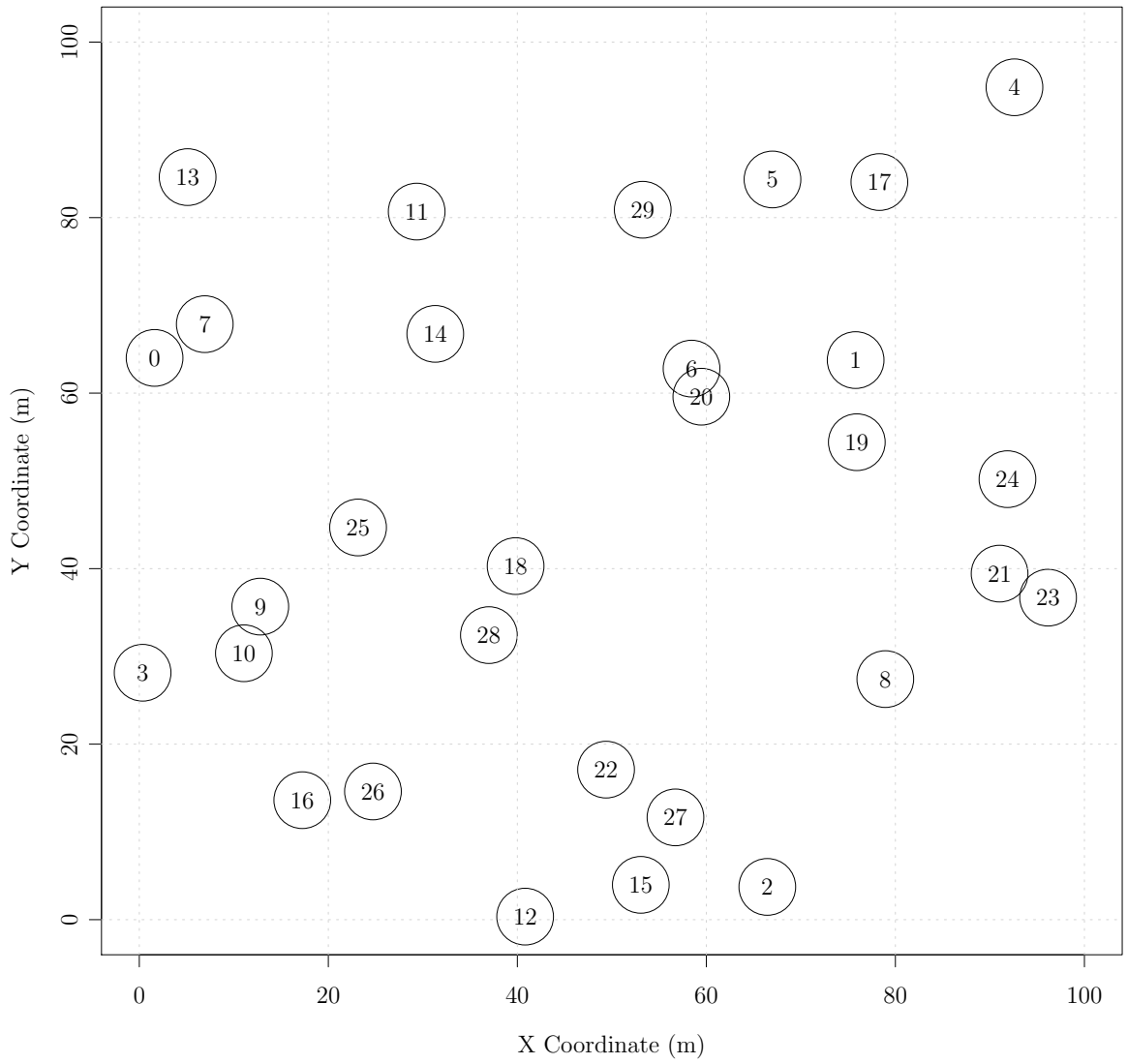


Figure 5.17: Representation of Random Topology II. All 30 nodes are distributed in an area of 100 m X 100 m.

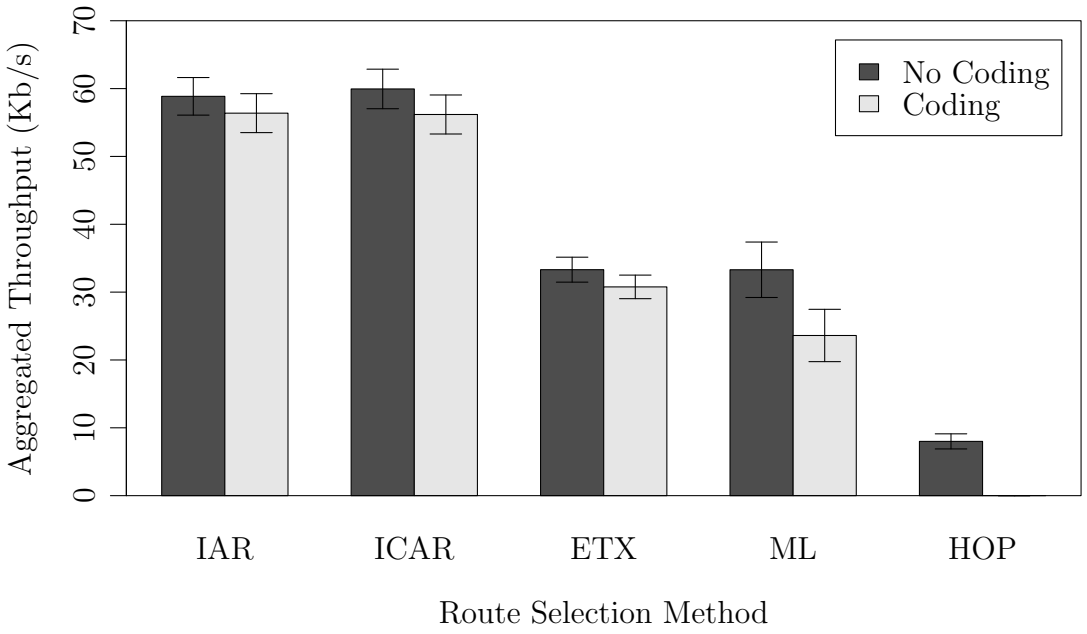


Figure 5.18: Comparison between IAR, ICAR, ETX, ML, and Hop Count in terms of aggregated throughput in the Random Topology I. Results are shown for both coding-enabled and coding-disabled simulations. In the coding-enabled cases, the Simple Broadcast was used.

5.1.2.1 Unidirectional CBR Flows

The first case analyzed in the generic scenarios is that of the unidirectional CBR flows, that is, a scenario with multiple CBR flows each of which has packets flowing in a single direction. To this end, we ran a series of simulations in the Random Topology I with two flows: $12 \Rightarrow 23$ and $13 \Rightarrow 5$. Both flows were configured to generate 1450-byte packets at a rate of 100 Kb/s, which is enough to saturate the capacity of the network. As with other experiments, both flows started at 500 seconds and ended at 1500 seconds. For the statistical analysis that follows, we discard the first 100 seconds of each flow. Each simulation was repeated 6 times with different seeds.

Figure 5.18 shows the average aggregated throughput obtained by each route selection method with and without network coding (using Simple Broadcast). Except for ML, network coding did not result in much different throughput in this scenario. Actually, those results consistently show slightly better performance without network coding. In any case, IAR and ICAR clearly improved aggregated throughput with respect to the

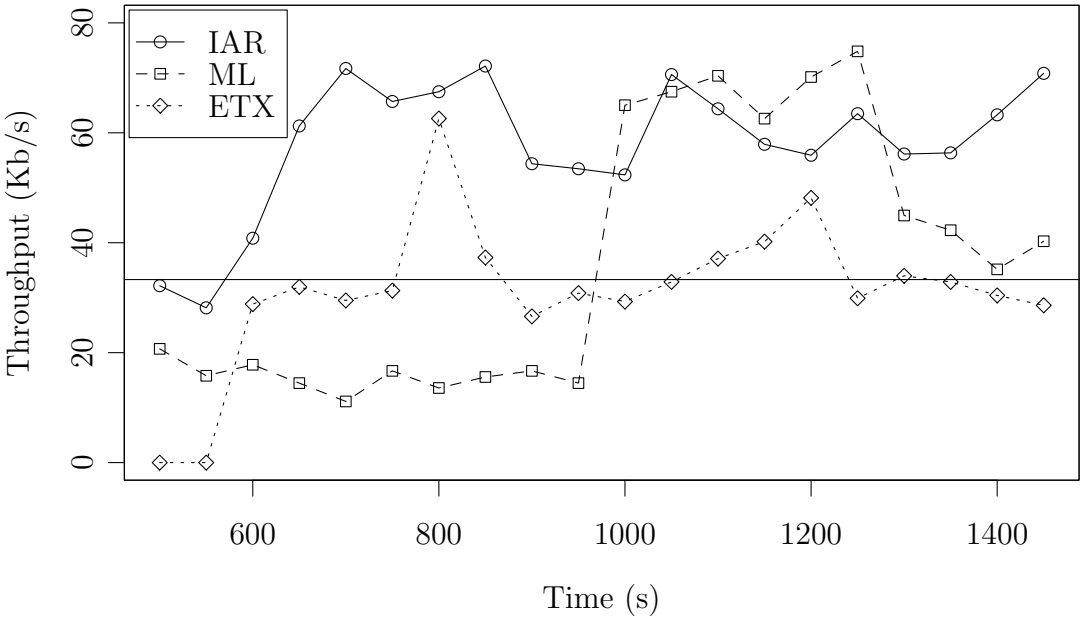


Figure 5.19: Comparison between the behavior of the aggregated throughput obtained by IAR, ETX, and ML during one of the simulations in the Random Topology I. Results are shown for coding-disabled simulations.

other route selection methods. When comparing results from simulations without network coding, for example, IAR showed an improvement of 76.7% with respect to ETX (which obtained the third best result in this case).

The similar results obtained by ICAR with and without network coding suggest that the solutions found by this method (and, consequently, by IAR) did not involve bringing the flows together for coding. Instead, both IAR and ICAR took advantage of an opportunity to reduce inter-flow interference by separating the flows. That can be seen in Table 5.6, which shows the four most frequently chosen path sets for each route selection method. All four path sets chosen by ETX present a similar trend: for establishing a route between nodes 12 (which is located in the top cluster) and 23 (which is located in the bottom right-cluster), ETX chooses more frequently to pass through nodes from the bottom-left cluster (such as 29, 27, 15 and 24), in order to avoid link $7 \rightarrow 9$, which is long and lossy. The problem with such choice is that the best path for flow $13 \Rightarrow 5$ also uses nodes from the bottom-left cluster, meaning both flows have to share resources, thus interfering with each other. On the other hand, both IAR and ICAR opt for rerouting flow $12 \Rightarrow 23$ through links $7 \rightarrow 9$, $7 \rightarrow 14$ or $21 \rightarrow 9$, avoiding the bottom-left cluster. This

Table 5.6: Comparison between the path sets more frequently chosen by each route selection method in the Random Topology I with unidirectional CBR flows. These values correspond to the set of simulations without network coding.

Rank	**IAR**	**ICAR**
1	$\left\{\begin{array}{c} 13 \to 24 \to 15 \to 27 \to 29 \to 8 \to 5 \\ 12 \to 7 \to 9 \to 14 \to 23 \end{array}\right\}$ (48.2%)	$\left\{\begin{array}{c} 13 \to 24 \to 15 \to 27 \to 29 \to 8 \to 5 \\ 12 \to 7 \to 9 \to 14 \to 23 \end{array}\right\}$ (58.0%)
2	$\left\{\begin{array}{c} 13 \to 24 \to 15 \to 27 \to 29 \to 1 \to 6 \to 5 \\ 12 \to 7 \to 9 \to 14 \to 23 \end{array}\right\}$ (15.1%)	$\left\{\begin{array}{c} 13 \to 24 \to 15 \to 27 \to 29 \to 1 \to 6 \to 5 \\ 12 \to 7 \to 9 \to 14 \to 23 \end{array}\right\}$ (9.9%)
3	$\left\{\begin{array}{c} 13 \to 24 \to 15 \to 27 \to 29 \to 8 \to 5 \\ 12 \to 7 \to 21 \to 9 \to 14 \to 23 \end{array}\right\}$ (9.9%)	$\left\{\begin{array}{c} 13 \to 24 \to 15 \to 27 \to 29 \to 8 \to 5 \\ 12 \to 7 \to 21 \to 9 \to 14 \to 23 \end{array}\right\}$ (8.5%)
4	$\left\{\begin{array}{c} 13 \to 24 \to 15 \to 27 \to 29 \to 1 \to 6 \to 5 \\ 12 \to 7 \to 21 \to 9 \to 14 \to 23 \end{array}\right\}$ (6.5%)	$\left\{\begin{array}{c} 13 \to 24 \to 15 \to 27 \to 29 \to 1 \to 6 \to 5 \\ 12 \to 7 \to 14 \to 23 \end{array}\right\}$ (4.0%)

Rank	**ETX**
1	$\left\{\begin{array}{c} 13 \to 24 \to 15 \to 27 \to 29 \to 8 \to 5 \\ 12 \to 7 \to 21 \to 2 \to 1 \to 6 \to 5 \to 8 \to 29 \to 27 \to 15 \to 24 \to 13 \to 26 \to 11 \to 14 \to 23 \end{array}\right\}$ (63.3%)
2	$\left\{\begin{array}{c} 13 \to 24 \to 15 \to 27 \to 29 \to 8 \to 5 \\ 12 \to 7 \to 21 \to 2 \to 1 \to 29 \to 27 \to 15 \to 24 \to 13 \to 26 \to 11 \to 14 \to 23 \end{array}\right\}$ (12.0%)
3	$\left\{\begin{array}{c} 13 \to 24 \to 15 \to 27 \to 29 \to 1 \to 6 \to 5 \\ 12 \to 7 \to 21 \to 2 \to 1 \to 29 \to 27 \to 15 \to 24 \to 13 \to 26 \to 11 \to 14 \to 23 \end{array}\right\}$ (7.3%)
4	$\left\{\begin{array}{c} 13 \to 24 \to 15 \to 27 \to 29 \to 8 \to 5 \\ 12 \to 7 \to 9 \to 14 \to 23 \end{array}\right\}$ (6.4%)

Rank	**ML**	**Hop Count**
1	$\left\{\begin{array}{c} 13 \to 24 \to 15 \to 27 \to 29 \to 8 \to 5 \\ 12 \to 7 \to 9 \to 14 \to 23 \end{array}\right\}$ (33.2%)	$\left\{\begin{array}{c} 13 \to 27 \to 5 \\ 12 \to 9 \to 23 \end{array}\right\}$ (18.9%)
2	$\left\{\begin{array}{c} 13 \to 27 \to 29 \to 8 \to 5 \\ 12 \to 7 \to 9 \to 14 \to 23 \end{array}\right\}$ (11.7%)	$\left\{\begin{array}{c} 13 \to 24 \to 8 \to 5 \\ 12 \to 9 \to 23 \end{array}\right\}$ (12.3%)
3	$\left\{\begin{array}{c} 13 \to 24 \to 15 \to 27 \to 29 \to 8 \to 5 \\ 12 \to 9 \to 14 \to 23 \end{array}\right\}$ (6.6%)	$\left\{\begin{array}{c} 13 \to 24 \to 29 \to 5 \\ 12 \to 9 \to 23 \end{array}\right\}$ (12.0%)
4	$\left\{\begin{array}{c} 13 \to 28 \to 21 \to 2 \to 1 \to 6 \to 5 \\ 12 \to 7 \to 9 \to 14 \to 23 \end{array}\right\}$ (6.4%)	$\left\{\begin{array}{c} 13 \to 27 \to 5 \\ 12 \to 7 \to 9 \to 23 \end{array}\right\}$ (9.4%)

Table 5.7: Breakdown of the throughputs obtained by each individual flow during the simulations in the Random Topology I with unidirectional CBR flows. These values correspond to the simulations without network coding. The last row of the table lists, for each route selection method, the resultant Jain's Fairness Index.

Flow	**IAR**	**ICAR**	**ETX**	**ML**	**HOP**
$12 \Rightarrow 23$	25.43 Kb/s	24.05 Kb/s	15.16 Kb/s	17.77 Kb/s	5.70 Kb/s
$13 \Rightarrow 5$	34.34 Kb/s	36.13 Kb/s	18.81 Kb/s	15.99 Kb/s	2.31 Kb/s
$\mathcal{J}\,(12 \Rightarrow 23, 13 \Rightarrow 5)$	0.978	0.961	0.989	0.997	0.848

reduces the interference between flows, increasing aggregated throughput. Hop Count also takes this strategy, although it selects very long links, resulting in poor performance anyway.

Another interesting information shown by Table 5.6 concerns the choices made by ML. Except for the fourth most frequent path set, ML's choices resemble those made by IAR and ICAR in the sense that they all separate both flows, reducing interference between them. Nevertheless, as Figure 5.18 shows, ML had an average aggregated throughput very similar to the obtained by ETX. Figure 5.19 provides further insight into this issue. It shows the behavior of the aggregated throughput as a function of the time for the simulations with IAR, ETX, and ML without network coding and using one specific seed. The graph also shows a horizontal line marking the value of average aggregated throughput obtained by ML overall. It is possible to see that, while ML could, at times, reach the same level of performance achieved by IAR (from 1000 seconds to 1250 seconds), at other moments it would make choices that would cause its performance to drop considerably bellow its own average (from 500 seconds to 950 seconds). ETX, on the other hand, had a much more stable performance, presenting a small oscillation throughout the simulation.

The reason for the bad performance by ML during intervals such as the one shown in Figure 5.19 is the option for path sets that involve both flows avoiding the bottom-left cluster. In that case, not only both paths would use lossy links such as $9 \rightarrow 7$, but they would also interfere with each other, resulting in very low aggregated throughput.

Table 5.7 shows the individual throughput values obtained by each route selection method for each flow in this scenario. The last row of the table shows the Jain's Fairness Index obtained by each method. Once again, IAR and ICAR resulted in lower fairness indexes than the other methods. However, it is interesting to notice that the strategy adopted by IAR and ICAR was able to greatly improve the throughput of both flows, only not uniformly. In other words, although IAR and ICAR rerouted flow $12 \Rightarrow 23$

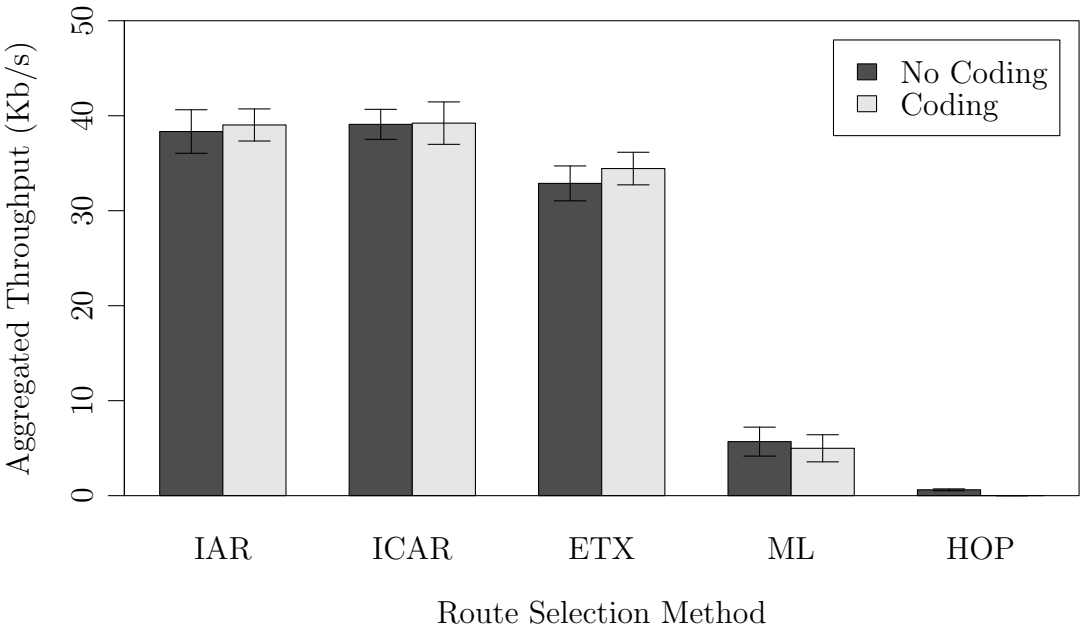


Figure 5.20: Comparison between the average aggregated throughputs obtained by IAR, ICAR, ETX, ML, and Hop Count during simulations in the Random Topology II for the symmetric bidirectional CBR flow 9 ⇔ 24. Results are shown for both coding-enabled and disabled simulations.

through a path that is not optimal (in terms of that individual flow), this choice led to an improvement for both flows in terms of throughput.

5.1.2.2 Symmetric Bidirectional CBR Flows

The last section presented a scenario in which IAR and ICAR were able to improve network aggregated throughput by increasing the distance between two distinct unidirectional flows. In that case, it was somewhat easy to increase the distance between those flows, partially because both flows were completely unrelated, *i.e.*, neither the source or destination nodes from one flow were source or destination of the other.

However, a common trait of many network flows in practice is the bidirectionality, meaning that in these flows data is sent in both directions. This raises the question of whether IAR and ICAR are able to find alternative paths for one or both directions in order to avoid interference between them so that the performance can be improved.

In this section, we analyze scenarios containing symmetric bidirectional CBR flows.

In this context, the word *symmetric* is used to refer to bidirectional flows that generate packets at the same rate in both directions. To this end, we created a scenario based on Random Topology II containing a single symmetric bidirectional flow between nodes 9 and 24. For these simulations, the flow was configured to generate packets at a rate of 100 Kb/s in both directions. As with the previous experiments, the flow was initiated at 500 seconds and ended at 1500 seconds, while the first 100 seconds were discarded for computing statistics. Each simulation was repeated 4 times with different seeds.

Figure 5.20 shows results of aggregated throughput obtained with and without network coding. As in the results presented in the previous section, IAR and ICAR performed very similarly both with and without network coding. Therefore, this again suggests that the routing solutions they found for this scenario likely do not rely on network coding. Both with and without network coding, though, IAR and ICAR were able to perform considerably better than ETX, the third best route selection method in this scenario. For example, without network coding, IAR and ICAR resulted in gains of 16.6% and 18.9%, respectively, with respect to ETX.

To help understanding what led to the difference in performance between IAR (and ICAR) and ETX, Table 5.8 shows the 4 most frequent routes chosen by IAR, ETX, ML, and Hop Count in this scenario during the coding-disabled simulations. As expected, ETX has mostly opted for symmetrical paths. Its three most frequent path sets all follow the same pattern: both routes circulate the gap of the topology by its upper side. The fourth most frequent path set is also symmetrical, but instead uses nodes below the gap. In both cases, though, packets from each direction of the flow compete with each other both for medium usage and for buffer space in the intermediate nodes.

IAR, on the other hand, was able to find alternative path sets that took advantage from the scenario to spread each direction of the flow through different regions of the topology. In fact, all four most frequent path sets employed by IAR present that same characteristic. That led to a decrease in inter-flow interference, resulting in improved aggregated throughput.

As Table 5.9 shows (specifically for the simulations without network coding), the choice for non-interfering paths for each direction of the flow improved the throughput of both flows, even if, at times, one of the flows had to use a non-optimal path (individually speaking). That had already happened in the scenario used to evaluate unidirectional flows and reinforces the concept that treating flows individually in multi-flow scenarios may lead to suboptimal performance, not only globally, but also for each individual flow.

Table 5.8: Comparison between the path sets more frequently chosen for symmetric bidirectional flow 9 ⇔ 24 in the Random Topology II without network coding.

Rank	**IAR**
1	$\left\{\begin{array}{c} 24 \to 19 \to 1 \to 6 \to 29 \to 11 \to 14 \to 25 \to 9 \\ 9 \to 10 \to 16 \to 26 \to 12 \to 15 \to 2 \to 8 \to 21 \to 24 \end{array}\right\}$ (27.1%)
2	$\left\{\begin{array}{c} 24 \to 21 \to 8 \to 2 \to 15 \to 12 \to 26 \to 16 \to 10 \to 9 \\ 9 \to 25 \to 14 \to 11 \to 29 \to 5 \to 17 \to 1 \to 19 \to 24 \end{array}\right\}$ (8.8%)
3	$\left\{\begin{array}{c} 24 \to 19 \to 20 \to 6 \to 29 \to 11 \to 14 \to 25 \to 9 \\ 9 \to 10 \to 16 \to 26 \to 12 \to 15 \to 2 \to 8 \to 21 \to 24 \end{array}\right\}$ (7.3%)
4	$\left\{\begin{array}{c} 24 \to 21 \to 8 \to 2 \to 15 \to 12 \to 26 \to 16 \to 10 \to 9 \\ 9 \to 25 \to 14 \to 11 \to 29 \to 6 \to 1 \to 19 \to 24 \end{array}\right\}$ (7.3%)
Rank	**ETX**
1	$\left\{\begin{array}{c} 24 \to 19 \to 1 \to 6 \to 29 \to 11 \to 14 \to 25 \to 9 \\ 9 \to 25 \to 14 \to 11 \to 29 \to 6 \to 1 \to 19 \to 24 \end{array}\right\}$ (43.7%)
2	$\left\{\begin{array}{c} 24 \to 19 \to 6 \to 29 \to 11 \to 14 \to 25 \to 9 \\ 9 \to 25 \to 14 \to 11 \to 29 \to 6 \to 19 \to 24 \end{array}\right\}$ (10.2%)
3	$\left\{\begin{array}{c} 24 \to 19 \to 20 \to 6 \to 29 \to 11 \to 14 \to 25 \to 9 \\ 9 \to 25 \to 14 \to 11 \to 29 \to 6 \to 20 \to 19 \to 24 \end{array}\right\}$ (8.4%)
4	$\left\{\begin{array}{c} 24 \to 21 \to 8 \to 27 \to 22 \to 28 \to 25 \to 9 \\ 9 \to 25 \to 28 \to 22 \to 27 \to 8 \to 21 \to 24 \end{array}\right\}$ (7.7%)
Rank	**ML**
1	$\left\{\begin{array}{c} 24 \to 19 \to 18 \to 25 \to 9 \\ 9 \to 25 \to 20 \to 19 \to 24 \end{array}\right\}$ (24.6%)
2	$\left\{\begin{array}{c} 24 \to 19 \to 20 \to 18 \to 25 \to 9 \\ 9 \to 25 \to 18 \to 20 \to 19 \to 24 \end{array}\right\}$ (8.7%)
3	$\left\{\begin{array}{c} 24 \to 19 \to 20 \to 18 \to 25 \to 9 \\ 9 \to 25 \to 18 \to 6 \to 20 \to 19 \to 24 \end{array}\right\}$ (8.0%)
4	$\left\{\begin{array}{c} 24 \to 19 \to 18 \to 25 \to 9 \\ 9 \to 25 \to 18 \to 6 \to 20 \to 1 \to 19 \to 24 \end{array}\right\}$ (4.3%)
Rank	**HOP**
1	$\left\{\begin{array}{c} 24 \to 19 \to 1 \to 6 \to 29 \to 11 \to 14 \to 25 \to 9 \\ 9 \to 10 \to 16 \to 26 \to 12 \to 15 \to 2 \to 8 \to 21 \to 24 \end{array}\right\}$ (27.1%)
2	$\left\{\begin{array}{c} 24 \to 21 \to 8 \to 2 \to 15 \to 12 \to 26 \to 16 \to 10 \to 9 \\ 9 \to 25 \to 14 \to 11 \to 29 \to 5 \to 17 \to 1 \to 19 \to 24 \end{array}\right\}$ (8.8%)
3	$\left\{\begin{array}{c} 24 \to 19 \to 20 \to 6 \to 29 \to 11 \to 14 \to 25 \to 9 \\ 9 \to 10 \to 16 \to 26 \to 12 \to 15 \to 2 \to 8 \to 21 \to 24 \end{array}\right\}$ (7.3%)
4	$\left\{\begin{array}{c} 24 \to 21 \to 8 \to 2 \to 15 \to 12 \to 26 \to 16 \to 10 \to 9 \\ 9 \to 25 \to 14 \to 11 \to 29 \to 6 \to 1 \to 19 \to 24 \end{array}\right\}$ (7.3%)

Table 5.9: Breakdown of the throughputs obtained by each individual flow during the simulations in the Random Topology II with symmetric bidirectional flow $9 \Leftrightarrow 24$. These values correspond to the simulations without network coding. The last row of the table lists, for each route selection method, the resultant Jain's Fairness Index.

Flow	**IAR**	**ETX**	**ML**	**HOP**
$9 \Rightarrow 24$	19.07 Kb/s	15.40 Kb/s	3.11 Kb/s	0.39 Kb/s
$24 \Rightarrow 9$	20.05 Kb/s	17.92 Kb/s	2.62 Kb/s	0.24 Kb/s
$\mathcal{J}(9 \Rightarrow 24, 24 \Rightarrow 9)$	0.999	0.994	0.993	0.947

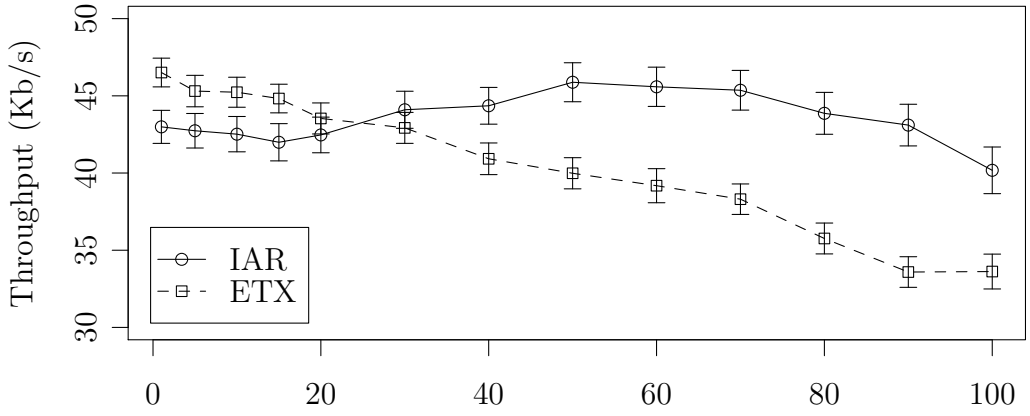
Interestingly, IAR actually resulted in a better fairness index for this scenario, albeit by a small margin.

5.1.3 Asymmetric CBR Flows

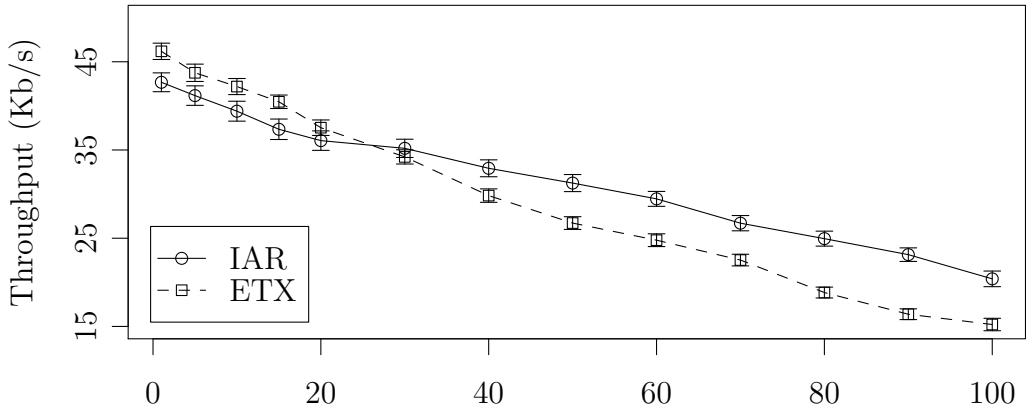
Up to this point, most scenarios used in this evaluation have employed symmetric flows, in the sense that all flows generate packets at the same rate. In this section we study the effect of asymmetry in the performance of IAR and ICAR for both the unidirectional and the bidirectional cases. Specifically, we seek to answer whether the usage of asymmetrical traffic can decrease the gains obtained by IAR and ICAR or even make their route selection decisions worse than those made by other route selection methods. As discussed in Chapter 3, the model adopted by IAR and ICAR supposes the flows' sources are always backlogged. If a flow does not generate packets at a sufficient rate, that may have an impact in route selection because that flow may not be generating enough interference on others so that it is worth to reroute it (or the other flows suffering interference from it) to an alternative path. This may also affect ICAR's model for network coding, since a low rate flow may result in fewer coding opportunities than predicted during route selection.

To evaluate the impact of asymmetry in the performances of IAR and ICAR, we revisit two scenarios already used in this evaluation. First, we analyze the impact of asymmetry in the Random Topology II, in the same scenario used in Section 5.1.2.2. To this end, we repeat the set of simulations performed in that section without network coding with the same basic methodology. However, this time, instead of fixating both flows at 100 Kb/s, we decrease the rate for the direction $24 \Rightarrow 9$ in steps to verify what is the impact to the aggregated throughput for each route selection method.

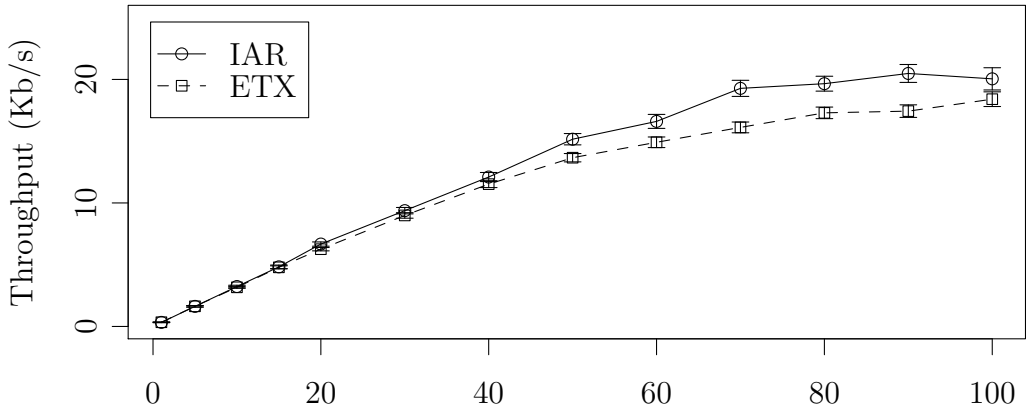
The graphs in Figure 5.21 show the results obtained in this scenario. Since ML and Hop Count obtained much worse results than ETX and IAR in this scenario, they were left out of those graphs to improve legibility. Each point of the graphs corresponds to the



(a) Aggregated



(b) 9 ⇒ 24



(c) 24 ⇒ 9

Figure 5.21: Comparison between IAR and ETX in the Random Topology II with the asymmetric flow 9 ⇔ 24. The transmission rate was kept constant in the direction 9 ⇒ 24 but varied between 1 Kb/s and 100 Kb/s for the opposite one. The top graph shows the aggregated throughput, while the two others show the throughput in each direction.

average of 20 simulations with different seeds. As shown by Figure 5.21b, when the rate for direction 24 ⇒ 9 is lower than 30 Kb/s, ETX actually obtains a higher throughput for flow 9 ⇒ 24, which results in a higher aggregated throughput (Figure 5.21a). This happens because IAR disregards the actual transmission rate of the flows in its model and deviates one or both flows from their individual optimal paths. Since the interference caused by flow 24 ⇒ 9 on flow 9 ⇒ 24 is low (due to the low rate), keeping the latter on its individual optimal path is actually the best option.

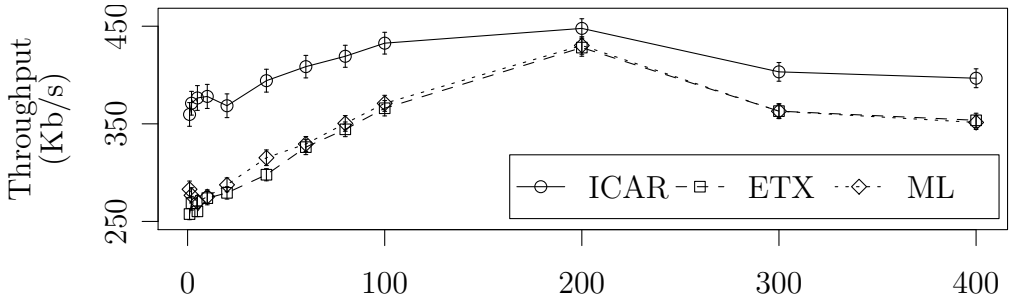
As the rate for flow 24 ⇒ 9 increases, the throughput for flow 9 ⇒ 24 decreases for both IAR and ETX. However, the decreasing rate is different for both route selection methods. Since ETX tends to choose symmetrical paths, it suffers more intensely due to interference. For this reason, when the rate for flow 24 ⇒ 9 reaches 30 Kb/s, IAR achieves better throughput results for the flow 9 ⇒ 24 and for the aggregated result. The gap between results obtained by IAR and ETX continues to grow until the transmission rate reaches 100 Kb/s, at which point the paths are already saturated for both route selection methods.

These results demonstrate that, indeed, asymmetry can lead IAR to obtain sub-optimal performance. Nevertheless, IAR only started to perform worse than ETX in this scenario for highly asymmetric transmission rates.

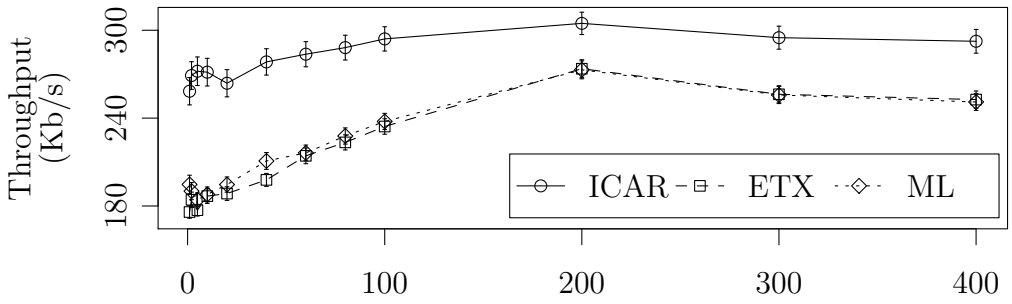
As a second experiment involving asymmetric flows, we revisit the Pro-Coding Scenario to evaluate how asymmetry affects the performance of path sets based on network coding. In this case, flows 13 ⇒ 14 and 9 ⇒ 5 kept their original packet generation rates (1000 Kb/s and 500 Kb/s, respectively), while flow 0 ⇒ 4 had its rate ranging from 1 Kb/s to 400 Kb/s. All simulations were run with network coding enabled and were repeated 20 times with different seeds.

The throughput results for this scenario can be seen on the graphs in Figure 5.22. When flow 0 ⇒ 4 has a low rate, the graphs show that ICAR still outperforms both ETX and ML by a considerable margin (by 39.62% and 27.10%, respectively). Although the low transmission rate decreases the number of coding opportunities (which is not predicted by ICAR), flow 9 ⇒ 5 is still rerouted away from flow 13 ⇒ 14, causing both flows to perform better than they do with ETX and ML (as shown in Figures 5.22b and 5.22d).

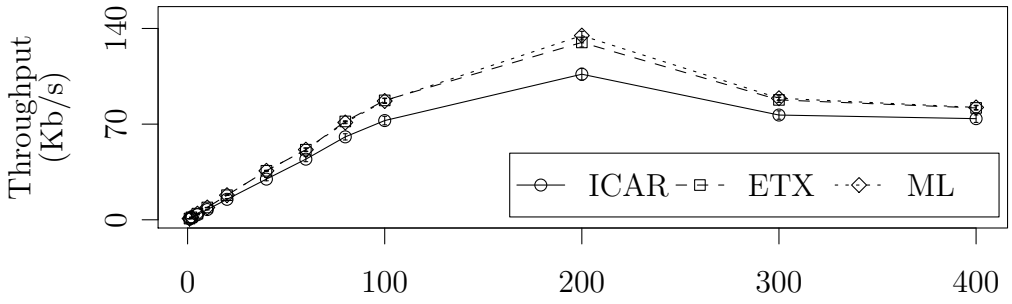
As the rate for flow 0 ⇒ 4 increases, the throughput for flow 9 ⇒ 5 decreases due to an increase in interference. This decrease in throughput happens approximately at the same rate for ICAR, ETX and ML, which maintains the performance gap for this flow. Although ICAR routes flow 9 ⇒ 5 closer to flow 0 ⇒ 4 than ETX and ML, the increase



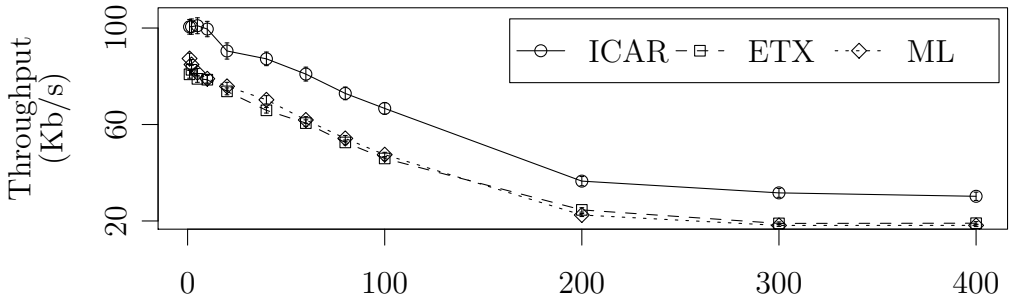
(a) Aggregated



(b) 13 ⇒ 14



(c) 0 ⇒ 4



(d) 9 ⇒ 5

Figure 5.22: Comparison between IAR, ETX, and ML in the Pro-Coding Scenario with asymmetric flows. Rate was varied from 1 Kb/s to 400 Kb/s for flow 0 ⇒ 4.

in interference is mitigated by the increase in coding opportunities. When flow $0 \Rightarrow 4$ reaches 200 Kb/s, throughput is maximum for flow $0 \Rightarrow 4$, regardless of route selection method. Beyond that point, the network becomes saturated and performance drops due to node 0 competing more for accessing the wireless medium.

Another interesting information shown by these graphs, is that the gap in performance between ICAR and the other two methods is lower for heavier loads of $0 \Rightarrow 4$ than it is for lighter ones. This happens mostly due to an increase in performance for flow $13 \Rightarrow 14$ for ETX and ML. As the rate of flow $0 \Rightarrow 4$ increases, it creates more interference for flow $9 \Rightarrow 5$. As a consequence, the amount of time which flow $9 \Rightarrow 5$ can use to transmit its packets decreases, leading not only to a decrease in its own performance, but also to an increase in the performance of flow $13 \Rightarrow 14$ due to less competition. Although this phenomenon also happens with ICAR, it is less pronounced in this case, given that flow $9 \Rightarrow 5$ has less impact on flow $13 \Rightarrow 14$ due to the alternative path used by the former.

5.1.4 TCP Streams

All results presented so far were obtained by simulating CBR flows. Nevertheless, a representative share of networking applications run on the top of TCP streams [94]. A TCP stream presents different characteristics, if compared to CBR flows. For one, TCP streams are naturally bidirectional, even in the absence of data in one of the directions, due to the transmission of ACK segments. In this case, a TCP stream would be better classified as an asymmetric bidirectional flow, since ACK segments tend to be much smaller than data segments. But there is yet another relevant difference: while CBR flows, by definition, generate packets at constant rate, TCP tries to adapt its transmission rate to the capacity of the network, thus resulting in a rate that varies with time.

At first glance, the adaptive operation of TCP could interact well with the model adopted by IAR and ICAR, because it intrinsically assumes each flow is transmitted at the highest transmission supported by the path. However, TCP usually transmits in bursts, which is not predicted by the model. Moreover, traditional versions of TCP are known to underperform in wireless networks due to their algorithms being unable to discern between losses caused by wireless phenomena and by network congestion [95]. In those cases, TCP is not able to reach or sustain transmission rates close to the actual path capacity.

Due to all those reasons, the conclusions found so far in this chapter cannot be automatically extended to the cases involving TCP streams. Therefore, in this section we

Table 5.10: Comparison between IAR, ETX, ML, and Hop Count in terms of aggregated throughput in the Pro-Interference Scenario with a single unidirectional TCP stream from node 0 to node 1. These results correspond to simulations without network coding.

IAR	ETX	ML	Hop Count
129.48 ± 2.17	125.72 ± 2.11	126.32 ± 2.12	0.02 ± 0.004

extend our analysis to these cases. In the next sections, results will be presented concerning both unidirectional and bidirectional TCP streams.

5.1.4.1 Unidirectional TCP Streams

As a first experiment with unidirectional TCP streams, we use a variation of the Pro-Interference scenario. Instead of using two streams, though, we employ a single FTP agent transmitting data on top of the standard TCP agent implemented by ns-2 from node 0 to node 1. As discussed before, this is not a completely unidirectional communication, since there is a small flow of ACKs from node 1 back to node 0.

Table 5.10 shows a summary of the obtained results in terms of aggregated throughput (*i.e.*, considering both the data flow and the ACK flow). The intervals presented on the table correspond to the 95% confidence intervals. For each route selection method, the simulation was repeated 30 times. In all cases, network coding was disabled. It is possible to see that IAR resulted in a slightly higher throughput (2.5% better than ML, the second best in this scenario), in comparison to the other route selection methods. This is somewhat expected, due to the bidirectional nature of TCP streams. Naturally, since the ACK flow is of a much lower rate than the data flow, the gains are very limited. In any case, the usage of less interfering paths by IAR still resulted in better performance in comparison to the more traditional approach.

5.1.4.2 Bidirectional TCP Streams

As a second experiment regarding TCP streams, we use a slightly modified version of the previous scenario. The only difference is the addition of a second TCP stream, from node 1 to node 0, such that there are both data and ACKs flowing in both directions. In other words, in this scenario we consider the case of a bidirectional TCP stream.

As shown in Table 5.11, the results obtained by each route selection method are very close, with the exception from Hop Count, which, as expected, performed poorly. As with

Table 5.11: Comparison between IAR, ETX, ML, and Hop Count in terms of aggregated throughput in the Pro-Interference Scenario with a bidirectional TCP stream between nodes 0 and 1. These results correspond to simulations without network coding.

IAR	**ETX**	**ML**	**Hop Count**
136.40 ± 3.42	130.27 ± 3.08	132.05 ± 3.15	0.01 ± 0.005

Table 5.12: Comparison between IAR, ETX, ML, and Hop Count in terms of lost TCP segments in the Pro-Interference Scenario with a bidirectional TCP stream between nodes 0 and 1. These results correspond to simulations without network coding. The percentage values are computed with respect to the total number of segments generated by TCP.

Reason	**IAR**	**ETX**	**ML**	**Hop Count**
Loop	0.002%	0%	0%	0%
Excess of Retries	3.0%	2.9%	3.0%	76.8%
Buffer Overflow	0%	0%	0%	0%

the unidirectional case, there is a slightly advantage for IAR, but not as expressive as in the case with CBR flows (3.3% with respect to ML).

The explanation for the lower gains obtained by IAR in the TCP case actually involves two different phenomena. Table 5.12 provides information regarding the reasons for lost TCP segments during the simulations in this scenario. While IAR was the only route selection method to result in losses due to routing loop, the percentage is very small. In terms of losses caused by excessive retries by the MAC layer, except for Hop Count, all methods resulted in similar figures. What is interesting, though, is the lack of losses due to buffer overflow. The absence of such losses indicates that TCP was never able to saturate the capacity of the paths, thus underusing the available bandwidth. That is a consequence of the well-known deficiency of TCP in wireless networks, in which losses due to medium related phenomena are confused with losses due to congestion, leading TCP to wrongly decrease its transmission rate. In any case, this data suggests that the aggregated throughput obtained by all route selection methods could be higher had TCP been able to better use the available resources.

Still, it is somewhat surprising the aggregated throughput obtained by ETX and ML with the TCP streams, given that their results with CBR flows were much lower. The reason for the better results obtained with TCP is explained by the graph in Figure 5.23. The graph shows, for all route selection methods in both the CBR and TCP scenarios, the percentage of lost frames (with respect to the total number of MAC layer transmission attempts) divided in two causes: collisions and decode failure. With the exception of Hop

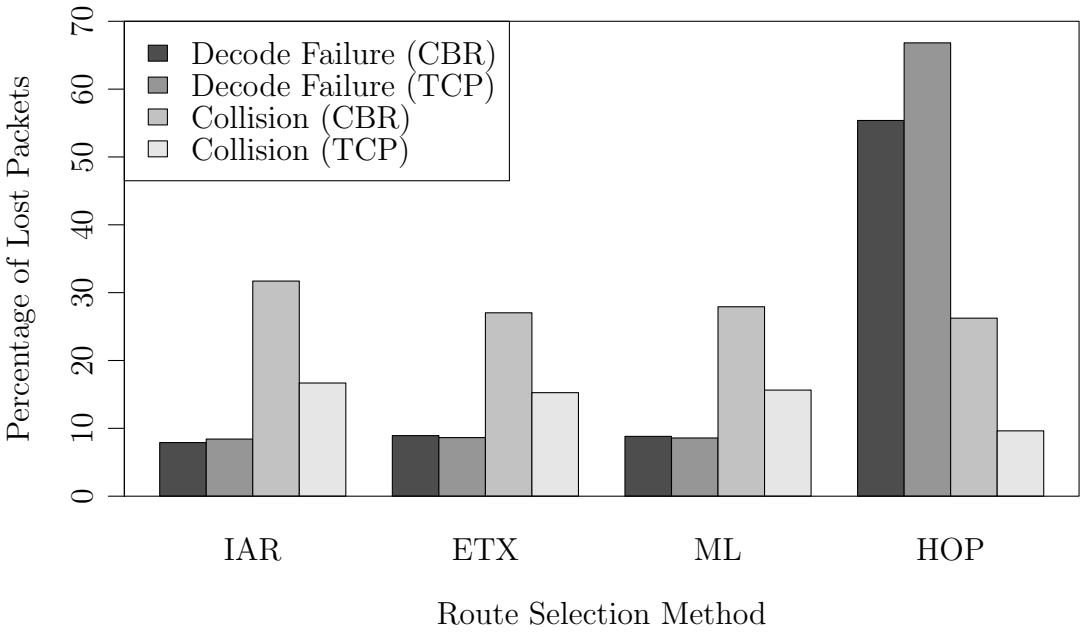


Figure 5.23: Comparison between TCP streams and CBR flows in the Pro-Interference Scenario in terms of the reasons for frame losses. The graph show percentage values computed with respect to the total number of MAC layer transmission attempts. Frame loss reasons are clustered in two groups: collision and decode failure (due to any other reason). All results represent data from simulations without network coding.

Count (which has a much lower sample size with TCP), all methods presented very similar loss percentages due to decode failure in the CBR and TCP cases. This is expected, since the path set choices for a given route selection method should not be much different whether TCP streams or CBR flows are used (although they might affect link quality estimates differently). On the other hand, the percentage of losses due to collision was much higher (roughly doubled) in the CBR case for all methods. This information suggests a much higher competition for the wireless medium, which is likely caused by a higher rate of packet generation in the CBR case.

These results indicate that, while TCP was unable to reach the maximum possible throughput (because of the lack of discards due to buffer overflow), its resultant aggregated throughput was actually better than with CBR flows because it did not cause as many losses due to collision. Figure 5.24 provides further evidence supporting this claim. The graph shows the behavior of the aggregated throughput as a function of the transmission rate of each direction of the CBR flow (*i.e.*, the aggregated transmission rate is the

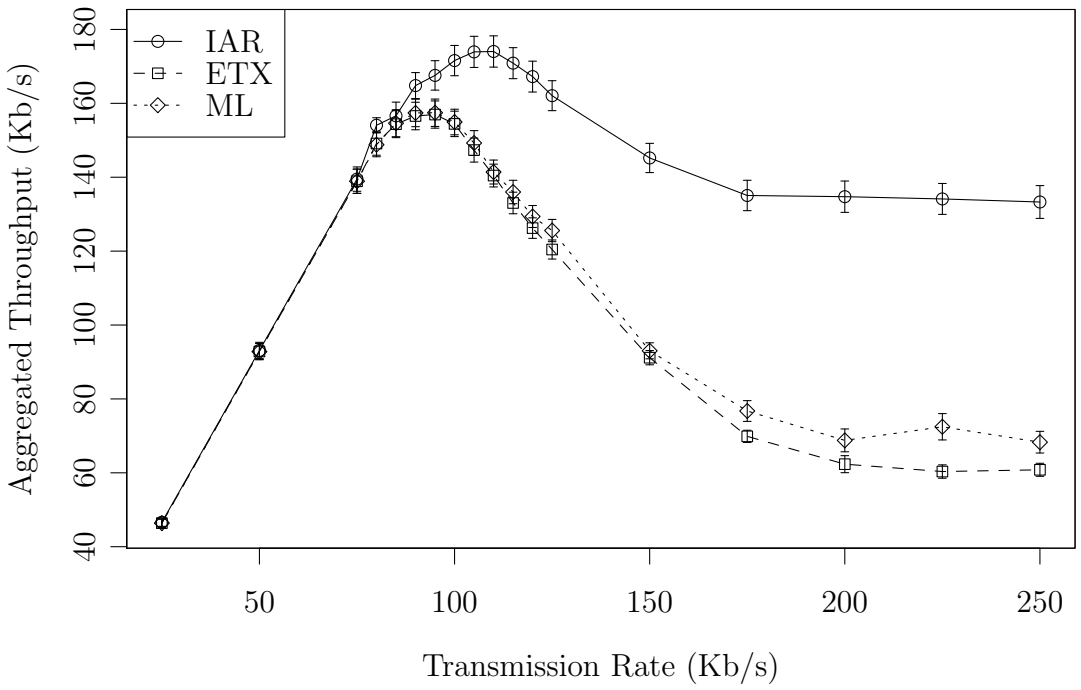
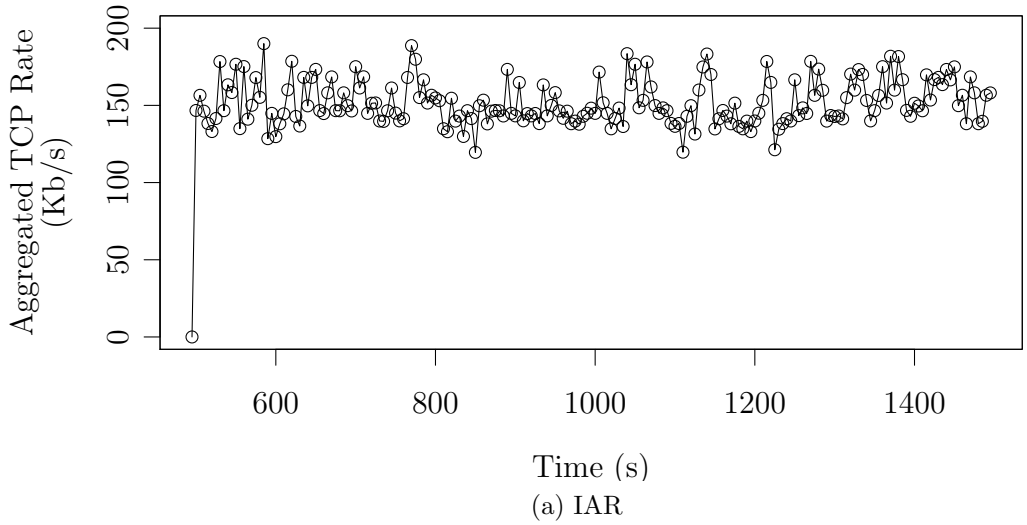


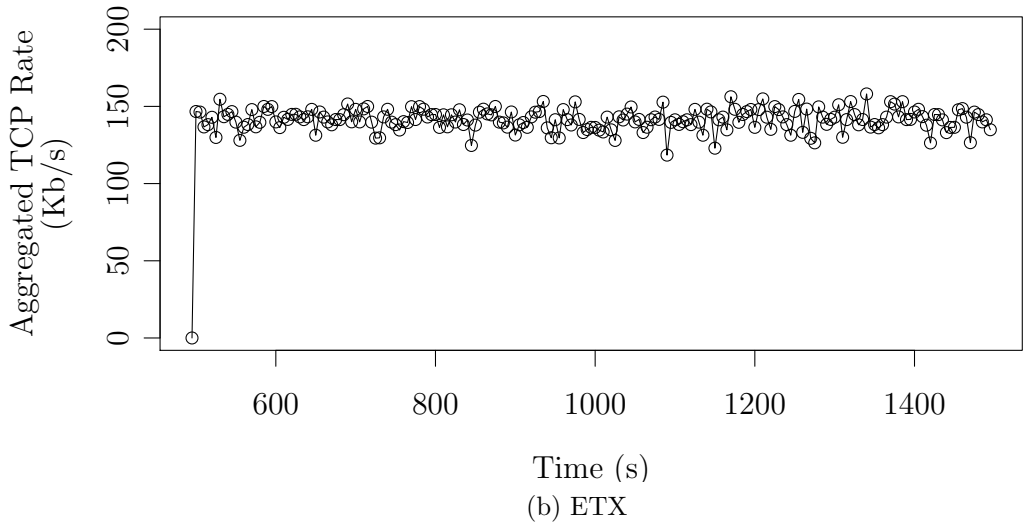
Figure 5.24: Aggregated throughput as a function of the transmission rate for each CBR flow in the Pro-Interference Scenario for each route selection method. All results are for simulations without network coding.

double of the value of the x-coordinate). Each point of the graph is the average of 20 repetitions with different seeds. The graph shows that ETX and ML reach the maximum aggregated throughput in this scenario when each direction transmits packets at a rate around 95 Kb/s. After that, further increasing the transmission rate rapidly decreases the aggregated throughput due to more collisions until approximately 200 Kb/s, at which point the results do not change much. IAR follows a similar trend, but due to the different path set selection profile, its maximum throughput is achieved at a higher transmission rate (at approximately 110 Kb/s). The highest throughput achieved by IAR is also higher than with any other method (10.5% more than the maximum obtained by ML, the second highest). Another interesting characteristic of IAR is that its aggregated throughput drops much less than the obtained by the other methods after reaching its maximum. That is also a byproduct of its selection of less interfering paths, and it explains the large gains found on the previous CBR simulations.

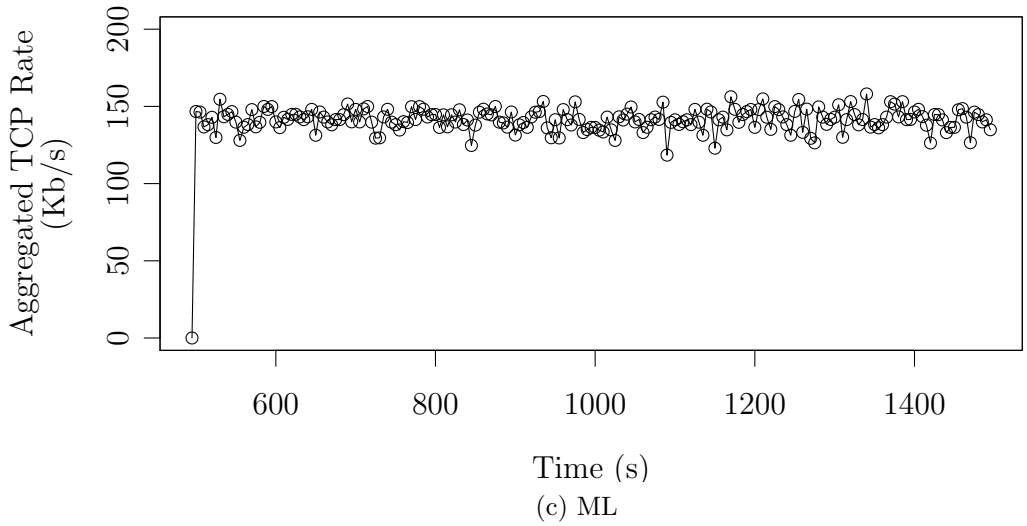
For comparison, Figure 5.25 shows the evolution of the aggregated transmission rates for both TCP streams during simulations with a specific seed for IAR, ETX and ML. Each point in the graph represents the average of the previous 5 seconds. As shown



(a) IAR



(b) ETX



(c) ML

Figure 5.25: Aggregated load generated by TCP in the Pro-Interference Scenario for each route selection method. All results are for simulations without network coding.

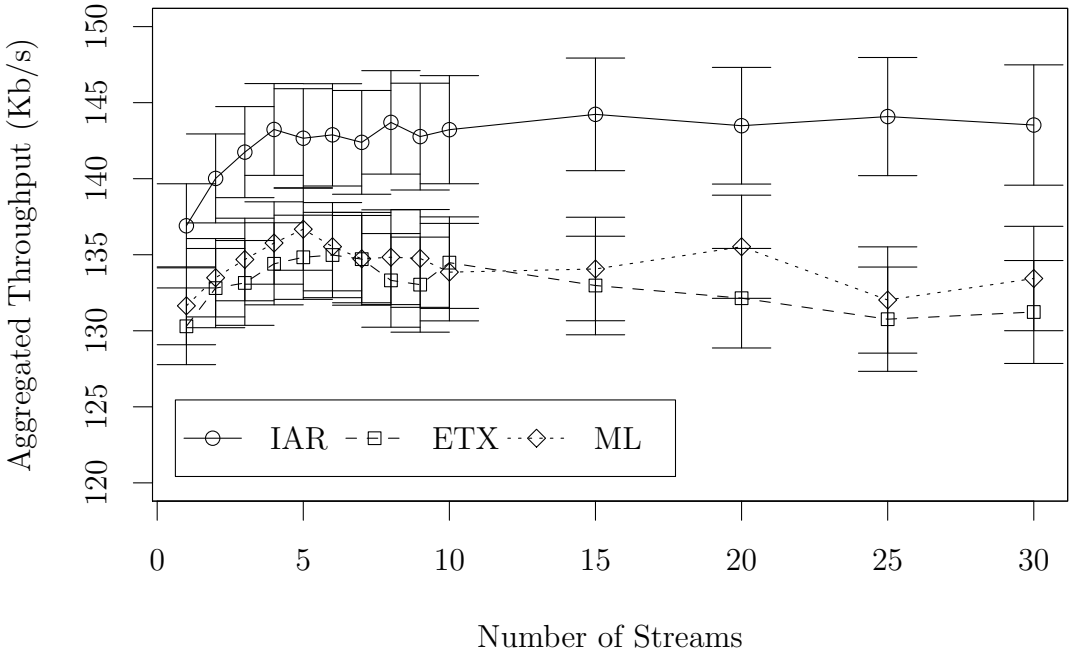


Figure 5.26: Aggregated throughput as a function of the number of TCP streams in each direction in the Pro-Interference Scenario for IAR, ETX, and ML. All results are for simulations without network coding.

by the graphs, for all three route selection methods, the average transmission rate is kept near 150 Kb/s throughout the simulation (although IAR seems to result in higher variation, reaching slightly higher values), which, given the symmetry of the scenario, suggests a rate of approximately 75 Kb/s in each direction. According to the information provided by the graph in Figure 5.24, for CBR flows, that transmission rate resulted in aggregated throughputs of 139.49 Kb/s, 138.90 Kb/s and 138.92 Kb/s for IAR, ETX, and ML, respectively. Those values are 19.85%, 11.50%, and 11.77% lower than the respective maximum throughputs obtained by each method. Hence, these data further support the hypothesis that TCP is unable to use the available bandwidth for any of the route selection methods.

One way to mitigate this deficiency by TCP is to use multiple streams in each direction. This way, the random losses caused by the wireless medium are going to be spread among the multiple flows, limiting the scope of the decrease in the window size, *i.e.*, each loss due to wireless phenomena will cause only its respective stream to react negatively, allowing the other streams to keep increasing their windows. With that in mind, we repeated the experiment with bidirectional TCP streams varying the number of streams

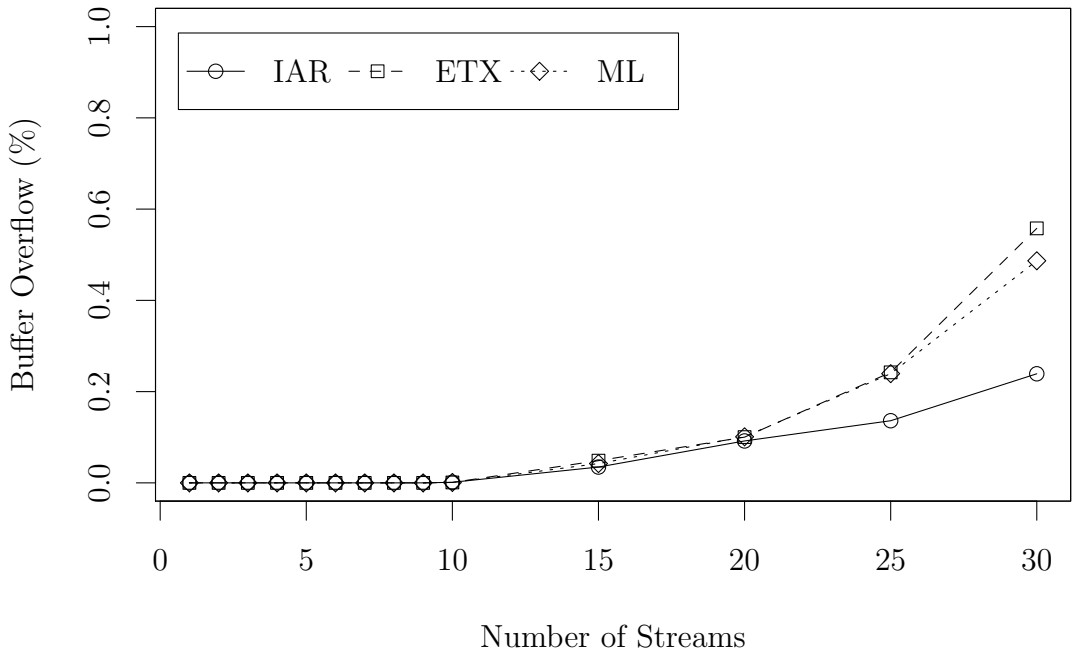


Figure 5.27: Percentage of TCP segments dropped due to buffer overflow as a function of the number of TCP streams in each direction in the Pro-Interference Scenario for IAR, ETX, and ML. All results are for simulations without network coding.

going in each direction to evaluate if that strategy is indeed able achieve throughput values closer to the maximums showed in Figure 5.24.

Figure 5.26 shows the average aggregated throughput as a function of the number of TCP streams in each direction for IAR, ETX, and ML. Each point in the graph represents the average for 30 executions of the respective simulation with different seeds. Once again, Hop Count was left out the graph to improve legibility, due to its result being much worse than the others. By increasing the number of streams in each direction from one to five, there is a slight increase in aggregated throughput for all route selection methods (4.20%, 3.48%, and 3.83% for IAR, ETX, and ML, respectively). The graph suggests a slight increase trend for up to five streams, although, considering the error bars, it is not possible to definitively make such statement. Beyond five streams, the situation seems to stabilize, in the sense that adding new streams do not further increase throughput. Notice that the gap in performance between IAR and the other two methods also increases from one to two streams (from 3.00% to 5.80% for ETX, respectively).

Figure 5.27 shows the percentage of TCP segments dropped (with respect to the total number of segments generated) due to buffer overflow as a function of the number of TCP

streams in each directions. As the graph shows, only with more than 10 streams in each direction, TCP starts to cause a small fraction of the segments to be lost due to buffer overflow, suggesting an approach to the saturation point of the paths.

5.1.5 Impact of Imprecise Link Quality Estimates

The last aspect analyzed through simulations in this thesis is the impact of the imprecision in link quality estimates over the performance obtained by IAR and ICAR. This topic has already been briefly discussed in this evaluation in Section 5.1.1.3, where it was shown that the usage of the collision probability adjustment technique could improve performance in high variability scenarios. However, as discussed in Chapter 4, the technique is not perfect, and as such, IAR and ICAR are still susceptible to variations and imprecisions in the link quality estimates which, in turn, may lead both route selection methods to make bad decisions.

In this section, we seek to quantify (at least, within the available scenarios) how much performance is lost by IAR and ICAR due to these imprecisions. To this end, the following methodology is employed. For each of the basic scenarios and the scenarios with symmetric CBR flows in the random topologies, we run simulations during 500 seconds, until routing information converges. At that point, we take a snapshot of the current information a certain node has of the network links' quality (we arbitrarily choose node 0 in all scenarios). This snapshot is used as the input for the algorithms of IAR and ICAR along with the flows for the specific scenario. The output of that single execution of the algorithms is then stored and used as a static routing table for various repetitions of the simulation with different seeds. The rationale behind this procedure is that the information available to the routing protocol at that point (immediately before the beginning of the flows) is the most reliable. Therefore, it represents the ideal conditions for both IAR and ICAR. In the remaining of this section, to differentiate IAR and ICAR from these static versions, they will be referred to as SIAR and SICAR. Notice that SIAR and SICAR are not actually feasible in practice, since a protocol must be able to handle dynamic network conditions. But they serve as a proper baseline to evaluate how much IAR and ICAR could be further improved with the development of better techniques to solve imprecisions in link quality estimates.

Table 5.13 shows the routes found using the methodology described above for both SIAR and SICAR. Except for the Pro-Coding scenario, SIAR and SICAR have chosen the exact same routes. That happens because, of the five scenarios in the table, the Pro-

Table 5.13: Static routes chosen by SIAR and SICAR for each scenario.

Pro-Interference	
SIAR	$\left\{\begin{array}{c} 0 \to 7 \to 8 \to 9 \to 10 \to 11 \to 1 \\ 1 \to 6 \to 5 \to 4 \to 3 \to 2 \to 0 \end{array}\right\}$
SICAR	$\left\{\begin{array}{c} 0 \to 7 \to 8 \to 9 \to 10 \to 11 \to 1 \\ 1 \to 6 \to 5 \to 4 \to 3 \to 2 \to 0 \end{array}\right\}$
Pro-Coding	
SIAR	$\left\{\begin{array}{c} 0 \to 2 \to 4 \\ 9 \to 12 \to 11 \to 10 \to 5 \\ 13 \to 15 \to 14 \end{array}\right\}$
SICAR	$\left\{\begin{array}{c} 0 \to 1 \to 2 \to 3 \to 4 \\ 9 \to 12 \to 2 \to 10 \to 5 \\ 13 \to 15 \to 14 \end{array}\right\}$
Grid	
SIAR	$\left\{\begin{array}{c} 0 \to 6 \to 12 \to 18 \to 24 \\ 23 \to 17 \to 11 \to 5 \end{array}\right\}$
SICAR	$\left\{\begin{array}{c} 0 \to 6 \to 12 \to 18 \to 24 \\ 23 \to 17 \to 11 \to 5 \end{array}\right\}$
Random Topology I	
SIAR	$\left\{\begin{array}{c} 12 \to 7 \to 9 \to 14 \to 23 \\ 13 \to 24 \to 15 \to 27 \to 29 \to 8 \to 5 \end{array}\right\}$
SICAR	$\left\{\begin{array}{c} 12 \to 7 \to 9 \to 14 \to 23 \\ 13 \to 24 \to 15 \to 27 \to 29 \to 8 \to 5 \end{array}\right\}$
Random Topology II	
SIAR	$\left\{\begin{array}{c} 24 \to 19 \to 6 \to 29 \to 11 \to 14 \to 25 \to 9 \\ 9 \to 10 \to 16 \to 26 \to 12 \to 15 \to 2 \to 8 \to 21 \to 24 \end{array}\right\}$
SICAR	$\left\{\begin{array}{c} 24 \to 19 \to 6 \to 29 \to 11 \to 14 \to 25 \to 9 \\ 9 \to 10 \to 16 \to 26 \to 12 \to 15 \to 2 \to 8 \to 21 \to 24 \end{array}\right\}$

Coding is the only one where coding can bring gains over simply separating flows to avoid interference.

As expected, in all scenarios, SIAR and SICAR improved the results obtained by IAR and ICAR, respectively. Figure 5.28 shows the percentage improvements obtained by SIAR (with respect to the results by IAR) in the Pro-Interference, Grid, Random Topology I and Random Topology II scenarios. In all cases, network coding was disabled. The lowest gain was of 12.3% and happened in the Random Topology I, while the largest gain was from the Pro-Interference Scenario at 30.8%. The difference between IAR and SIAR is lower in the scenarios of the two random topologies because, in those cases, separating the flows to avoid interference is a much better option than any other alternative path set. Although the route selection by IAR oscillated on those scenarios due to errors and variation in the link quality estimates, the most important characteristic was maintained

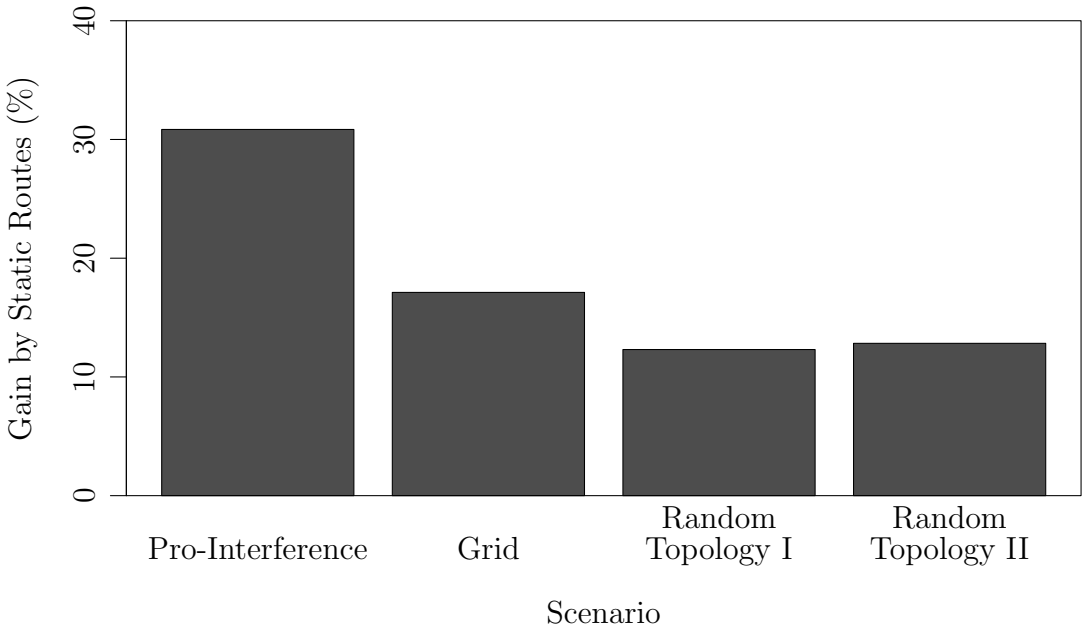


Figure 5.28: Gains obtained by SIAR with respect to the results obtained by IAR in the same scenarios. The reported values represent the percentage of improvement obtained by SIAR in terms of aggregated throughput. All values are based on simulations without network coding.

most of the time in its path selections: choosing disjoint paths to reduce interference.

The same does not apply to the Grid Scenario. In this scenario, there is no way to improve performance by distancing flows. Therefore, the best paths are very dependent on the individual link qualities and, as such, relatively small variations in the measurements may result in a link being replaced, leading to sub-optimal choices. Moreover, the propagation parameters of these scenarios were specially chosen as to result in high variation.

Although the Pro-Interference Scenario is similar to the scenarios with random topologies (in the sense that separating flows through the two disjoint paths results in much better performance than keeping them together), it actually presents an extra challenge due to the symmetry of the topology. Given this symmetry, both correspondent paths in each side of the topology have exactly the same quality. Nevertheless, due to the probabilistic nature of link quality measurements, the estimates vary and, therefore, the best path (as evaluated by the routing protocol) tends to oscillate. For IAR, while most of the time each flow is going to be routed by different sides of the topology, this oscillation

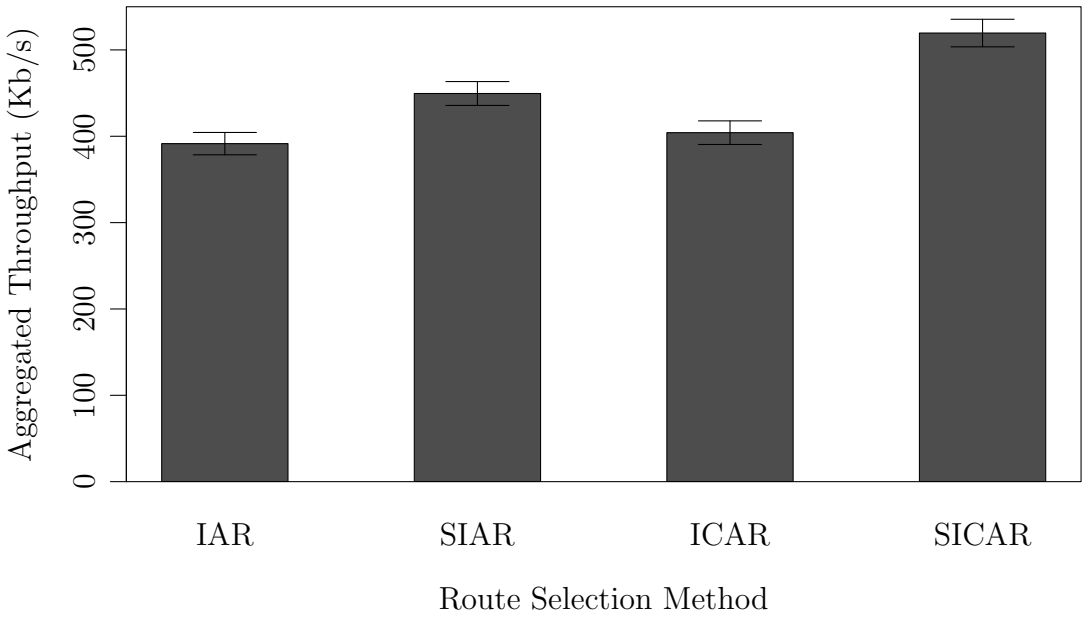


Figure 5.29: Comparison between IAR, ICAR, SIAR, and SICAR in the Pro-Coding Scenario with network coding enabled.

might result in frequent route flapping (*i.e.*, each flow is rerouted to the opposite of its current side). Since IAR uses a distributed route computation algorithm, once a flow is rerouted to the other side, there can be a delay before the other flow is rerouted as well, resulting in both flows taking interfering paths. SIAR does not change paths throughout the simulation. Thus it can achieve a much higher throughput.

Figure 5.29 shows the results obtained by SIAR and SICAR in the Pro-Coding Scenario, along with the respective results by IAR and ICAR. Those results correspond to simulations with network coding. In this scenario, both IAR and ICAR were improved by their respective static versions. What is more interesting, though, is the fact that the improvement of SICAR with respect to ICAR is considerably higher than the improvement of SIAR with respect to IAR (28.5% versus 14.8%). That result shows that ICAR's performance is being more affected by the imprecisions in link quality estimates than IAR's. One reason for such behavior is the fact that ICAR's model for evaluating path sets is more complex than its counterpart in IAR. Since ICAR considers network coding, path sets that are considered to be much worse than the optimal by IAR may be evaluated as being much more similar. That increases the probability that small changes in links' qualities result in sub-optimal path selections.

Figure 5.30: Node distribution in the real testbed representation of the Pro-Interference Scenario.

Another issue that makes ICAR more sensitive to link quality variations than IAR is the sub-routine of the candidate generation heuristic for generating candidate path sets with coding opportunities. That sub-routine is highly dependent on link quality estimates, since after choosing a node a as a coding point, it reconnects the paths using the shortest paths (in terms of ETX) from the source and destination nodes to a. By inspecting the logs from the simulations, we could notice a number of instances in which ICAR would choose a sub-optimal path set not because the optimal one was wrongly evaluated as being worse, but because it was actually never generated by the sub-routine due to imprecisions in link quality estimates.

5.2 Real Testbed

While simulations provide flexibility of scenarios and repeatability of conditions for the experiments, they may not fully capture all the aspects involved in a system as complex as a multihop wireless network. Moreover, the feasibility of implementation and usage of a solution in a simulated environment does not necessarily imply the same feasibility in a real network. For that reason, in this thesis we also provide experimental performance results based on a real testbed as a means to validate our simulation results and as a proof of concept of the viability of using IAR and ICAR and a real network composed of nodes with severe processing and storage restrictions.

Due to the difficulties of assembling and maintaining a real testbed in a controlled environment that can provide a reasonable degree of repeatability for the experiments, the results presented in this section are not as extensive as the ones based on simulations.

Nevertheless, we believe we have achieved the goals of validation and proof of concept by reproducing one of the topologies used in the simulation environments with real wireless routers. With this topology, we attempted to recreate the Pro-Interference Scenario in the real testbed.

Figure 5.30 shows the distribution of the routers during one of the experiments. Twelve TP-Link TL-WR740N wireless routers were placed in one of the laboratories of the Telecommunications Engineering Department of Universidade Federal Fluminense, disposed in such a way that two logical "lines" connected the two extreme nodes, a placement similar to the one shown by Figure 5.2. The difference, however, is the distance between each node. Given the dimensions of the room, nodes had to be placed closer to each other. For example, each two consecutive nodes of a line were spaced by 0.95 meters from each other, instead of the 10 meters used for the simulations. Even using the lowest transmission power available at the routers (0 dB), the smaller dimensions caused all nodes to be able to communicate directly. Since the resultant topology did not meet the requirements of the Pro-Interference Scenario, special boxes were build of paperboard covered in aluminium foil to accommodate each router, as shown in Figure 5.30. By placing the routers within the boxes, the necessary attenuation for reaching a topology similar to the one used for simulations was achieved.

In order to obtain more information from the experiments, a passive sniffer was placed in the center of the topology, capturing all packets transmitted within the BSSID (Basic Service Set Identification) of the test network. The TP-Link TL-WR740N operates in the 2.4 GHz band and, while it was impossible to completely isolate the test network from interference from other networks, we chose the least occupied channel at the time for the experiments (in terms of received signal strength). Each router is equipped with an Atheros IEEE 802.11n compliant wireless card, a 400 MHz CPU, 4 MB of non-volatile memory (flash), and 32 MB of RAM. The original firmware was replaced with a custom image of the OpenWRT Linux Distribution version 12.09-rc2 [70].

The procedure adopted for the experiments performed in the real testbed was very similar to the ones employed in the simulated environment. Each experiment consisted of generating CBR flows on the top of the UDP transport protocol. Both flows, sourced at each of the extreme nodes of the topology, generated packets at a rate of 200 Kb/s. An experiment always begins with 500 seconds of silence, so that the routing information can converge and the best routes are found. This initial interval is followed by the start of the flows, that run for 500 seconds. All routers had their transmission rates fixed at

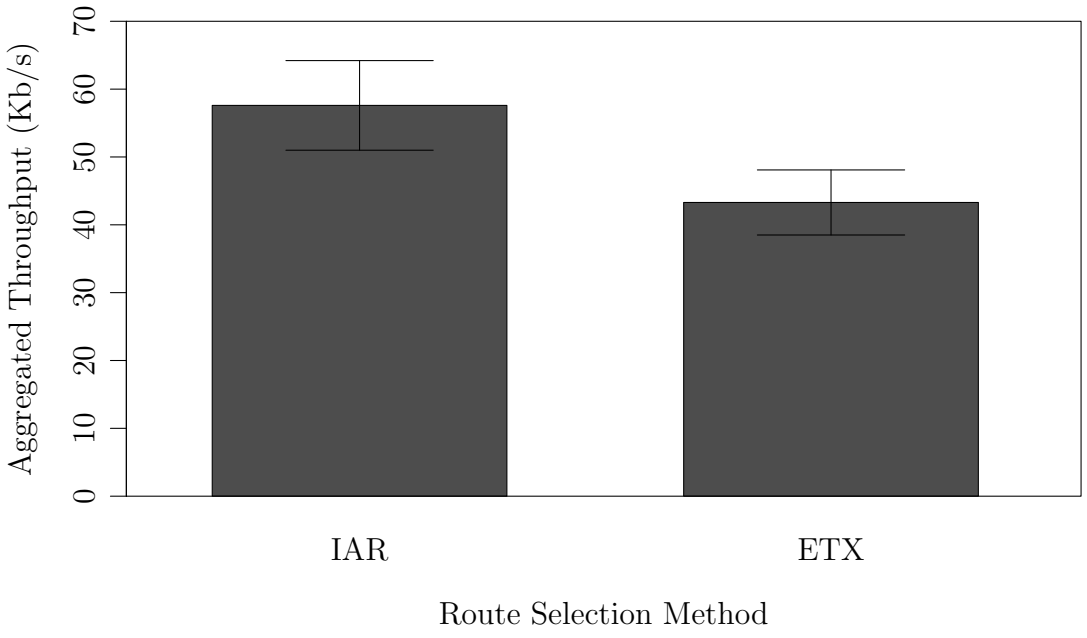


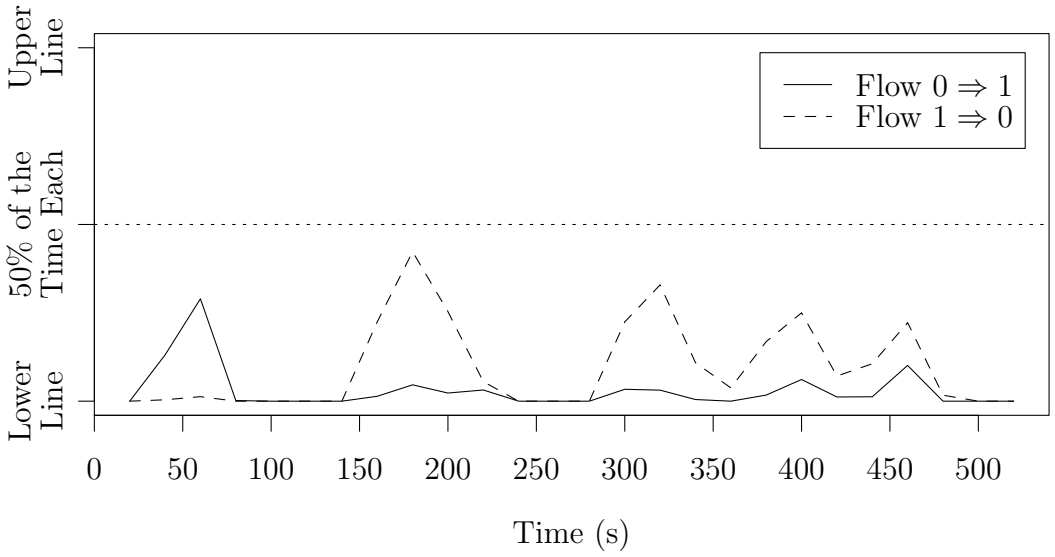
Figure 5.31: Comparison between IAR and ETX in a real testbed representation of the Pro-Interference Scenario without network coding.

1 Mb/s.

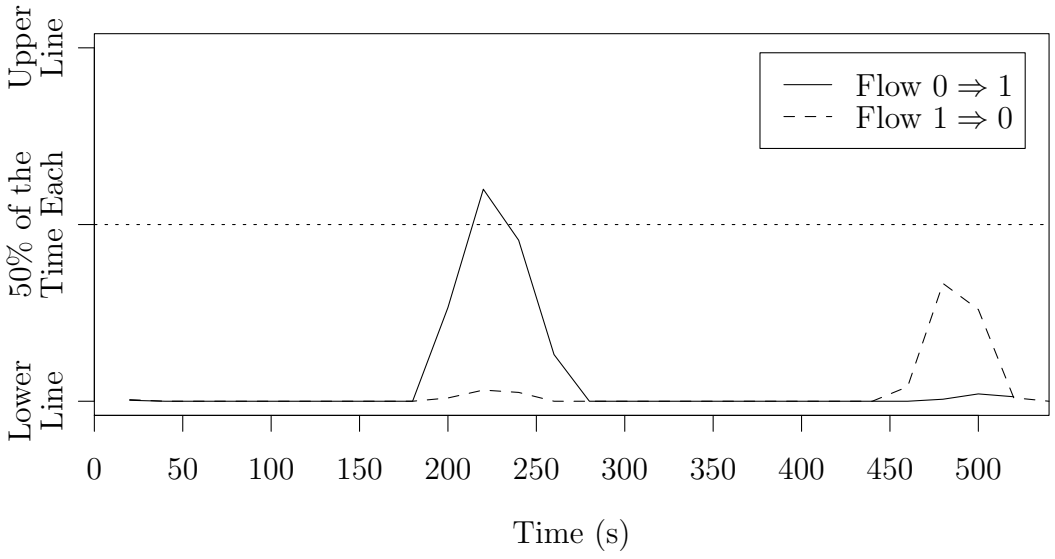
Since the ETX metric obtained the best results (or very close) among the traditional route selection methods during the simulations, we decided to limit the experiments to a comparison between IAR and ETX. Since the Pro-Interference Scenario does not present a clear alternative route solution based on network coding, all experiments were executed without it — thus the exclusion of ICAR from these experiments.

Figure 5.31 shows aggregated throughput results obtained for the Pro-Interference Scenario in the real testbed. The results are shown in the form of an average of throughput samples taken at intervals of 1 second. The error bars represent the 95% confidence intervals with respect to the mean. As in the case of the simulated environment, IAR was able to outperform ETX. While the percentage difference was not as large as in the simulations (IAR was 34.8% better than ETX in the real testbed), it still represents an improvement beyond the bounds provided by the confidence intervals.

The graphs in Figure 5.32 provide more information on this experiments. The two graphs show, for IAR (top) and ETX (bottom), the utilization of network links — for each flow — separated in two groups: links belonging to the “upper line” of the topology



(a) IAR



(b) ETX

Figure 5.32: Comparison between the route choices made by IAR and ETX in the real testbed representation of the Pro-Interference Scenario. Each graph represents the fraction of links used on the upper side of the topology in comparison the fraction of links used in the lower side, as a function of time. For a given point of the graph, if it is closer to the label "Upper Line" in the y-axis, for example, then, during that sample, more links of the upper line of the topology were used for the respective flow.A point in the middle of the y-axis indicates that 50% of the links used during the sample were from the lower line, while the other 50% were from the upper line.

and those from the "lower line". Each point used to plot those graphs represent a window of 20 seconds during the respective experiments. During such a window, we isolated the transmissions captured by the sniffer according to the flow to which the respective packets belong. Then, for each captured transmission, the used link was determined and that information was used to classify the transmission between "upper line" and "lower line". Finally, for each flow, we computed the percentage of transmissions in each line. The y-coordinate of a point in the graph, thus, represents this proportion for that specific sample of the respective flow. The closer the point is from the "Upper Line" label (with respect to its y-coordinate value), the higher is the proportion of transmissions for that flow that used links of the upper line of the topology. Likewise, the closer the point is from the "lower line", the higher is the proportion of transmissions in the lower line of the topology. A point in the middle represents a window in which the same amount of transmissions were performed on both lines.

In an ideal scenario where both lines are equally good (such as the one used in the simulated environment), the best possible path selection routes each flow in a different line. Even if the selection varies with time (*i.e.*, the line assigned to each flow changes), the route selection will still be optimal as long as two flows do not share one of the lines at any time. In terms of the graphs shown in Figure 5.32, that would be represented by the curves for each flow always staying at different ends of the y-axis (although they might alternate throughout the time).

Neither Figure 5.32a nor Figure 5.32b show that perfect behavior. In fact, both graphs show a strong bias towards choosing paths comprised of links from the lower line of the topology. The reason for that is the imperfection of the Pro-Interference Scenario built on the real testbed with respect to the symmetry of the topology. Due to various sources of asymmetry, such as possible subtle differences in the assembly of the boxes, different objects nearby the routers and small errors in the placement of nodes, the resultant topology of the real testbed presented asymmetrical lines in terms of performance.

Nevertheless, while ETX only attempted to use the upper line at two occasions during its experiment, IAR tried the alternative route more frequently, concluding from time to time that, even if one of the paths results in worse performance, it is still better (in terms of aggregated network throughput) to separate both flows. That indicates that, even in a real scenario with complex phenomena affecting links' qualities, IAR was able to find path sets that differed from the common choices made by traditional route selection methods in order to reduce interference. As suggested by the results in Figure 5.31 those alternative

choices indeed resulted in better aggregated throughput. Moreover, IAR was able to do so in real time, even running on top of a constrained hardware.

Chapter 6

Conclusion

In this thesis we studied the problem of route selection taking into account the effects of self interference and considering the possibility of using network coding, if available for the network. Specifically, we proposed an interference-aware route selection algorithm and, as a natural extension, an interference and coding-aware route selection algorithm, called IAR and ICAR, respectively.

Both algorithms use a novel approach to the problem of route selection, based on complete view of the network's active data flows, and perform a simplified simulation of the behavior of the network with the usage of a given set of paths for the current flow demands. This simulation is capable of determining the long term aggregated network throughput resultant from the usage of the specified path set, assuming deterministic medium access scheduling rules that resemble the long term statistical characteristics of practical MAC layer protocols used in multihop wireless networks. We mathematically prove that, given the models used by our route selection algorithms, they always converge to a solution and that the aggregated throughput found by this method is always the long term average, unaffected by any possible short term transient effects.

This novel approach allowed the algorithms to tackle the problem of route selection with a global view of all flows, *i.e.*, how flows interact with each other, possibly reducing each others performances. This global view made it possible that both IAR and ICAR could be successful at choosing routes with low self interference.

Throughout this thesis, we reiterate the objective of obtaining practical route selection algorithms, which lead to the creation of multiple auxiliary heuristics, in order to keep the computational complexity of our proposal low enough so that it could be deployed in off-the-shelf equipments with low processing capabilities. This practical vision towards the implementation of the proposed route selection algorithms also resulted in multiple

practical aspects of how to implement IAR and ICAR in a real link state routing protocol being approached in Chapter 4. Specifically, this chapter discussed how to detect and propagate the information of the existent network flows for all nodes, how to reduce the initial delay until the establishment of the optimal routes, how to transmit coded packets in best possible way, and how to improve the estimates of the links' delivery probabilities by decoupling them from the collision probabilities. This last method has been proved to be especially useful (in terms of increasing network performance) in the experiments presented in Chapter 5.

This thesis also provided two practical implementations for both IAR and ICAR — one for the ns-2 simulator and another for Linux. We presented an extensive experimental evaluation of the proposals, comparing their performances with the ones provided by traditional and widely adopted routing metrics. Various different characteristics of IAR and ICAR were evaluated, including their behavior in scenarios with clear opportunities for avoiding interference — either by placing concurrent flows apart or by joining flows and exploring network coding — as well as random scenarios and scenarios with particularly high variability. The performance of IAR and ICAR in those multiple scenarios demonstrated that, as expected, they were able to find alternative path sets that resulted in less interference, increasing the aggregated network throughput.

While one of the hypothesis considered by both IAR and ICAR is that flows are always backlogged and transmitting at the highest possible rate, we also evaluated their performance in scenarios with lower transmission rates and with TCP flows. With those experiments, we demonstrate that both algorithms still can provide gains for those cases, although the well-known deficiency of TCP on lossy wireless networks reduced those gains.

We also extended the results obtained through simulations by recreating one of the topologies in a real testbed composed of simple off-the-shelf routers. The results show that the premises of our algorithms hold even in a real scenario, resulting in better aggregated throughput than the traditional metrics. Moreover, we demonstrated the viability of executing those algorithms in real time even in equipments with severe memory and processing power restrictions.

The importance of the results obtained in the real testbed are two folded. On one hand, they prove the viability of running IAR and ICAR under real circumstances. On the other hand, they prove the feasibility of the more general concept of selecting routes based on the simulation routine provided by this thesis. While IAR and ICAR have shown good results, the second consequence is arguably more important, since the same

framework used by these algorithms can be used and extended with different models incorporating new or more characteristics. One example of the power of this route selection framework is how IAR and ICAR can separate the effects of the link delivery probability into two components: the link transmission delay and the link packet loss. By treating those components separately and, later, combining them, IAR and ICAR can find a direct estimate for individual and aggregated throughput. Traditional routing metrics, by contrast, employ indirect metrics that only correlate with throughput, such as end-to-end delay and packet loss (separately).

This thesis also provided an extensive revision on the literature of routing in multihop wireless networks, including traditional routing protocols and metrics, as well as interference-aware and coding-aware proposals.

Finally, the appendices of the thesis provide some interesting results regarding the estimation of delivery probabilities for unicast links, as well as for multicast links (in the case of the transmission of a coded packet). We present non-expected experimental results that suggest that hardware limitations may cause packets to be unexpectedly dropped when network interfaces are under heavy loads. We also discuss the issue of whether the reception events of a single coded packet transmission at different receivers are independent. We found that those events are not always independent, which diverges from assumptions found commonly on the literature. Under the light of this finding, we propose a method for estimating the joint delivery probability for a coded packet addressed to multiple receivers that does not rely on any assumption regarding the independence of the receiving events.

6.1 Future Work

There are many possible future research paths following the work developed in this thesis. One of them is to explore in more depth the issue of the stability of the links' delivery probability estimates under heavy network traffic. As shown in Chapter 5, while the collision probability adjustment method was able to mitigate the effects of the collisions on the estimates, there is still considerable room for improvement.

Another issue that deserves more investigation is the design of different models for the case of network coding working with different methods for transmitting coded packets. While this thesis provides a general path for creating such models for Random Pseudo-Broadcast and Deterministic Pseudo-Broadcast, the issue of how to estimate the reception

probability component is still an open question. It is important to create such models so that a fairer comparison between Simple Broadcast and the other two methods can be carried out.

It is also possible to explore different concepts for what defines a flow for IAR and ICAR. For instance, instead of grouping all packets with the same source and destination nodes, individual flows could be created based on transport protocol or application type. With changes in the model of the path set evaluation routine, QoS requirements might be added to the route selection process. This could also be used to include load balancing capabilities in IAR and ICAR.

Still regarding flow information, one clear path for improvement of both IAR and ICAR is taking into consideration the actual rate of packet generation for UDP flows. As shown in Chapter 5, a deficiency of both proposals is the lack of consideration for this parameter. Both algorithms may deviate flows from their individual optimal paths even if other potentially interfering flows have low rates. In that case, knowing that the level of interference is low, IAR and ICAR could avoid rerouting flows, reaching better aggregated throughput. Another possible optimization in this regard would be to disregard short-lived flows. As stated in Chapter 3, IAR and ICAR implicitly assume flows to be long in terms of duration. If it was possible to differentiate flows according to their durations (or expected durations), IAR and ICAR could ignore them during route selection, assuming their short duration would not have a great effect on the performance of other flows.

Support for multiple transmission rates is also an important investigation point. Both IAR and ICAR were modeled assuming that all network nodes use the same transmission rate (which is also assumed to be the same transmission rate in which link delivery probability estimates were obtained). In general, this is not true: it is common for multiple transmission rates to be available and be selected dynamically by a rate adaptation algorithm. Notice, however, that IAR could be easily extended to include multiple rate information. In order to do so, it is simply necessary to know, for every link, the current transmission rate among with an estimate for the delivery probability. This could be done by integrating IAR with MARA [75]. In that case, MARA would be responsible for selecting the best transmission rate for each link, while also providing the necessary information for IAR, which would then compute the best routes accordingly. From IAR's point of view, the only difference would be in computing the transmission delay for each link (the time they occupy the wireless medium, on average).

For ICAR, the addition of support for multiple transmission rates is not as straight-

forward. The first issue, in this case, is how to select the transmission rate for a coded packet. While some authors suggest the simple employment of the lowest transmission rate among those selected for each intended receiver, which would guarantee that all receivers would be able to receive the transmitted packet, it has been shown that a lower rate is not always more robust than a higher one [75]. Thus, this question is still open. Moreover, for Deterministic and Random Pseudo-Broadcast, there would be the issue of how to compute the joint reception probabilities at multiple transmission rates.

Finally, in terms of performance evaluation, it would be important to explore in more depth the effects of some of the parameters used by IAR and ICAR on the performance of both route selection algorithms. In this thesis, the main focus of the performance evaluation was to demonstrate the viability of these algorithms in practice and that they are able to find paths that result in considerable gains in practical scenarios. For that reason, parameters were only evaluated in preliminary experiments so that reasonable values could be found. A deeper investigation could define how those parameters relate to the characteristics of each scenario, resulting in optimizations. Another aspect that deserves a deeper evaluation is the issue scalability. It is important to extend the evaluation scenarios to use more flows in order to analyze how the proposed algorithms behave. Furthermore, the performance evaluation could be extended to include topologies of real wireless mesh networks.

APPENDIX A – Packet Losses Under Heavy Loads

As discussed throughput this thesis, a traditional mechanism for measuring the delivery probability of a wireless link is the use of periodical probes [75, 23, 78, 12]. Nodes periodically transmit probes in broadcast (to avoid retransmissions) containing, among other fields, a sequence number. Based on this sequence number, each neighbor can detect probe losses (missing sequence numbers) in order to compute an estimate for the delivery probability, which is then sent back to the source of the probes.

It is well documented in the literature, especially on routing metrics for multihop wireless networks, the fact that, under heavier loads, the delivery probability of network links (as measured by periodical probes) tends to decrease [75, 17, 23]. Two commonly accepted explanations for this phenomenon are:

1. losses of probes due to buffer overflow; and
2. losses of probes due to an increase in the number of collisions.

The first explanation explores the fact that under heavy traffic the network congestion increases, causing the backlog on nodes' queues to increase as well, eventually leading to packet discards. Among the discarded packets, there would be a share of probes. Therefore, a lower number of probes would reach the wireless interface to actually probe the wireless medium. Since a neighbor cannot distinguish between a loss due to degradation of the wireless medium or due to buffer overflow, both cases would account for a decrease in the perceived delivery probability of the link. The second explanation is based mostly on the problem of hidden terminals, a characteristic which is specially accentuated in multihop wireless networks [4].

As discussed in this thesis, however, it is not always desirable that the estimates for the delivery probability include factors other than the link quality itself. For this reason,

in this appendix, we analyze the true causes of the decrease in delivery probability, as measured by periodical probing, under heavy traffic loads. Our analysis specifically targets IEEE 802.11 [49] based networks, since this technology is widely adopted in a wide range of applications. We present results from a number of experiments conducted in a real testbed using implementations of the standard by different vendors. Our results show that the increases in collisions and congestion are not the only causes for this phenomenon. Specifically, the presented results suggest that the simple increase in the amount of time a wireless interface spends on the transmission state decreases its capacity of receiving frames from its neighbors [77]. Incidentally, we show that this decrease in the delivery probability happens in both infra-structured and ad hoc modes of the IEEE 802.11 standard. Moreover, our results are consistent throughout all evaluated interfaces, regardless of vendor. Therefore, we believe the results reported in this appendix demonstrate an unexpected hardware limitation in commercial off-the-shelf implementations of the IEEE 802.11 standard.

A.1 Experimental Analysis

The starting point of our analysis is a very simplified scenario composed of two wireless nodes running a multihop wireless network routing protocol. On this scenario, depicted in Figure A.1, nodes a and b run `olsrd` [68], a widely adopted implementation of the OLSR [21] (Optimized Link-State Routing) protocol. In addition, node c, a passive element of the testbed, runs a sniffer capturing traffic of the network, providing a way to monitor the packets transmitted during the experiments. Due to the proximity of nodes, we assume that any collisions at nodes a or b would also affect node c with high probability. In Section A.1.6 we present experimental data that validates this hypothesis. Unless noted otherwise, the wireless interfaces from nodes a and b are configure to operate in IEEE 802.11 ad hoc mode. This basic scenario is employed for all experiments, with only small modifications in each case.

A.1.1 First Experiment: Routing Protocol

In this experiment, nodes a and b are two identical laptops with a Pentium M processor at 1.7 GHz and 512 MB of RAM, running Ubuntu (Linux Kernel version 2.6.32). Node c is also a laptop with a Core2Duo processor at 2.53 GHz and 3 GB of RAM, running Backtrack Linux 5 (Linux Kernel version 2.6.39). Nodes a and b use a D-Link DWA-

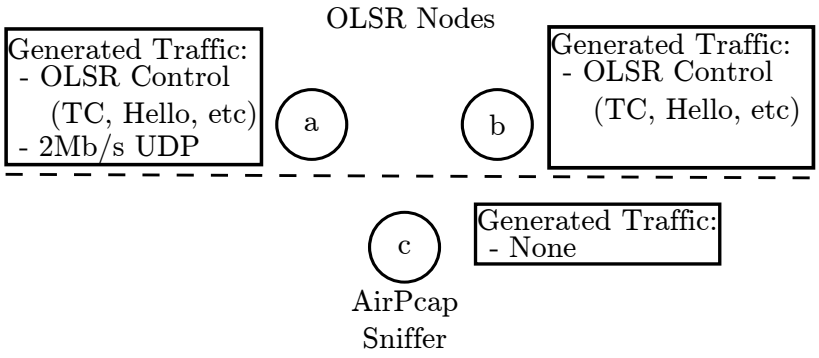


Figure A.1: Scenario used as a testbed for the first set of experiments.

110 IEEE 802.11b/g USB adapter as their wireless interfaces. This adapter is based on the RaLink RT73 chipset. Node c uses an AirPcap USB adapter, a wireless interface specifically designed to be used as a sniffer. All nodes are placed close to each other, so that the delivery probability can be as high as possible between them. The interfaces of both nodes a and b were configured to use only the basic rate of 1 Mb/s, so that it would be easier to saturate the network capacity during the tests.

This first experiment consists of switching on and off a UDP flow of 2 Mb/s (so that the link becomes saturated), from node a to node b, generated with the well-known bandwidth measurement tool `iperf`. Since the path between nodes a and b is comprised of a single hop, there is no intra-flow interference to cause the number of collisions to increase. Therefore, the only effect of the UDP flow in the probes of the routing protocol would be in terms of increasing congestion and perhaps causing probe losses due to buffer overflow. Hence, the expected behavior for this experiment would be that the delivery probability (as measured by the routing protocol) would decrease only in the direction $a \rightarrow b$ (*i.e.*, from source to destination of the UDP flow).

Figure A.2 shows a plot of the evolution of the measured delivery probability measured by OLSR (the solid line). The graph also presents the same information, but as measured by the Sniffer (node c, dashed line). During the first 120 seconds of the experiment, the UDP flow was disabled, so that OLSR could converge to the actual probability. From that point on, the UDP traffic was activated and deactivated in periods of 150 seconds. As the graph shows, the UDP traffic had little influence in the measured probability throughout the experiment. This suggests that losses due to buffer overflow do not influence (at least, not to a great extent) the measurements of the routing protocol[1].

[1]To further validate this conclusion, we repeated this experiment giving priority to OLSR traffic over any other type of traffic and the results were very similar.

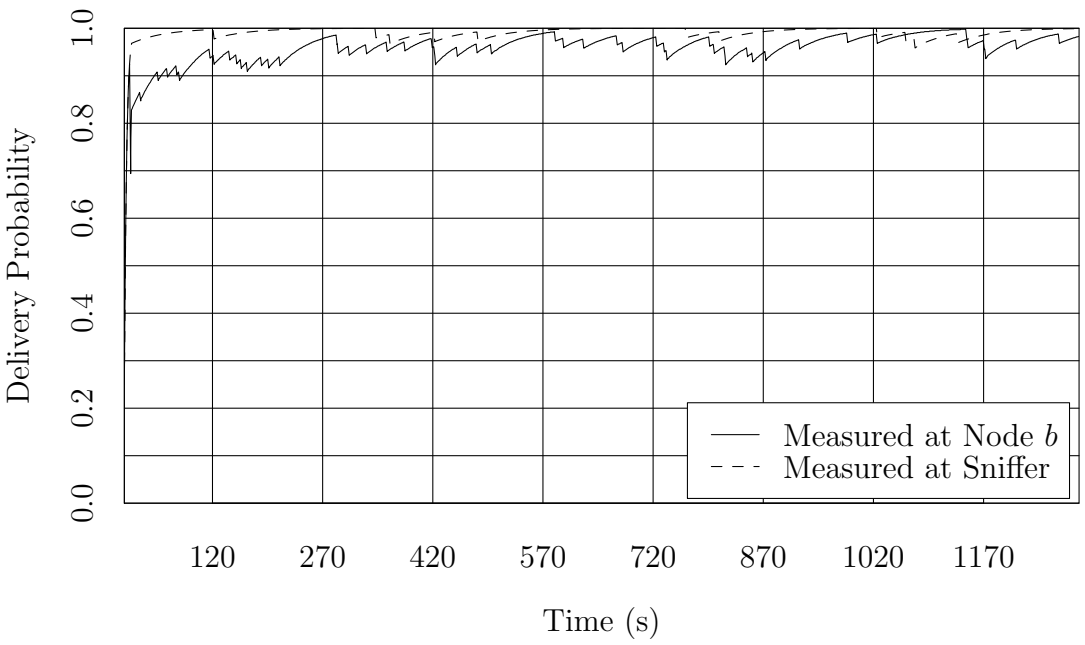


Figure A.2: Delivery probabilities reported by the sniffer and by OLSR in the direction $a \rightarrow b$.

Figure A.3 shows the same information, but in the opposite direction (*i.e.*, for the probes transmitted from node b to node a). In this case, we can clearly see an accentuated drop in the delivery probability during the periods of activity of the UDP flow (*e.g.*, from 120 to 270 seconds), followed by an increase back to the normal levels during the periods of inactivity (*e.g.*, from 270 to 420 seconds). It is interesting to notice, however, that the probability measured at the sniffer suffers a much smaller variation. Actually, the perceived variation in the graph for the reception at the sniffer is due to an increase in jitter, rather than a decrease in the delivery probability. The plotted values are based on the exponentially weighted moving average method used by OLSR that penalizes jitter. If a probe is not received within a given interval, OLSR decreases its estimate of the delivery probability. Under heavier loads, transmissions tend to present higher jitter due to contention, leading the estimated delivery probability to be decreased more frequently. This difference between the delivery probabilities indicates that node b is indeed able to transmit OLSR probes and that the corresponding frames are not being lost due to collision (as will be discussed in more depth in Section A.1.6). Due to what appears to be hardware implementation constrains, when node a has more traffic to transmit, its capacity of receiving OLSR probes diminishes.

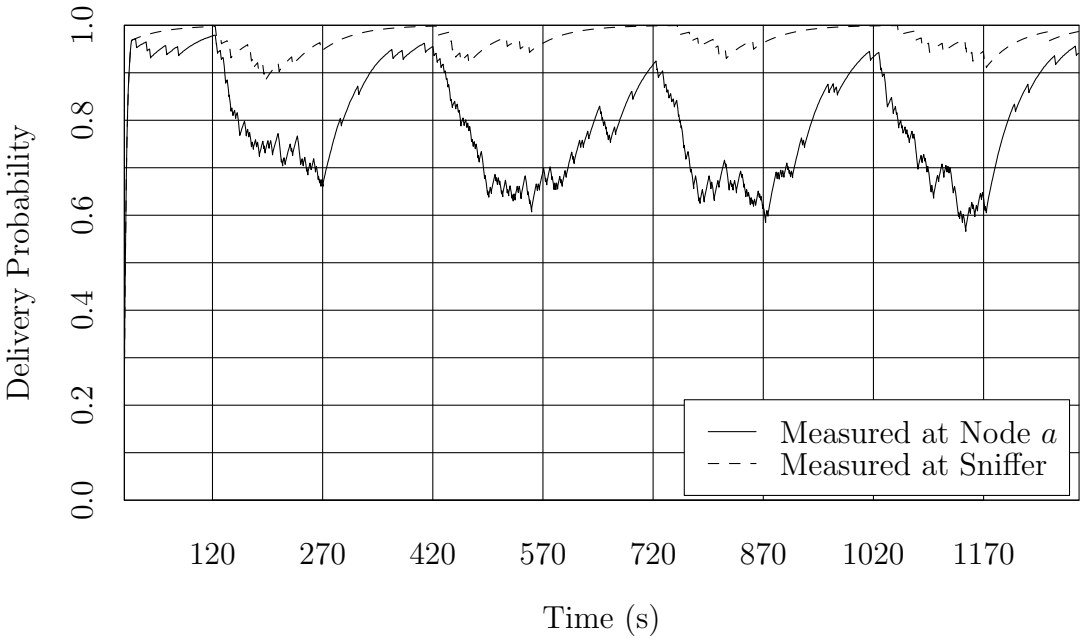


Figure A.3: Delivery probabilities reported by the sniffer and by OLSR in the direction $b \rightarrow a$.

A.1.2 Second Experiment: Generic Client-Server Application

To guarantee that these results are not an artifact of an implementation issue of `olsrd`, we performed a new experiment in the same basic scenario, with a few modifications. The OLSR protocol was replaced by a simple client/server application. The client generates a constant bit-rate traffic giving each packet a unique sequence number. The server receives these packets and log their sequence numbers. Packets are always 1500 bytes long. Node b runs an instance of the client sending broadcast packets every 500 ms. Node a runs an instance of the server, generating a log file for post-processing so that the evolution of the delivery probability with time can be computed.

At the beginning of the experiment, node a also runs an instance of the client for 600 seconds, constantly generating unicast traffic addressed to node b (*i.e.*, the interval between transmissions is 0). After 600 seconds of silence, the instance of the client at node a is restarted. This process is repeated for 6 rounds. To guarantee that the processing overhead of the client at node a does not cause the server to lose packets sent by b, the server process is executed with the highest priority available for the FIFO scheduler in Linux.

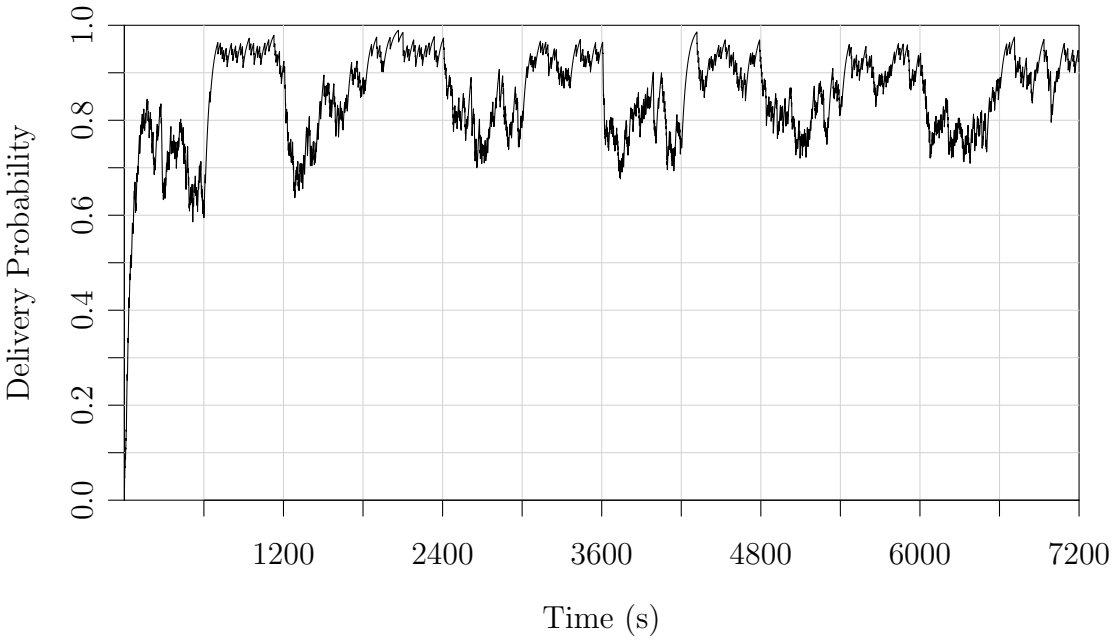


Figure A.4: Evolution of the delivery probability estimated by node a of the link $b \rightarrow a$ as a function of time in the second experiment.

Figure A.4 shows the behavior of the delivery probability of link $b \rightarrow a$ throughout the experiment. As with the first experiment, without the presence of traffic from node a to node b, the delivery probability for node a was around 90% or more. However, during the intervals in which node a produced traffic, the probability drops considerably.

Figure A.5 shows the same data, but summarized by each of the six rounds. Even considering the 95% confidence intervals (shown for each average), the probability drops at least 5 percentage points in all rounds when the client instance at node a is restarted.

A.1.3 Third Experiment: Unicast Frames

A question that arises from the first two experiments is whether this behavior is limited to broadcast frames. In other words, if the traffic transmitted by node b to node a was composed of unicast frames, would the same drop in the delivery probability of a happen as well? To answer this question, we conducted a third experiment as follows. Both nodes a and b generate unicast UDP flows addressed to each other during 300 seconds with `iperf`. The sniffer at node c logs all frames, including retransmissions. The delivery probability is then estimated as the ratio between the number of unique frames and the

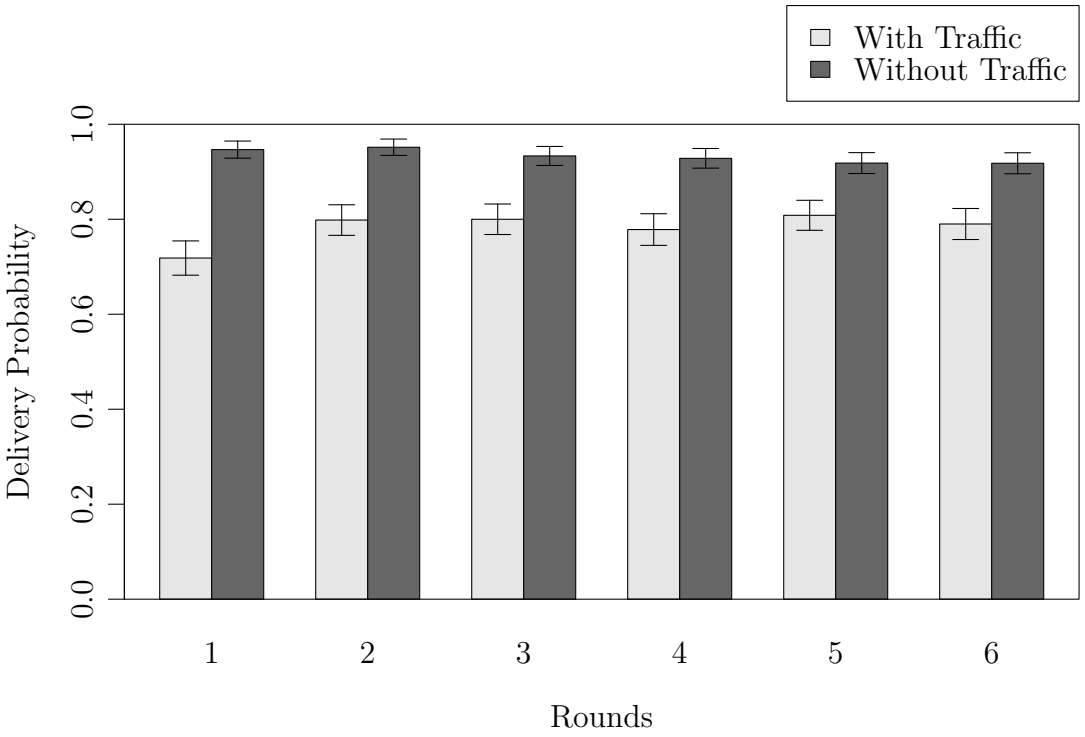


Figure A.5: Average values of the delivery probability of the link $b \rightarrow a$ for each round of the experiment, with and without concurrent traffic.

total transmissions (including retransmissions) received by the sniffer. Notice that this summarizes frames from both flows into a single average, weighted by the number of packets transmitted by each node (which was roughly equal in our experiments).

This experiment was repeated in two modes: unidirectional and bidirectional. In the bidirectional mode, both flows are started at the same time and thus compete with each other. In the unidirectional mode, first the flow from a to b is started and only after it ends, the second flow begins.

After four runs of both modes of this experiment, the data presented in Figure A.6 was obtained. We repeated this experiment for four different interfaces manufactured by different vendors: Atheros 2313 WiSoC, Intel 5100 AGN, Broadcom 5352 and Ralink RT2501USB. In all four cases the results were similar: the delivery probability drops significantly in the bidirectional mode. Notice that, according to our assumption, this drop cannot be attributed to collisions between the transmissions of the two nodes, because the frames are being correctly received by the sniffer.

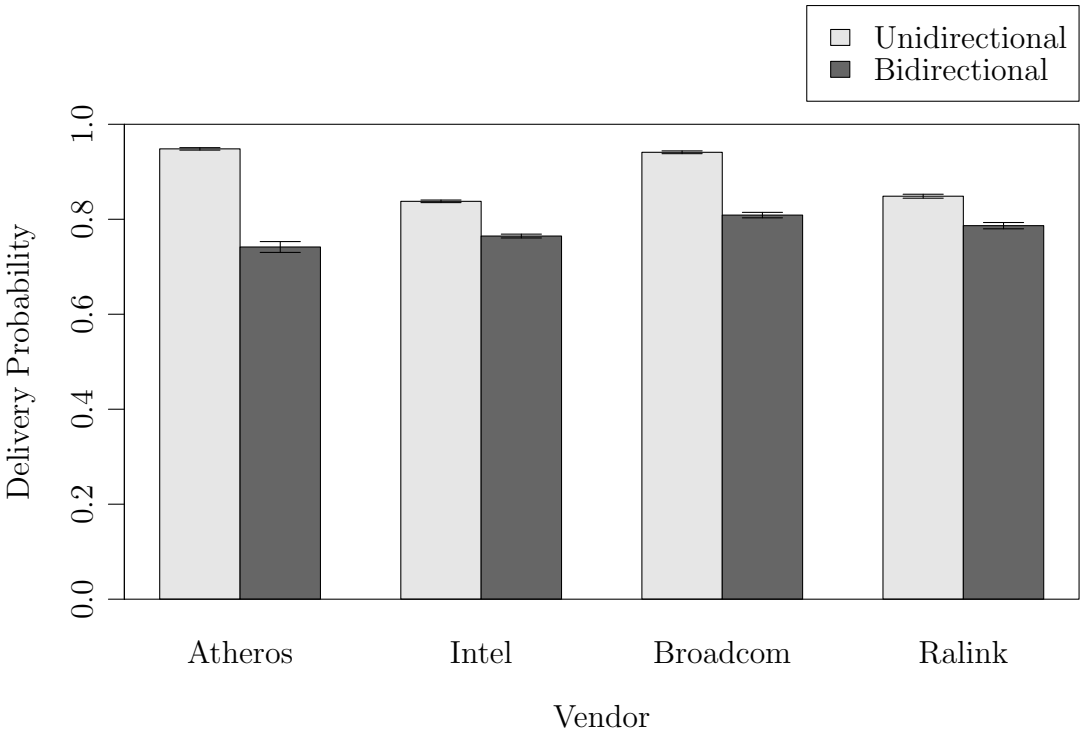


Figure A.6: Estimate of delivery probability for nodes a and b from the sniffer point of view during the third experiment.

A.1.4 Fourth Experiment: Probability Drop vs. Network Load

It is interesting to evaluate how the drop in the delivery probability relates with the transmission load of the interface. To do so, we proposed a fourth experiment. Basically, we repeated the second experiment, but varying the transmission interval for the client process at node a. This experiment lasted 3600 seconds, divided in 6 windows of 600 seconds each. In the first window, there was no instance of the client process running at node a. In the second window, the client was configured to send one packet every 22 ms. For each of the next windows, the interval between packets was decreased in 2 ms. This procedure resulted in loads of 0, 545, 600, 667, 750 and 857 Kb/s, respectively.

Figure A.7 shows the evolution of the delivery probability for the link $b \rightarrow a$ throughout the experiment. Although there are short term variations, it is possible to see a downwards trend from the first window (lowest load) to the second and from the fourth window to the end of the test (heaviest load). It is easier to see these trends in Figure A.8, which presents a summary of the average of each window along with the correspondent 95% confidence interval. Although, on average, the delivery probability slightly drops from the second window to the third and from the third to the fourth, the variation is

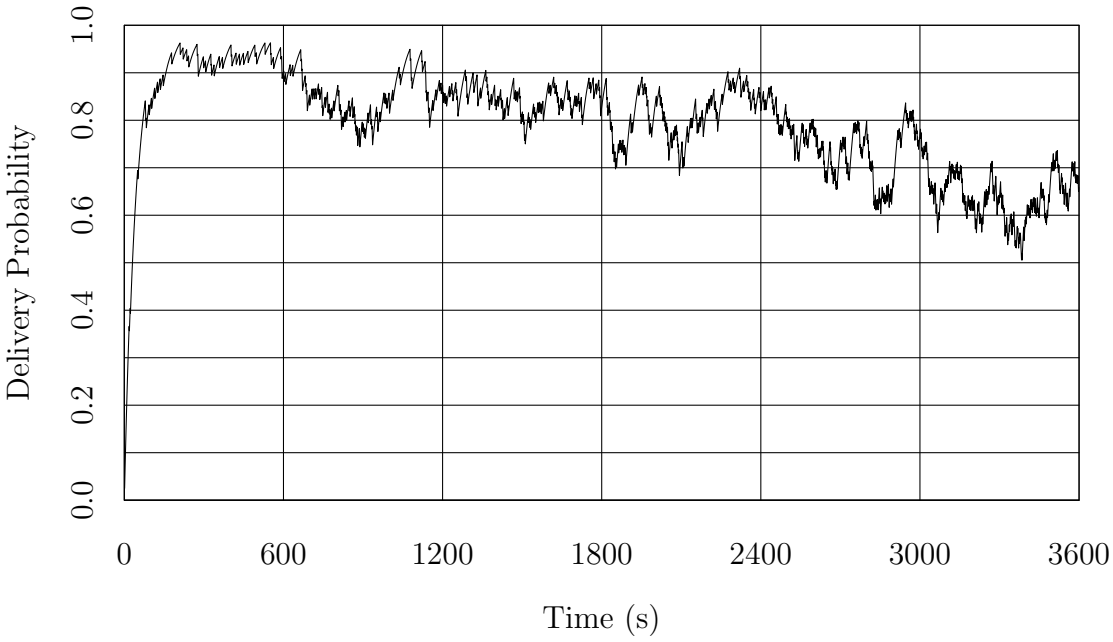


Figure A.7: Evolution of the delivery probability of the link $b \rightarrow a$ as a function time with varying transmission load at node a.

very small, considering the confidence interval. However, it is possible to state with 95% of confidence that the probability decreases with the increase of the transmission load for the last 3 windows.

A.1.5 Fifth Experiment: Infra-Structured Mode

All the results presented so far were obtained in networks operating in IEEE 802.11 ad hoc mode. This brings the question of whether this phenomenon affects also interfaces operating in the infra-structured mode of the standard, since this mode is the most widely used. To evaluate this hypothesis, we performed one more experiment that consisted in repeating the second experiment, but with the interfaces configured to operate in the infra-structured mode. In this setup, node a was configured to operate as an Access Point, while node b acts as an associated client.

Notice that in infra-structured mode, when a client transmits a broadcast frame, it is actually first transmitted in unicast to the access point, which, in turn, retransmits it in broadcast. This means that such frames are susceptible to retransmissions by the link layer of node b. Therefore, to accurately measure the delivery probability we use data

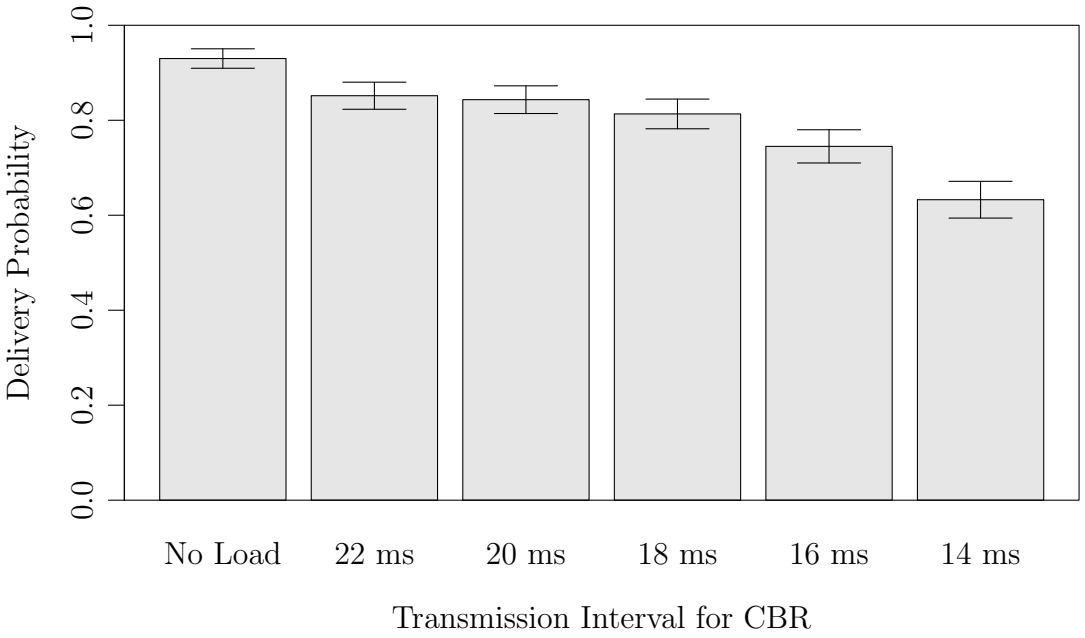


Figure A.8: Average delivery probability of the link $b \rightarrow a$ for each different transmission load for node a.

captured by node c (the sniffer), as described in Section A.1.3. Similarly to the third experiment, then, we define the delivery probability as the ratio between the number of unique frames and the total number of transmissions (including retransmissions), as reported by node c.

Figure A.9 shows the results obtained for the Atheros wireless interfaces. The graph shows that during the intervals in which the concurrent traffic is off (*e.g.*, from 600 to 1200 seconds), node a receives the frames from node b with probability very close to 1 (*i.e.*, there are very few retransmissions). On the other hand, during the intervals in which there is concurrent traffic (*e.g.*, from 0 to 600 seconds), the delivery probability visibly drops to around 90%. This behavior is very similar to the one reported in all previous experiments, suggesting that the infra-structured mode is also prone to the phenomenon.

A.1.6 Collisions at the Sniffer vs. Collisions at Node *a*

Throughout our experiments, we always assume that when a packet is lost due to collision at node a, it should, with high probability, also be lost due to collision at the sniffer. However, if two packets are transmitted at the same time but one of them reaches a

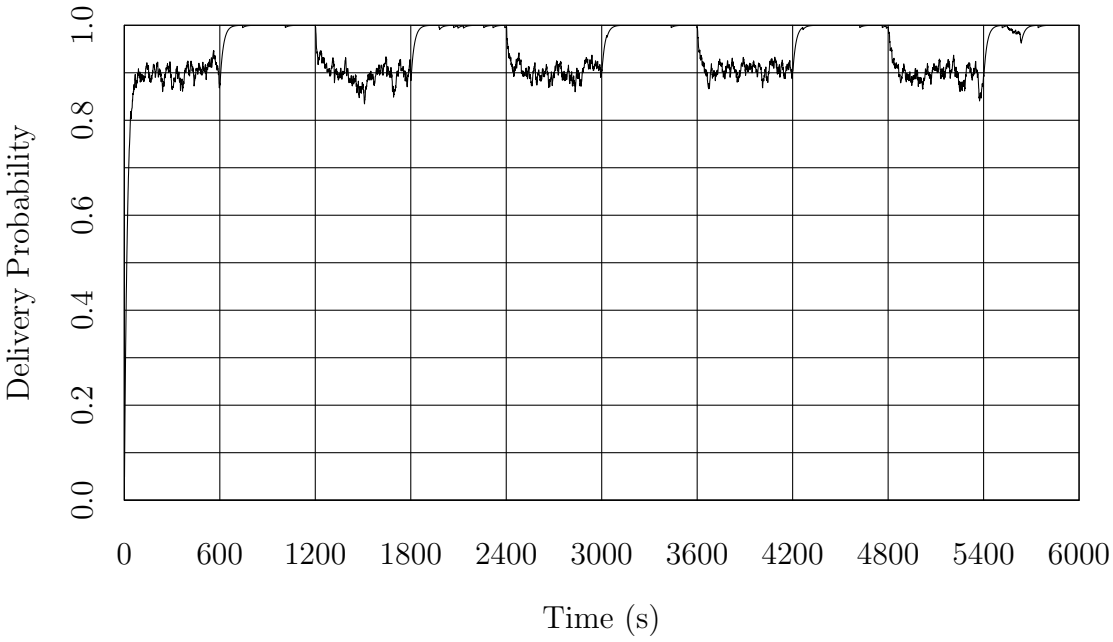


Figure A.9: Average delivery probability of the link $b \rightarrow a$ as a function of time in the fifth experiment.

receiver with much higher power than the other, the receiver might be able to correctly receive the first. This is known in the literature as the *capture effect* [31]. Hence, in general, a packet may be lost due to collision at one receiver and, yet, be correctly received by another. However, we argue that in our scenario, given the physical disposition of the nodes (*i.e.*, the proximity of the nodes), our assumption is valid.

To prove the validity of our assumption in our scenario, we conducted an experiment as follows. Node a transmits a UDP flow to node b generating packets as fast as it can (*i.e.*, node a's interface constantly has packets to transmit). On the other hand, node b generates a constant bit-rate traffic with 1 packet transmitted every 100 ms addressed to node a. The sniffer captures packets from both flows and stores them in a file. Both flows have the same duration of 600 seconds. Node a also stores a sequence number for each packet received from b.

By the end of the experiment, node b had transmitted a total of 5994 packets from which 918 packets were lost by node a but successfully received by the sniffer. For each of those 918 packets, we applied the procedure depicted in Algorithm A.1. This procedure goes through the file generated by sniffer and classifies a packet lost by node a according

Table A.1: Distribution of the causes for packet losses by node a.

Cause	**No Collision**	**Possible Collision**
Percentual	93,14%	6,86%

to a set of rules.

Algorithm A.1 Algorithm used to classify the types of losses.

1: **function** ClassifyLoss(Sequence Number s)
2: Find p, the packet from flow $b \rightarrow a$ with sequence number s
3: Find p_{before}, the last packet from flow $a \rightarrow b$ on the log before p
4: Find p_{after}, the first packet from flow $a \rightarrow b$ on the log after p
5: **if** Sequence Number of p_{after} = Sequence Number of $p_{before} + 1$ **then return** *no collision*
6: **elsereturn** *possible collision*
7: **end if**
8: **end function**

Algorithm A.1 yields two different kinds of classifications: *possible collision* or *no collision*. The algorithm starts by looking for packet p with sequence number s from the flow $b \Rightarrow a$. Once p is found, the algorithm looks for two packets from the flow $a \Rightarrow b$: packet p_{before} received by the sniffer immediately before packet p and the packet p_{after} received immediately after p. Suppose packet p is lost by node a due to collision, but, because of the capture effect, the sniffer can correctly decode it. Let x be the sequence number of the packet p_x from the flow $a \Rightarrow b$ that collided with p. Packet p_x cannot have been received by the sniffer, since the sniffer captured packet p. Hence, the sequence number from p_{before} has to be, at most, $x - 1$, while the sequence number from p_{after} has to be, at least, $x + 1$. It follows, then, that the difference between the sequence numbers from p_{before} and p_{after} has to be, at least, 2. If this is not the case (*i.e.*, if the sequence numbers are consecutive), the loss cannot have been due to collision. Therefore, Algorithm A.1 returns *no collision*. Notice that, even when the algorithm yields a *possible collision* classification, it is still doubtful that the loss was actually due to collision.

The results obtained by applying Algorithm A.1 to the traces generated by this experiment are summarized in Table A.1. The table shows that, during this experiment, whenever a packet was lost by node a but successfully received by the sniffer, there was a chance of more than 93% that the loss could not have happened due to collision. Therefore, this experiment demonstrates that our assumption is reasonable in our scenario.

A.2 Discussion

During the first experiment presented in Section A.1, the routing protocol clearly experienced the effect of decrease in the estimated delivery probability under heavy traffic. Although this result is expected given the previous reports in the literature, the causes of this decrease are somewhat unexpected.

As the results show, the losses of the probes are not caused by buffer overflows, given that, contrary to intuition, the most noticeable drops in delivery probability happened in the opposite direction of the concurrent data flow. Another potential explanation for the phenomenon, the losses due to collision are not causing this behavior, given that the sniffer was able to correctly decode the probe packets generated by both nodes. As show in Section A.1.6, in our scenario, whenever the sniffer is able to receive a packet, it is possible to discard the possibility of collision at node a with high probability.

As the fourth experiment shows, there is a correlation between the transmission load of the interface and its capability of receiving a frame. This implies that when the interface is in the transmission state (or in one of its sub-states), it is unable to receive frames, at least as efficiently as it would under lighter loads. Notice that by *transmission state* we do not refer to the actual physical transmission of the frame through the medium. If that was the case, frames would have been lost due to collision, leading our sniffer to fail to receive the frames as well. Therefore, we refer to the sub-states of the transmission process, such as carrier sensing and backoff.

This kind of behavior exists in simpler radios as well. One such example is ATMEL's AT86RF230 [8], which implements the IEEE 802.15.4 standard [48]. In the specific case of this radio, the hardware contains only one buffer to store frames for reception and transmission. Hence, when a frame is in the buffer to be transmitted, the radio cannot receive another frame while it expects the medium to be free, as this would override the bits of the outgoing frame. In the case of the results reported in Section A.1, though, it is not possible to pinpoint the exact limitation in the implementations that cause the unexpected behavior since the documentation for the evaluated chipsets is scarce. Nevertheless, the presented results clearly point out the existence of such limitation.

We contacted some vendors that develop IEEE 802.11 network interfaces regarding the obtained results. However, the only information we received was that recent projects of IEEE 802.11n radios include more buffers in the reception and transmission chains.

Independently of its cause, this behavior can potentially have a number of implica-

tions. The most obvious issue is the reduction of links' performance under bidirectional traffic. As shown by the results of the third experiment of Section A.1, since both nodes are transmitting, both have a lower capacity of receiving frames, thus decreasing the delivery probability on both directions. This increases the packet loss rate at the higher layers, as well as the effective link transmission delay (considering all necessary retransmission attempts).

Another issue concerns rate adaptation algorithms. Some of those algorithms, such as ARF (Auto Rate Fallback) [53], rely heavily on the assumption that the lower the bitrate used by the physical layer, the more robust it is and most of them use frame losses as an indication of degradation of the link quality. Under bidirectional load, the delivery probability decreases, which would lead a rate adaptation algorithm such as ARF to lower the interface bitrate. However, by lowering this bitrate, the rate adaptation algorithm would be only aggravating the issue, since this would be equivalent to increase the transmission load of the interface.

APPENDIX B – Joint Reception Probability Estimation

There are a number of factors which affect the result (success or failure) of the reception of a packet. Among them, one can cite:

- transmission power of the source;
- transmission rate of the packet (modulation and symbol rate);
- reception sensitivity of the receiver's radio;
- noise and interference in the receiver's location;
- distance, obstacles and propagation paths.

Considering a given coded transmission attempt, the power and rate are fixed for every intended receiver. The radio sensitivity, however, may differ among different network nodes. Nevertheless, the sensitivity values reported by manufactures for radios of the same technology are usually similar. Hence, for a generic evaluation, it is reasonable to consider that this parameter is also fixed for all receivers.

The last two factors are the ones that, indeed, present the most variability among different receivers. Since each receiver occupies a different position in space, they are prone to different sources and magnitudes of interference. Likewise, in most real environments, there are different obstacles in the propagation path between the source and each receiver, resulting in different levels of signal attenuation. For this reason, links for different receivers present different reception probabilities.

However, these two factors are directly related to the physical position of the nodes. Figure B.1 shows an example. In this hypothetical situation, the gray squares represent interference sources (*e.g.*, electronic devices generating noise in the network frequency or

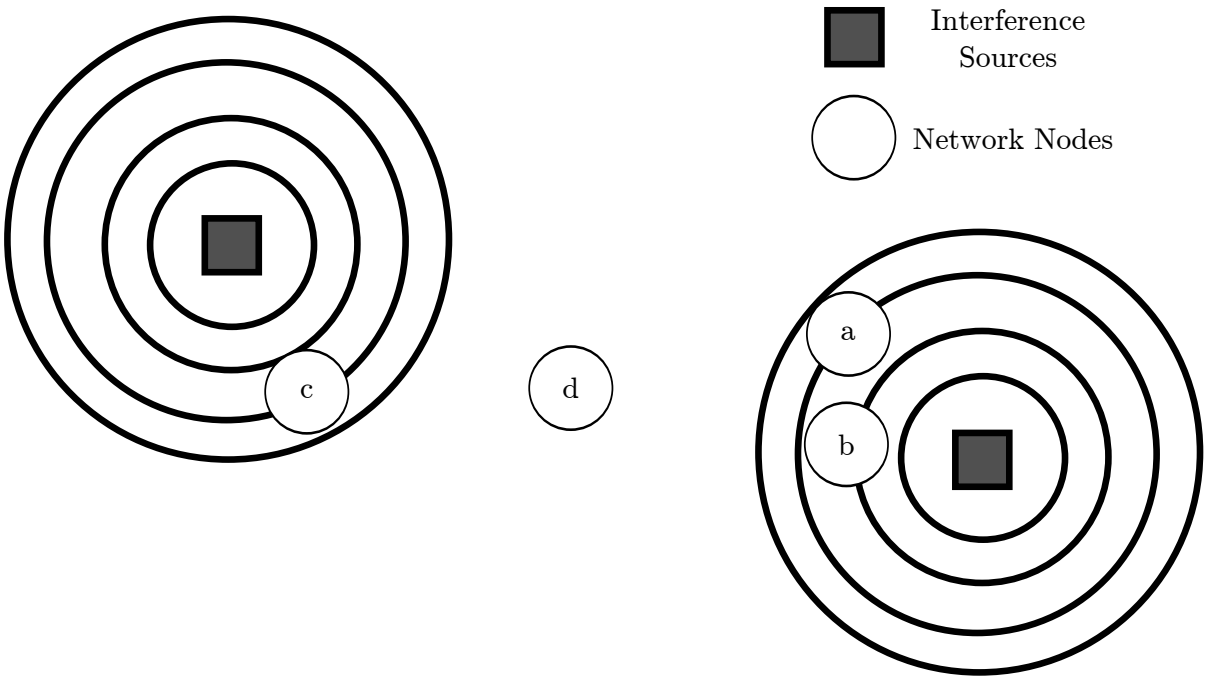


Figure B.1: Example of interference sources affecting nodes of a wireless network. Closer nodes tend to suffer effects from the same sources, with similar intensity. Distant nodes, in general, suffer effects from distinct sources.

nodes from other networks using the same channel). The concentric circles around the interference sources illustrate the interference zone caused by the signal generated by these devices. The remaining circles show the position of each network node. In this scenario, it is reasonable to assume that the interference sources that affect node a (such as the rightmost source) are very similar to the ones that affect node b, because both nodes are close. On the other hand, the interference sources that affect node c (such as the leftmost source) may be completely different, since c is distant from a and b.

Suppose node d wishes to send a coded packet for nodes a and c. In a given transmission attempt, it is possible that the leftmost interference source is generating lower noise, allowing c to correctly receive the packet. At the same time, the level of noise generated by the rightmost source can be high enough so that a is not able to correctly decode it. Hence, in this case, the hypothesis of independent reception probabilities is reasonable. However, the situation changes when we consider a and b as the packet receivers. If the rightmost interference source stops generating noise at the moment of transmission of the packet, both a and b are favored. Conversely, if this interference source generates high levels of noise during the transmission, both nodes have lower probability of decoding the packet.

It is possible to look at this situation from another point of view. If a is able to correctly decode a transmission performed by d, this means that the noise level generated by the rightmost interference source was low enough. In this case, the reception probability of node b for the same transmission attempt increases. On the other hand, given that a was not able to correctly decode the transmission, we can assume the environment noise was excessively high. Therefore, the reception probability of b for this transmission attempt decreases. Mathematically, assuming both nodes have non-zero reception probabilities, we can write:

$$P(Suc_b|Suc_a) > P(Suc_b|\neg Suc_a), \tag{B.1}$$

where Suc_a and Suc_b denote the events of successful reception of the coded packet by a and b, respectively. It follows that:

$$P(Suc_b|Suc_a) \neq P(Suc_b) \tag{B.2}$$

or

$$P(Suc_b|\neg Suc_a) \neq P(Suc_b) \tag{B.3}$$

Therefore, the occurrence or not of the reception event at node a makes the occurrence of the reception event at node b more or less likely. It follows that, under these conditions, the two events are not independent. This argument can be easily extended for more than two nodes.

B.1 Experimental Evidences

Although the previous example clearly favors the argument of dependence between reception events in specific scenarios, it is not clear whether in practice this situation is common. To evaluate this, we performed experiments in a real wireless mesh network. The employed network is a the same indoor topology used in Chapter 4 to evaluate the methods for sending coded packets, illustrated in Figure 4.1. The lines between nodes represent links frequently available in the network. Each network node is an off-the-shelf Linksys WRT54g router running the Linux-based OpenWrt operating system.

The experiment consists of sending, from a single source node, a sequence of broadcast packets (representing coded packets for all neighbors). Upon the reception of such a packet, a node registers the received sequence number in a log file. At the end of the experiment, logs for all neighbors are crossed to estimate the probabilities associated with the various reception events. The experiment was repeated 10 times, so that each

network node could be used once as a source node. The interval between packets was set to 100 ms and each repetition ran for 300 s, resulting in a total of 3000 packets of 1500 bytes. While processing the logs, empty files were discarded and the associated nodes were not considered neighbors. From the remaining files, the joint reception probabilities were computed for each pair of neighbors. This resulted in 71 tuples of the form $< source, receiver\_A, receiver\_B >$.

Table B.1 shows a subset of the tuples. The upper half of the table shows tuples for which we detected high dependence between reception events, while the lower half shows tuples for which low dependence was detected. We define a tuple with high dependence as one for which either $|P(a|b) - P(a)| >= 0.1$ or $|P(b|a) - P(b)| >= 0.1$. Otherwise, the tuple is considered to present low dependence.

In the upper half, we can see the tuples $< 2, 3, 5 >$, $< 3, 7, 6 >$ and $< 7, 5, 3 >$, for which both receivers have an average individual reception probability (between 37% and 59.5%), but which increases considerably when we look at the conditional events (varying from 14 to 20 percent points of increase in these cases). In these 3 cases there is a certain proximity between the receivers, which suggests that both are prone to similar interference conditions, as in the example of Figure B.1. The other two cases in this half of the table show situations in which one receiver has a reasonable reception probability (above 61%), whereas the other has a low reception probability (below 38%). In these cases, it is worth noting the significant increase in the reception probability for the most likely receiver, given that the least likely receives the packet. For the tuple $< 8, 5, 6 >$, for instance, the reception probability for node 6 increases more that 20 percent points, given that node 5 receives the packet, becoming an almost certain event. Once again, the proximity of both receivers causes a large intersection between the reception events for both nodes.

Regarding the lower half of the table, in almost all cases the tuples have a "central" source node and diametrically opposed receivers. Specifically, node 5 appears in 3 tuples as the source node. Indeed, this node is located in the middle of the topology and is able to communicate with a large number of neighbors. For these tuples, there is a small intersection between reception events for the two receivers. In other words, as expected, the reception events are independent (or present very low dependence). The tuple $< 2, 5, 4 >$, however, presents two relatively close receivers. In fact, the disposition of the nodes of this tuple is very similar to those of the tuples found in the upper half of the table. Nevertheless, the joint probabilities do not present an increase higher than 6 percent points in comparison to the individual probabilities. In this case, some other

Table B.1: Estimates for the joint probabilities, for pairs of receivers, obtained in the experiments. Column s represents the source, whereas a and b represent the two receivers.

s	**a**	**b**	**P(a)**	**P(b)**	**P(a ∩ b)**	**P(a|b)**	**P(b|a)**
High Dependence							
2	3	5	0.391	0.370	0.197	0.533	0.504
3	7	6	0.595	0.648	0.473	0.730	0.795
7	5	3	0.537	0.555	0.406	0.732	0.756
5	7	8	0.610	0.079	0.074	0.937	0.122
8	5	6	0.380	0.760	0.367	0.482	0.964
Low Dependence							
2	5	4	0.391	0.281	0.126	0.450	0.323
3	6	2	0.648	0.369	0.245	0.663	0.378
3	7	2	0.595	0.369	0.212	0.575	0.357
5	7	4	0.610	0.976	0.594	0.609	0.974
5	7	3	0.610	0.722	0.441	0.611	0.722
5	8	2	0.079	0.123	0.011	0.087	0.134
6	8	4	0.361	0.065	0.022	0.333	0.060
6	8	3	0.361	0.456	0.141	0.309	0.391

propagation particularity has more influence over the results than simply the physical distribution of the nodes.

These results show that, in many cases, is not reasonable to ignore the dependence between the reception events. Even in a real environment, these dependences can appear at different degrees, requiring a more accurate model not to negatively estimate coding and routing performances.

B.2 Estimating Joint Probabilities

As shown in this appendix, the reception events for a coded packet in two or more different receivers is not always independent. This is an important result because the literature on network coding usually assumes this independence, specially for computing the joint reception probability — *i.e.*, the individual delivery probabilities are multiplied and the result is considered to be the joint probability.

Since that is not always true, it is important to devise a method that can estimate the joint reception probability more accurately. We argue that this estimate can be obtained using basically the same resources employed to compute the individual probabilities.

The vast majority of routing protocols use those probes to derive statistics about link quality, such as the individual reception probabilities [17]. To compute these probabilities,

nodes keep, for every neighbor, information about received probes within a predefined window (of the last n probes). Prior to broadcasting its own probe, a node computes the reception probability regarding each neighbor and includes this information in the probe packet.

The method we propose here slightly changes this process. Instead of including the precomputed value of the reception probability in its probes, a node includes a bitmap for each neighbor representing the results (success or failure) for the reception of each probe within the current probe window. In this case, once a node receives the probes from its neighbors, it can compute not only the individual receiving probabilities, but also the joint receiving probabilities for any subset of neighbors by simply performing a bitwise *and* operation between the bitmaps. It is important to notice that the received bitmaps can be misaligned due to loss of probes and differences between the moments when each node transmits its probes. Therefore, when including a bitmap in a probe, nodes append the sequence number of the probe associated with the first position of the bitmap. Before computing the *and* operation over two bitmaps, it is necessary to shift them to obtain a correct alignment.

In terms of overhead, this solution does not increase the amount of transmitted probes. On the other hand, the size of each probe is increased. To represent a bitmap for the last n probes, n bits are necessary, whereas the precomputed reception probability can be encoded using only $log_2(n)$ bits (assuming a predefined size for the probe window). In practice, however, the number of probes used for computing the reception probabilities is relatively small. For instance, in the implementation of the OLSR (Optimized Link State Routing) protocol available in [68], the default size for the window is 20 probes. For a window of this size, less bits are necessary to encode the bitmap than to represent a single precision float point value using the IEEE 754 standard.

References

[1] R. Ahlswede, N. Cai, S. Li, and R. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[3] I. F. Akyildiz and X. Wang. A survey on wireless mesh networks. *IEEE Communications Magazine*, 43(9):S23–S30, 2005.

[4] I. F. Akyildiz, X. Wang, and W. Wang. Wireless mesh networks: a survey. *Computer networks*, 47(4):445–487, 2005.

[5] J. N. Al-Karaki and A. E. Kamal. Routing techniques in wireless sensor networks: a survey. *IEEE Wireless Communications*, 11(6):6–28, 2004.

[6] U. Ashraf, S. Abdellatif, and G. Juanole. An interference and link-quality aware routing metric for wireless mesh networks. In *IEEE 68th Vehicular Technology Conference (VTC 2008-Fall)*, pages 1–5. IEEE, 2008.

[7] ath9k – Linux Wireless. `http://wireless.kernel.org/en/users/Drivers/ath9k`, 2013. Accessed in 06/26/2013.

[8] Atmel Corporation. AT86RF230: Low power 2.4 GHz transceiver for ZigBee, IEEE 802.15.4, 6LoWPAN, RF4CE and ISM applications, 2009.

[9] N. Baldo, F. Maguolo, and S. Merlin. dei80211mr: an enhanced 802.11 implementation for ns2 and nsmiracle, 2013. Available at `http://telecom.dei.unipd.it/media/download/225`.

[10] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear Programming and Network Flows*. Wiley, 3°edition, 2004.

[11] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, 1987.

[12] J. Bicket. Bit-rate selection in wireless networks. Master's thesis, Massachusetts Institute of Technology, Cambridge, Feb. 2005.

[13] J. Bicket, D. Aguayo, S. Biswas, and R. Morris. Architecture and evaluation of an unplanned 802.11b mesh network. In *Proceedings of the 11th Annual International Conference on Mobile Computing and Networking (MobiCom 2005)*, pages 31–42, 2005.

[14] S. Biswas and R. Morris. Opportunistic routing in multi-hop wireless networks. *ACM SIGCOMM Computer Communication Review*, 34(1):69–74, 2004.

[15] J. Boyer, D. D. Falconer, and H. Yanikomeroglu. Multihop diversity in wireless relaying channels. *IEEE Transactions on Communications*, 52(10):1820–1830, 2004.

[16] M. E. M. Campista, L. H. M. K. Costa, and O. C. Duarte. A routing protocol suitable for backhaul access in wireless mesh networks. *Computer Networks*, 56(2):703–718, 2012.

[17] M. E. M. Campista, D. Passos, P. M. Esposito, I. M. Moraes, C. V. N. Albuquerque, D. C. M. Saade, M. G. Rubinstein, L. H. M. K. Costa, and O. C. M. B. Duarte. Routing metrics and protocols for wireless mesh networks. *IEEE Network*, 22(1):6–12, Jan./Feb. 2008.

[18] S. Chachulski, M. Jennings, S. Katti, and D. Katabi. Trading structure for randomness in wireless opportunistic routing. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2007)*, pages 169–180. ACM, 2007.

[19] X. Cheng, P. Mohapatra, S.-J. Lee, and S. Banerjee. Maria: Interference-aware admission control and qos routing in wireless mesh networks. In *Proceedings of the IEEE International Conference on Communications (ICC 2008)*, pages 2865–2870. IEEE, 2008.

[20] P. A. Chou, Y. Wu, and K. Jain. Practical network coding. In *Proceedings of the 41st Annual Allerton Conference on Communication Control and Computing*, pages 40–49, 2003.

[21] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626, Oct. 2003.

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2°edition, 2001.

[23] D. S. J. D. Couto. *High-throughput routing for multi-hop wireless networks*. PhD thesis, Massachusetts Institute of Technology, 2004.

[24] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking (MobiCom 2003)*, pages 134–146, 2003.

[25] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. *Wireless Networks*, 11(4):419–434, 2005.

[26] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[27] X. Ding, X. Zhang, M. Fan, and Y. Yang. Coding-aware routing for unicast sessions in wireless networks. In *Proceedings of the 5th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2009)*, pages 1–4, Sept. 2009.

[28] R. Draves, J. Padhye, and B. Zill. Routing in multi-radio, multi-hop wireless mesh networks. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking (MobiCom 2004)*, pages 114–128. ACM, 2004.

[29] P. Elias, A. Feinstein, and C. Shannon. A note on the maximum flow through a network. *IRE Transactions on Information Theory*, 2(4):117–119, Dec. 1956.

[30] K. Fan, X. Wei, and D. Long. A load-balanced route selection for network coding in wireless mesh networks. In *Proceedings of the IEEE International Conference on Communications (ICC 2009)*, pages 1–6, June 2009.

[31] S. Ganu, K. Ramachandran, M. Gruteser, I. Seskar, and J. Deng. Methods for restoring mac layer fairness in ieee 802.11 networks with physical layer capture. In *Proceedings of the 2nd International Workshop on Multi-hop Ad Hoc Networks: From Theory To Reality (REALMAN 2006)*, pages 7–14, New York, NY, USA, 2006. ACM.

[32] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. A Series of Books in the Mathematical Sciences. W. H. Freeman, 1979.

[33] I. Gitman, R. V. Slyke, and H. Frank. Routing in packet-switching broadcast radio networks. *IEEE Transactions on Communications*, 24(8):926–930, 1976.

[34] C. Gkantsidis and P. R. Rodriguez. Network coding for large scale content distribution. In *Proceedings of the 24th IEEE International Conference on Computer Communications (INFOCOM 2005)*, pages 2235–2245, 2005.

[35] A. D. Gore, A. Karandikar, and S. Jagabathula. On high spatial reuse link scheduling in stdma wireless ad hoc networks. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM 2007)*, pages 736–741, 2007.

[36] O. Goussevskaia, L. Vieira, and M. Vieira. Wireless multi-rate scheduling: From physical interference to disk graphs. In *Proceedings of the 37th IEEE Conference on Local Computer Networks (LCN 2012)*, pages 651–658, 2012.

[37] O. Goussevskaia and R. Wattenhofer. Scheduling with interference decoding: Complexity and algorithms. *Ad Hoc Networks*, 11(6):1732–1745, 2013.

[38] J. Grönkvist and A. Hansson. Comparison between graph-based and interference-based STDMA scheduling. In *Proceedings of the 2nd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2001)*, pages 255–258, 2001.

[39] P. Gupta and P. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, 2000.

[40] J. Ha, K. Lee, H. Kim, and I. Kang. A snooping rate adaptation algorithm for IEEE 802.11 WLANs. In *Proceedings of the 3rd International Symposium on Wireless Pervasive Computing (ISWPC 2008)*, pages 606–609, 2008.

[41] X. Hei, Y. Liu, and K. W. Ross. Inferring network-wide quality in p2p live streaming systems. *IEEE Journal on Selected Areas in Communications*, 25(9):1640–1654, Dec. 2007.

[42] T. Ho, R. Koetter, M. Médard, D. R. Karger, and M. Effros. The benefits of coding over routing in a randomized setting. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT 2003)*, pages 442–442, 2003.

[43] T. Ho and D. S. Lun. *Network Coding: An Introduction*. Cambridge University Press, 2008.

[44] T. Ho, M. Médard, J. Shi, M. Effros, and D. R. Karger. On randomized network coding. In *Proceedings of the 41st Annual Allerton Conference on Communication Control and Computing*, pages 11–20, 2003.

[45] G. Holland, N. Vaidya, and P. Bahl. A rate-adaptive MAC protocol for multi-hop wireless networks. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom 2001)*, pages 236–251, 2001.

[46] B. Hubert. Linux advanced routing & traffic control HOWTO. Online, Oct. 2003. Available: http://ds9a.nl/2.4Networking/lartc.pdf.

[47] IEEE LAN/MAN Standards Committee. 802.16-2009 IEEE standard for local and metropolitan area networks – part 16: Air interface for broadband wireless access systems, 2009.

[48] IEEE LAN/MAN Standards Committee. 802.15.4-2011 IEEE standard for local and metropolitan area networks – part 15.4: Low-rate wireless personal area networks (LR-WPANs), 2011.

[49] IEEE LAN/MAN Standards Committee. 802.11-2012 IEEE standard for information technology – lan/man – specific requirements – part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specification, 2012.

[50] K. Jain, J. Padhye, V. N. Padmanabhan, and L. Qiu. Impact of interference on multi-hop wireless network performance. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking (MobiCom 2003)*, pages 66–80. ACM, 2003.

[51] R. Jain, D.-M. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical report, DEC Research Report TR-30, Sept. 1984.

[52] D. B. Johnson, D. A. Maltz, and J. Broch. DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks. In *Ad Hoc Networking*, chapter 5, pages 139–172. Addison-Wesley, 2001.

[53] A. Kamerman and L. Monteban. WaveLAN-II: A high-performance wireless LAN for the unlicensed band. *Bell System Technical Journal*, 2(3):118–133, 1997.

[54] P. Karn. Maca-a new channel access method for packet radio. In *Proceedings of the ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, volume 140, pages 134–140, 1990.

[55] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft. XORs in the air: practical network coding. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2006)*, 2006.

[56] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft. XORs in the air: practical wireless network coding. *IEEE/ACM Transactions on Networking*, 16(3):497–510, June 2008.

[57] J. Kim, S. Kim, S. Choi, and D. Qiao. CARA: Collision-aware rate adaptation for IEEE 802.11 WLANs. In *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM 2006)*, pages 1–11, 2006.

[58] D. Koutsonikolas, C.-C. Wang, and Y. C. Hu. CCACK: Efficient network coding based opportunistic routing through cumulative coded acknowledgments. In *Proceedings of the 29th IEEE International Conference on Computer Communications (INFOCOM 2010)*, pages 1–9, march 2010.

[59] M. Krasnyansky. Universal TUN/TAP device driver. Documentation, Linux Kernel, 2000. Available at `https://www.kernel.org/doc/Documentation/networking/tuntap.txt`.

[60] J. Le, J. C. S. Lui, and D. M. Chiu. How many packets can we encode? an analysis of practical wireless network coding. In *Proceedings of the 27th IEEE International Conference on Computer Communications (INFOCOM 2008)*, pages 371–375, Apr. 2008.

[61] J. Le, J. C. S. Lui, and D.-M. Chiu. DCAR: Distributed coding-aware routing in wireless networks. *IEEE Transactions on Mobile Computing*, 9(4):596–608, Apr. 2010.

[62] J. Li, C. Blake, D. S. J. De Couto, H. I. Lee, and R. Morris. Capacity of ad hoc wireless networks. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom 2001)*, pages 61–69. ACM, 2001.

[63] Z. Li and B. Li. Network coding in undirected networks. In *Proceedings of the 38th Annual Conference on Information Sciences and Systems (CISS 2004)*, pages 257–262, 2004.

[64] D. S. Lun, M. Médard, and R. Koetter. Efficient operation of wireless packet networks using network coding. In *Proceedings of the International Workshop on Convergent Technologies (IWCT 2005)*, 2005.

[65] D. S. Lun, M. Médard, R. Koetter, and M. Effros. Further results on coding for reliable communication over packet networks. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT 2005)*, pages 1848–1852, Sept. 2005.

[66] D. S. Lun, M. Médard, R. Koetter, and M. Effros. On coding for reliable communication over packet networks. *Physical Communication*, 1(1):3–20, 2008.

[67] S. Nelakuditi, S. Lee, Y. Yu, J. Wang, Z. Zhong, G.-H. Lu, and Z.-L. Zhang. Blacklist-aided forwarding in static multihop wireless networks. In *Proceedings of the IEEE International Conference on Sensing, Communication, and Networking (SECON 2005)*, pages 252–262, 2005.

[68] OLSRD. `http://www.olsr.org`, 2013. Accessed in 05/06/2013.

[69] Open-Mesh. `http://www.open-mesh.org/projects/open-mesh/wiki`, 2013. Accessed in 05/06/2013.

[70] OpenWrt. `http://www.openwrt.org`, 2011. Accessed in 08/07/2013.

[71] D. Passos. Uma abordagem unificada para métricas de roteamento e adaptação automática de taxa em redes em malha sem fio. Master's thesis, Universidade Federal Fluminense, Niterói, 2010. In Portuguese.

[72] D. Passos. IAR and ICAR implementations for ns-2, 2013. Available at `http://www.midiacom.uff.br/~diego/SLSPCodingNS.tar.gz`.

[73] D. Passos. IAR and ICAR implementations for openwrt, 2013. Available at `http://www.midiacom.uff.br/~diego/SLSPCoding.tar.gz`.

[74] D. Passos and C. V. N. Albuquerque. Probabilidade condicional de sucesso no envio de pacotes codificados aplicada a roteamento ciente de codificação em redes sem fio de múltiplos saltos. In *29º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 103–116, May 2011. In Portuguese.

[75] D. Passos and C. V. N. Albuquerque. A joint approach to routing metrics and rate adaptation in wireless mesh networks. *IEEE/ACM Transactions on Networking*, 20(4):999–1009, Aug. 2012.

[76] D. Passos and C. V. N. Albuquerque. Modeling the transmission of coded packets for coding aware routing. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM 2012)*, pages 136–142, 2012.

[77] D. Passos, R. Carrano, and C. V. N. Albuquerque. On the decrease in frame reception probability under heavy transmission loads in ieee 802.11 networks. *Computer Standards & Interfaces*, 35:374–379, 2013.

[78] D. Passos, C. V. N. de Albuquerque, M. E. M. Campista, L. H. M. K. Costa, and O. C. M. B. Duarte. Minimum loss multiplicative routing metrics for wireless mesh networks. *Journal of Internet Services and Applications*, 1(3), Feb. 2011.

[79] J. Pavon and S. Choi. Link adaptation strategy for IEEE 802.11 WLAN via received signal strength measurement. In *Proceedings of the IEEE International Conference on Communications (ICC 2003)*, volume 2, pages 1108–1113, 2003.

[80] W. Peng and X.-C. Lu. On the reduction of broadcast redundancy in mobile ad hoc networks. In *Proceedings of the 1th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2000)*, pages 129–130, Piscataway, NJ, USA, 2000. IEEE Press.

[81] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *ACM SIGCOMM Computer Communication Review*, 24(4):234–244, 1994.

[82] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 1999)*, pages 90–100. IEEE, 1999.

[83] Promoter Members of Bluetooth SIG, Inc. Specification of the bluetooth system – master table of contents & compliance requirements – covered core package version: 4.0, 2010.

[84] B. Radunovic, C. Gkantsidis, P. Key, and P. Rodriguez. Toward practical opportunistic routing with intra-session network coding for mesh networks. *IEEE/ACM Transactions on Networking*, 18(2):420–433, Apr. 2010.

[85] R. Rajaraman. Topology control and routing in ad hoc networks: a survey. *SIGACT News*, 33(2):60–73, June 2002.

[86] T. S. Rappaport. *Wireless communications: principles and practice*, volume 2. Prentice Hall PTR New Jersey, 1996.

[87] S. Sengupta, S. Rayanchu, and S. Banerjee. An analysis of wireless network coding for unicast sessions: The case for coding-aware routing. In *Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM 2007)*, pages 1028–1036, May 2007.

[88] S. Sengupta, S. Rayanchu, and S. Banerjee. Network coding-aware routing in wireless networks. *IEEE/ACM Transactions on Networking*, 18(4):1158–1170, Aug. 2010.

[89] J. L. Sobrinho. Algebra and algorithms for qos path computation and hop-by-hop routing in the internet. *IEEE/ACM Transactions on Networking*, 10(4):541–550, Aug. 2002.

[90] A. P. Subramanian, M. M. Buddhikot, and S. Miller. Interference aware routing in multi-radio wireless mesh networks. In *Proceedings of the 2nd IEEE Workshop on Wireless Mesh Networks (WiMesh 2006)*, pages 55–63, 2006.

[91] J. Tang, G. Xue, and W. Zhang. Interference-aware topology control and qos routing in multi-channel wireless mesh networks. In *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2005)*, pages 68–77. ACM, 2005.

[92] The Click Modular Router Project. `http://www.read.cs.ucla.edu/click/click`, 2011. Accessed in 05/06/2013.

[93] The Network Simulator - ns-2. `http://isi.edu/nsnam/ns/`, 2011. Accessed in 08/07/2013.

[94] K. Thompson, G. J. Miller, and R. Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network*, 11(6):10–23, 1997.

[95] Y. Tian, K. Xu, and N. Ansari. Tcp in wireless environments: problems and solutions. *IEEE Communications Magazine*, 43(3):S27–S32, 2005.

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

[97] M. Wang and B. Li. Lava: A reality check of network coding in peer-to-peer live streaming. In *Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM 2007)*, pages 1082–1090, May 2007.

[98] M. Wang and B. Li. Network coding in live peer-to-peer streaming. *IEEE Transactions on Multimedia*, 9(8):1554–1567, Dec. 2007.

[99] M. Wang and B. Li. R2: Random push with random network coding in live peer-to-peer streaming. *IEEE Journal on Selected Areas in Communications*, 25(9):1655–1666, Dec. 2007.

[100] S. H. Y. Wong, H. Yang, S. Lu, and V. Bharghavan. Robust rate adaptation for 802.11 wireless networks. In *Proceedings of the 12nd Annual International Conference on Mobile Computing and Networking (MobiCom 2006)*, pages 146–157, 2006.

[101] Y. Yan, B. Zhang, J. Zheng, and J. Ma. Core: a coding-aware opportunistic routing mechanism for wireless mesh networks. *IEEE Wireless Communications*, 17:96–103, June 2010.

[102] Y. Yang, J. Wang, and R. Kravets. Interference-aware load balancing for multihop wireless networks. Technical report, University of Illinois at Urbana-Champaign, 2009.

[103] J. Y. Yen. An algorithm for finding shortest routes from all source nodes to a given destination in general networks. *Quarterly of Applied Mathematics*, 27(4):526–530, 1970.

[104] H. Yue, X. Zhu, C. Zhang, and Y. Fang. Cptt: A high-throughput coding-aware routing metric for multi-hop wireless networks. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM 2012)*, pages 5687–5692, 2012.

[105] Y. Zhu, B. Li, and J. Guo. Multicast with network coding in application-layer overlay networks. *IEEE Journal on Selected Areas in Communications*, 22(1):107–120, 2004.