Bringing Stability to Wireless Mesh Networks

THÈSE N^O 4960 (2011)

PRÉSENTÉE LE 18 MARS 2011

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE POUR LES COMMUNICATIONS INFORMATIQUES ET LEURS APPLICATIONS 3
PROGRAMME DOCTORAL EN INFORMATIQUE, COMMUNICATIONS ET INFORMATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Adel AZIZ

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2011

Abstract

Wireless mesh networks were designed as a mean to rapidly deliver large-scale communication capabilities without the support of any prior infrastructure. Among the different properties of mesh networks, the self-organizing feature is particularly interesting for developing countries or for emergency situations. However, these benefits also bring new challenges. For example, the scheduling decision needs to be performed in a distributed manner at each node of the network. Toward this goal, most of the current mesh deployments are based on the IEEE 802.11 protocol, even if it was not designed for multi-hop communications.

The main goals of this thesis are (i) to understand and model the behavior of IEEE 802.11-based mesh networks and more specifically the root causes that lead to congestion and network instability; (ii) to develop an experimental infrastructure in order to validate with measurements both the problems and the solutions discussed in this thesis; (iii) to build efficient hop-by-hop scheduling schemes that provide congestion control and inter-flow fairness in a practical way and that are backward-compatible with the current protocol; and (iv) to explain the non-monotonic relation between the end-to-end throughput and the source rate and to introduce a model to derive the rationale behind this artifact.

First, we propose a Markovian model and we introduce the notion of *stealing effect* to explain the root causes behind the 3-hop stability boundary, where linear networks up to 3 hops are stable, and larger topologies are intrinsically unstable. We validate our analytical results both through simulations and through measurements on a small testbed deployment.

Second, to support the experimental research presented in this thesis, we design and deploy a large-scale mesh network testbed on the EPFL campus. We plan our architecture to be as flexible as possible in order to support a wide range of other research areas such as IEEE 802.11 indoor localization and opportunistic routing.

Third, we introduce *EZ-flow*, a novel hop-by-hop congestion-control mechanism that operates at the Medium Access Control layer. EZ-flow is fully backward-compatible with the existing IEEE 802.11 deployments and it works without any form of message passing. To perform its task EZ-flow takes advantage of the broadcast nature of the wireless medium in order to passively derive the queue size at the next-hop node. This information is then used by each node to adapt accordingly its channel access probability, through the contention window parameter of IEEE 802.11. After detailing the different components of EZ-flow, we analyze its performance analytically, through simulations and real measurements.

Fourth, we show that hop-by-hop congestion-control can be efficiently performed at the network layer in order to not abuse the contention mechanism of IEEE 802.11. Additionally, we introduce a complete framework that jointly achieves congestion-control and fairness without requiring a prior knowledge of the network capacity region. To achieve the fairness part, we propose the *Explore & Enhance* algorithm that finds a fair and achievable rate allocation vector that maximizes a desired function of utility. We show experimentally that this algorithm reaches its objective by alternating between exploration phases (to discover the capacity region) and enhancement phases (to improve the utility through a gradient ascent).

Finally, we note that, as opposed to wired networks, the multi-hop wireless capacity is usually unknown and time-varying. Therefore, we study how the end-to-end throughput evolves as a function of the source rate when operating both *below* and *above* the network capacity. We note that this evolution follows a non-monotonic curve and we explain, through an analytical model and simulations, the rationale behind the different transition points of this curve. Following our analysis, we show that no end-to-end congestion control can be throughput-optimal if it operates directly over IEEE 802.11. Hence, this supports the methodology of performing congestion control in a hop-by-hop manner. After validating experimentally the non-monotonicity, we compare through simulations different state-of-the-art scheduling schemes and we highlight the important tradeoff that exists in congestion-control schemes between *efficiency* (i.e., throughput-optimality) and *robustness* (i.e., no throughput collapse when the sources attempt to operate at a rate above the network capacity).

Keywords

Wireless mesh networks, multi-hop networks, IEEE 802.11, scheduling, medium access control, congestion control, network stability, modeling, implementation, experimental measurements, testbed deployment.

Résumé

Les réseaux maillés sans fil ont été conçus afin de permettre le déploiement rapide d'un moyen de communication, et cela sans nécessiter le soutien d'une infrastructure préexistante. Parmi les avantages offerts par ces réseaux, l'auto-organisation semble particulièrement intéressante dans le cas de déploiements dans des pays émergents ou en situation de catastrophes naturelles. Cependant, ces avantages ne viennent pas sans nouveaux défis. Par exemple, la planification d'accès au canal doit être effectuée de manière distribuée. La plupart des déploiements de réseaux maillés sans fil actuels réalisent cela en utilisant le standard IEEE 802.11, même si ce protocole n'a pas été conçu pour des communications à sauts multiples.

Les principaux objectifs de cette thèse sont (i) de comprendre et de modéliser le comportement des réseaux maillés sans fil basés sur IEEE 802.11, en se concentrant sur les facteurs clés qui conduisent à l'instabilité et la congestion du réseau; (ii) de développer un réseau sans fil expérimental afin de valider avec des mesures réelles les problèmes et les solutions présentés dans cette thèse; (iii) de proposer des méthodes de planification d'accès au canal qui offrent simultanément un contrôle de congestion efficace et une forme d'équité entre les différents flux présents dans le réseau; et (iv) de souligner la non-monotonicité de la relation entre le débit de bout-en-bout et le débit reçu à la source, puis de proposer un modèle analytique pour en expliquer les raisons.

Dans un premier temps, nous proposons un modèle Markovien et nous introduisons la notion d'*effet de 'détournement'* pour expliquer les causes menant à la limite de stabilité de 3 sauts. Cette limite se manifeste par la stabilité des réseaux linéaires jusqu'à 3 sauts, en opposition à l'instabilité intrinsèque de plus grandes topologies. Nous validons nos résultats analytiques aussi bien par des simulations que par des mesures sur un réseau maillé à petite échelle.

Dans un deuxième temps, nous concevons et déployons un réseau maillé sans fil à grande échelle au sein du campus de l'EPFL. Nous planifions l'architecture du réseau afin qu'elle soit la plus flexible possible et qu'elle puisse soutenir un large éventail de thèmes de recherche tels que la localisation basée sur IEEE 802.11 et le routage opportuniste.

Dans un troisième temps, nous proposons *EZ-flow*, un nouveau mécanisme de contrôle de congestion lien-par-lien qui fonctionne au niveau de la couche MAC. EZ-flow est entièrement rétro-compatible avec les protocoles existants basés sur IEEE 802.11 et il fonctionne sans échange de messages de contrôle entre les nœuds.

Ce mécanisme tire parti de la nature diffusive du support sans fil: il calcule ainsi passivement la taille de la file du nœud suivant. Chaque nœud utilise ensuite cette information pour adapter sa probabilité d'accès au canal, en modifiant la taille de la fenêtre de contention de IEEE 802.11. Pour finir, nous analysons les performances du mécanisme EZ-flow en nous appuyant sur des résultats analytique, des simulations, et de mesures réelles.

Dans un quatrième temps, nous montrons que le contrôle de congestion lien-par-lien peut être effectué de manière tout aussi efficace au niveau de la couche réseau, afin de ne pas dénaturer le mécanisme se chargeant du contrôle de contention dans le protocole IEEE 802.11. En outre, nous développons une solution qui offre conjointement un contrôle de congestion et une forme d'équité entre les flux. Tout ceci est réalisé sans nécessiter la connaissance préalable de la région de capacité du réseau. Afin de fournir une forme d'équité, nous proposons l'algorithme *Explore & Enhance*. Cet algorithme trouve un vecteur d'allocation des débits réalisable maximisant une fonction d'utilité donnée. Nous montrons expérimentalement que ce mécanisme atteint cet objectif en alternant entre (i) des phases d'exploration afin découvrir la région de capacité et (ii) des phases d'amélioration qui font croitre l'utilité grâce à une montée de gradient.

Finalement, nous notons que, contrairement aux réseaux câblés, la capacité des réseaux sans fil à sauts multiples est généralement non seulement inconnue, mais aussi variable dans le temps. Il est donc primordial d'étudier l'évolution du débit de bout-en-bout du réseau lorsque l'on fait varier le débit reçu par les sources (aussi bien *en dessous* et qu'*au dessus* de la capacité du réseau). Nous remarquons que cette évolution est non monotone et expliquons les raisons de ce comportement en nous appuyant sur un modèle analytique et des simulations. Suite à la validation expérimentale de ce phénomène, nous montrons qu'il est impossible pour un mécanisme de contrôle de congestion fonctionnant de bout-en-bout d'atteindre le débit optimal si le protocole IEEE 802.11 est utilisé au niveau MAC. Cela soutient l'idée qu'un contrôle de congestion efficace doit s'effectuer lien-par-lien. Nous nous concentrons donc sur les différents mécanismes lien-par-lien et comparons leur performance, ce qui met en évidence l'importance du compromis qui existe entre *efficacité* (optimalité du débit) et *robustesse* (pas de chute du débit lorsque la quantité de trafic reçu par les sources est supérieure à la capacité du réseau).

Mots clés

Réseaux maillés sans fil, réseaux à sauts multiples, IEEE 802.11, protocole de gestion d'accès au canal, protocole de contrôle de congestion, stabilité des réseaux, modélisation, implémentation, mesures expérimentales, déploiement de réseaux.

Acknowledgments

I would like to thank my advisor Prof. Patrick Thiran for accepting me in his research group and for making this PhD such an enriching experience, both on a personal and a scientific level. I really appreciated his sense of scientific rigor and the freedom he gave me in exploring my own research interests.

In addition, I want to thank Sylviane Dal Mas for informing me about this PhD opportunity and for all her hard work she does to make the Communication Systems Section such a great department to study in. A special thank goes also to Dr. Roger Karrer from Deutsche Telekom Laboratories for having launched the MagNets project that mainly funded my research work. While working with him, I learned a lot from his experience in system research and this was a valuable complement to the more theoretical approach I acquired at EPFL.

It was a great pleasure and a humbling experience to have Prof. Martin Hasler, Prof. Jean-Pierre Hubaux, Dr. Konstantina Papagiannaki, and Prof. David Starobinski in my jury. I want to thank them for accepting to review this thesis.

Next, I would like to thank all the people with whom I had the opportunity to collaborate during my PhD research, in particular Dr. Alaeddine El Fawal, Julien Herzen, Dr. Ruben Merz, and Dr. Seva Shneer. I also want to thank all my EPFL colleagues for making this place such a lively environment.

I am also very thankful to the lab's staff for their support and enthusiasm. Thanks to Holly Cogliati for all her advice in English and for the dedication she showed in reviewing and helping me to improve the clarity of most of my publications. Thanks to Danielle Alvarez, Angela Devenoge, and Patricia Hjelt for making all the administrative processes go smoothly. Thanks to Hervé Chabanel, Yves Lopes, Marc-André Luthi, and Richard Timsit for letting me deploy my multi-hop testbed on the campus and for their support and advice.

Finally, I thank my parents for always being there for me and for the support and care they gave to me throughout my studies. I thank all my friends for making this PhD time a nice period of my life. In particular I thank Raphael and Mohamed for pushing me to keep the right balance between work and sports. And last but not least, I am really thankful to Mutunge for the precious support she gave me and for all the great moments we shared together during my PhD experience.

Contents

Chapter 1

Introduction

1.1 Motivation

Wireless Mesh Networks (WMNs) have received increasing attention since their introduction in 1994 under the name of *massive array cellular systems* [Pie94]. Initially, they were intended to be a cost-effective alternative to replace the last-mile infrastructure of large metropolitan cities. Examples of such deployments took place in cities such as San Francisco, where Meraki deployed a city-wide wireless mesh network in 2008. To realize this project, Meraki used around 10,000-15,000 indoor nodes and a few hundred solar-powered outdoor nodes [Fle08]. This type of successful commercial deployment shows that WMNs have the potential to deliver large-scale broadband connectivity to metropolitan areas. Nevertheless, due to widely available wired broadband connectivity, the utility of mesh networks has proved to be superfluous in developed countries. Consequently, mesh networks are now seen as an efficient way to quickly provide connectivity in uncovered areas, such as in developing countries or in emergency situations after a natural disaster, for example.

An example for the case of developing countries is the *One Laptop per Child* project launched in 2005 [OLP]. The goal of this project is to provide educational opportunities to the world's poorest children by giving them a low-cost and low-powered laptop with dedicated software. To reach this objective, a laptop named XO was designed. As this laptop is likely to be used in regions where little communication infrastructure exists, it relies on the wireless mesh network technology in order to provide a form of connectivity between the different machines.

Emergency situations can arise both in developing and developed countries. Indeed, different external factors could disrupt the smooth operation of the network. In the case of natural disasters, such as earthquakes or tsunamis, the wired infrastructure could be physically damaged thus preventing traditional communications from taking place. This lack of connectivity does not allow the rescue teams to efficiently synchronize, which leads to dramatic consequences in emergency situations, where every minute counts to save human lives. WMNs provide a serious solution to these situations as they allow for the rapid deployment of an operational network without relying on any pre-existing infrastructure. Moreover, recent events in Egypt have shown that connectivity can be shut down, even in the presence of a fully operational infrastructure. Indeed, during the Egyptian revolution of January 2011, the government reacted by disconnecting the whole country from Internet and switching off cellular communications. This communication blackout is an attack against freedom that could lead to chaos in the streets. The striking fact in this specific situation is that this country-wide blackout was relatively easy to perform for the government. This perfectly illustrates the vulnerability of the centralized communication technologies that we currently rely on. As mentioned in the New York Times [TOF11], wireless mesh networks technology could be deployed in smartphones and this might be a solution to prevent this type of connectivity blackout from happening in the future.

In order to not suffer from a single point of failure and to provide a reasonable level of robustness to hardware breakdown, mesh networks need to be decentralized. This requirement forces the scheduling decisions to be performed in a distributed manner with the transmission decision made locally at each node. Most of current WMN deployments rely directly on the IEEE 802.11 protocol to take the scheduling decision at the Medium Access Control (MAC) layer. But IEEE 802.11 was designed for single-hop networks and was not envisioned for multi-hop communications that significantly differ in nature and lead to new challenges.

In single-hop communications, IEEE 802.11 is widely used and it is the well-accepted standard for wireless local area networks (WLAN) that are largely deployed both in homes and offices. The particularity of these networks is that all the nodes are within the same collision set. This means that at most only one node can successfully transmit at each point in time, and each node can sense whether the channel is idle or if there is a communication taking place. This single-hop setting of IEEE 802.11 is modeled by Bianchi and he finds that the protocol performs reasonably well by delivering good throughput performance together with long-term fairness (short-term fairness is not achieved due to the exponential backoff policy of IEEE 802.11) [Bia00].

Nevertheless, the multi-hop environment is significantly different by nature. Our understanding of the exact behavior of multi-hop IEEE 802.11 networks is still in its infancy and the existence of relay nodes brings the additional challenge of congestion-control. Moreover, measurements from real deployments reflect poor performance [GSK04] and they lead some researchers to make surprising conclusions, such as ”*with current commodity wireless technology it does not make sense to handle more than three hops*” [1] (we call this finding the 3-*hop boundary* hereafter). Due to the lack of analytical models capturing the exact dynamics of IEEE 802.11 in multi-hop networks, we are still unable to explain the rationale behind such a 3-hop boundary result and this remains an open issue. Once the causes of this problem are understood, the next step would be to design appropriate practical

[1]Lunar project: http://cn.cs.unibas.ch/projects/lunar/

mechanisms that can overcome this challenge and that remain decentralized and backward-compatible with existing mesh network deployments.

Finally, we note that as opposed to wired networks, the wireless capacity is usually both unknown and time-varying. Therefore, without being too conservative, it is impossible to guarantee that the source rate is always within the network capacity. Yet, most of the recent works on distributed scheduling [SSR09, JWa, PYC08, TE92] focus on the notion of throughput-optimality that ensures that the network is stable for any source rate *within* the capacity region, which is therefore assumed to be known. This throughput-optimality criterion is useful (it gives a measure of *efficiency*), but it does not say anything about the network performance once the source rate is above the capacity (it remains clueless about *robustness*). Therefore, we stress that we really need to clearly understand how the network performance (i.e., the throughput) evolves for different source rates, either within or outside the capacity region. Indeed, an optimal scheduling scheme needs to be both *efficient* and *robust*.

1.2 Dissertation Outline

We begin by describing the IEEE 802.11 protocol and introducing the notion of wireless mesh networks in Chapter 2. After discussing the fundamental differences between a single-hop and a multi-hop environment, we describe some desirable properties of mesh networks, which need to be kept in consideration when designing new scheduling schemes.

In order to make it possible to formally study the root causes behind the 3-hop boundary (i.e., why is a 3-hop network stable, but not a 4-hop network), we propose a Markovian model in Chapter 3. We introduce the notion of *stealing effect*, a consequence of the hidden node problem and of non-zero transmission delays, and we discuss its impact on the network stability. After proposing a static stabilization strategy, we use six off-the-shelf wireless routers in order to validate experimentally both the instability result and the efficiency of our solution.

Because of the importance of experimental validation in the field of mesh networks and the lack of an experimental platform at our disposal, we decided to build from scratch an experimental multi-hop testbed on the EPFL campus. Our testbed is composed of around 60 wireless routers and it spans over the six buildings of the I&C department. In Chapter 4, we review some of the challenges and the practical lessons we learned while building our indoor testbed that was used for our own research work and is still used for various other projects today.

After pointing out the serious stability problem that occurs in wireless multi-hop networks, in Chapter 5 we introduce a practical hop-by-hop congestion-control scheme called *EZ-flow*. EZ-flow is designed to take advantage of the broadcast nature of the wireless medium in order for a node i to passively derive the queue occupancy at the next-hop q_{i+1} without any form of message passing or piggy-backing. Node i adapts its transmission rate in order to maintain the queue q_{i+1} stable. We validate the efficiency of EZ-flow in stabilizing the network (i.e., maintaining the end-to-end delay small) both through ns-2 simulations and through measurements from a practical implementation deployed on our indoor testbed.

After tackling the problem of congestion control within a flow at the MAC layer, in Chapter 6 we propose a more complete scheme that runs at the network layer and that delivers both intra-flow congestion-control and inter-flow fairness. Our distributed solution requires almost no message passing and is completely transparent to both the MAC (i.e., it does not interact with any parameter of the MAC layer) and the upper layers. First, our network-layer hop-by-hop congestion-control mechanism uses a rate limiter attached to each queue. It adaptively and automatically adjusts each rate limiter by passively computing the queue size at the next-hop relay, without any form of message passing. Second, at the mesh gateway, our inter-flow fairness algorithm finds a fair and achievable rate allocation vector that maximizes utility without prior knowledge of the capacity region. It runs (i) exploration phases to discover the capacity region and (ii) enhancement phases to improve the utility by a gradient ascent. Third, both mechanisms smoothly interact together to form a complete solution. The fair inter-flow allocation propagates into the network using the hop-by-hop intra-flow mechanism and we validate the efficiency of our solution on 12 wireless routers of our testbed.

Throughout this thesis, we focus on distributed scheduling schemes that can perform their task without the prior knowledge of the capacity region. As mentioned earlier, another approach taken by some researchers is to design throughput-optimal schemes that explicitly require the knowledge of the capacity region in order to perform their task. Nevertheless, as opposed to wired networks, the wireless capacity is usually unknown and thus it is important to have a clear understanding of how the network behaves when the sources are operating either *below* or *above* the network capacity. In Chapter 7, we formally study the case of an IEEE 802.11 multi-hop network in detail and we explain why the end-to-end throughput is a non-monotonic function of the source rate. Following our simulations and our mathematical study, we prove that it is impossible for an end-to-end congestion control scheme to be throughput-optimal if it runs over IEEE 802.11. This result supports the idea of performing congestion-control in a hop-by-hop manner instead of end-to-end. Therefore we compare in our simulator different state-of-the-art methodologies of performing hop-by-hop congestion control and we show the important tradeoff between optimality (throughput-optimality) and robustness (no throughput collapse beyond capacity) that should be taken into consideration when designing new hop-by-hop scheduling algorithms.

Finally, we conclude this thesis in Chapter 8 with a summary of the main findings and a discussion of possible directions for future work.

1.3 Contributions

Although wireless mesh networks are intrinsically unstable with the standard parameters, we show that it is possible to overcome this limitation through the use of new mechanisms that are both practical and backward-compatible. To support this statement, we provide the following main contributions in this thesis.

- We verify experimentally the 3-hop boundary in the stability of multi-hop networks and we propose a Markovian model that allows us to formally explain the root causes behind this artifact.

- We observe that in the case of multi-hop networks, packets collisions can help stabilize the network. Indeed, in the case of the *stealing effect*, the collisions help to favor downstream links toward upstream links. Moreover, if the probability of collisions (i.e., the probability of the stealing effect) is zero, then even a 3-hop network is unstable. In practice, this probability is never zero due to the non-zero transmission times.

- We show that simple modifications at the source node of some parameters of the IEEE 802.11 protocol (i.e.. the contention window CW_{min}) can efficiently be applied to stabilize a multi-hop network.

- We design and deploy the first large-scale IEEE 802.11 multi-hop testbed on the EPFL campus, which spans over the six building of the I&C department and is composed of around 60 off-the-shelf wireless routers.

- We design and implement a new hop-by-hop congestion control mechanism at the MAC layer called *EZ-flow*. EZ-flow takes advantage from the broadcast nature of the wireless medium in order to passively derive the queue size at the next-hop. Then each nodes automatically adapts its channel access probability (i.e., its contention window CW_{min}) in order to keep the queue at the next hop stable. Moreover, we show experimentally, analytically, and through simulations that EZ-flow succeeds in dynamically stabilizing the queues of a multi-hop networks.

- We show through an experimental deployment that it is possible to perform the hop-by-hop congestion control at the network layer. This design choice has the advantage of (i) being independent/transparent of the MAC protocol used (e.g., IEEE 802.11) and (ii) decoupling the task of congestion control from the task of contention control.

- We introduce and implemented a novel fairness algorithm called *Explore & Enhance*. It runs at the gateway and finds a fair and achievable allocation vector that maximizes a given notion of utility without any prior knowledge of the capacity region.

- We show how the congestion-control scheme, at the relay nodes, and the fairness algorithm, at the gateway, can be jointly combined in order to form a complete solution that works without requiring network-wide message passing (i.e., a broadcast message is only needed between the gateway and its direct neighbors).

- We observe the non-monotonic relation between the end-to-end throughput and the source rate in IEEE 802.11 multi-hop networks both through simulations and experimental measurements. We propose a mathematical model to capture this evolution and we analytically derive some results concerning the transition points of this non-monotonic curve.

- We show, both through simulations and with a formal analytical proof, that no end-to-end congestion control scheme can be throughput-optimal if it runs above an unmodified IEEE 802.11 MAC layer. This result supports the approach of performing congestion control in a hop-by-hop manner instead of end-to-end for wireless multi-hop networks.

- We compare different state-of-the-art methodology of performing hop-by-hop congestion control and we illustrate the important tradeoff that exists between efficiency (i.e., throughput-optimality) and robustness (i.e., no throughput collapse when the sources operate beyond the network capacity).

Chapter 2

Background

2.1 Wireless Mesh Networks

Wireless Mesh Networks (WMNs) are multi-hop communication networks that consist of wireless nodes organized in a mesh topology as depicted in Figure 2.1. Their purpose is to provide ubiquitous high-speed Internet access to the end-users and to remove the need for an expensive wired infrastructure in the last-mile of the network. In addition to cost-reduction, another key feature of WMNs is their self-organizing property that allows for a rapid deployment of mesh nodes in order to provide communication capabilities in difficult environments such as in emergency situations.

In order to provide a large-scale connectivity, the infrastructure of mesh networks is divided into two different parts: (i) an *access* part that ensures the direct connectivity between a Transit Access Point of the mesh (TAP) and the final user (the TAP may or may not be connected to the wired infrastructure); and (ii) a *backhaul* part that is in charge of transporting the data packets from the TAP serving the user to the Wired Access Point (WAP) that is the mesh gateway connected to

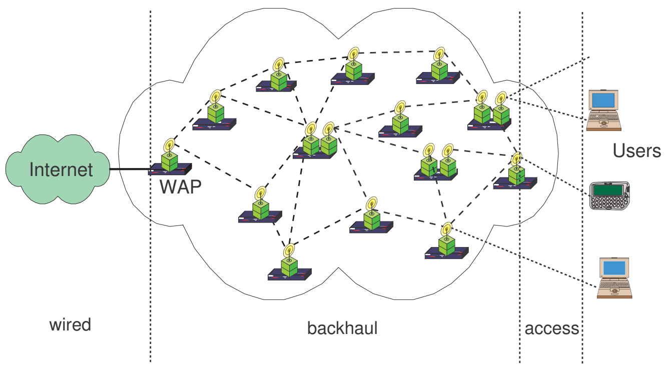


Figure 2.1: A wireless mesh network consists of a backhaul and an access part.

the wired infrastructure. We note that the connectivity problems existing in both the access and backhaul part can be seen as two separate sub-problems by considering that (i) the access points are equipped with two wireless interfaces dedicated to either the access or the backhaul, and (ii) each interface is configured to run in an independent channel. The access part of WMNs will not be the focus of this thesis, as it is similar to the scheduling problem of single-hop WiFi networks that has been heavily studied in the literature [Bia00].

Instead, the backhaul part brings multiple new and interesting challenges due to its multi-hop nature that requires the system to be decentralized and adaptive. Indeed, in order for a self-organizing system to optimally transport data packets in a hop-by-hop manner from a mesh node (TAP) to the mesh gateway (WAP), many technical challenges need to be solved such as spectrum management, scheduling, congestion-control, routing and security.

In this thesis, we focus more specifically on the problems of scheduling and congestion-control. To better understand the nature of these two problems, a useful analogy is found in the vehicular traffic problem, where (i) data packets are seen as cars, (ii) single-hop links are seen as streets, and (iii) intermediate nodes are seen as road intersections controlled by a traffic light.
The scheduling is then similar to a traffic-light problem: *”When should the traffic light at an intersection turn red or green in order to avoid/minimize collisions and to maximize the number of cars going through?”* The challenge in WMNs comes from the shared nature of the wireless medium, which implies that two neighboring nodes cannot transmit (i.e., turn green) simultaneously without creating a collision. Therefore, nodes cannot take the scheduling decision independently from each others, but the lack of central authority (as in traffic-light management) requires the design of efficient distributed stochastic scheme to perform the scheduling decision at each node.
The congestion-control problem takes into account the relation between the different links (i.e., streets) of an end-to-end path and it answers the question, *”How should the traffic light been controlled at the intersections in order to avoid the creation of a traffic jam in any street of the path?”* In WMNs, traffic jams correspond to data packets being queued at intermediate nodes. This is an important problem, because not only does it increase the end-to-end delay of each packet but it can also lead to packet being dropped (i.e., lost) due to the limited hardware size of the buffer at the intermediate nodes. Moreover, in WMNs few or no information is explicitly available concerning the status of the other links of the path that may vary over time. Thus, there is a need to propose some smart and adaptive schemes that can cope with the time-variability of both the links quality, and the traffic demand.

Real deployments of mesh networks already exist in academia [ROO, KZP06], in residential communities [FRE, NAN] and as industrial products [EAR, THE]. These deployments use a free standard technology (i.e., IEEE 802.11) that was not designed for multi-hop communications: this resulted in some interesting experimental findings, such as *”with current commodity wireless technology it does not* *make sense to handle more than three hops”* [1]. In this thesis we do not focus on top-down approaches resulting in clean-slate design, because we want to propose solutions that work on existing deployments with off-the-shelf hardware. Instead, we study analytically the root causes behind the 3-hop boundary of IEEE 802.11 mesh networks and then we follow a bottom-up approach to propose solutions that improve performance and are backward-compatible with existing designs.

2.2 The Layer Model

The challenges in communication networks are usually tackled by dividing the system into different layers, where each layer is responsible for transparently providing some features to the layer above. The first model traditionally proposed is the Open Systems Interconnection model (OSI model), which divides the networking stack into seven independent layers. Most of the current protocols are based on the TCP/IP model that we will use hereafter.

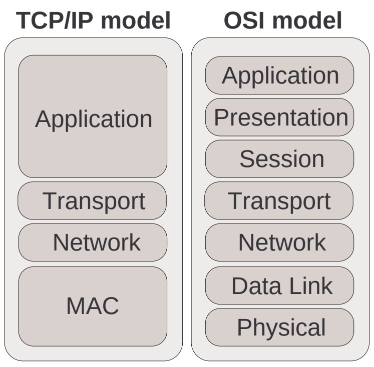


Figure 2.2: The two layer models used in communication systems and their mapping.

In this model, the four layers are:

- Medium Access Control (MAC) layer: It is responsible for transmitting the data to the physical medium. To do so, it decides when to transmit a packet to the next-hop and it verifies its successful reception without collision (e.g., through the use of an acknowledgment scheme). We note that it is not responsible for ensuring that the packet is not dropped at the next-hop after a successful reception (e.g., due to buffer overflow).

- Network layer: Based on the end-to-end destination, it is responsible for making the routing decision and for informing the MAC layer of the identity of the next-hop node. The most commonly used network layer we will consider hereafter is the Internet Protocol (IP).

[1] Lunar project: http://cn.cs.unibas.ch/projects/lunar/

- Transport layer: It is responsible for ensuring the end-to-end connectivity between two hosts. The two standard protocols are: (i) the User Datagram Protocol (UDP), which is connectionless (i.e., best-effort) and is commonly used for voice or video traffic; (ii) the Transmission Control Protocol (TCP), which is connection-oriented and provides reliable in-order packet delivery to the upper-layer by performing end-to-end congestion control.

- Application layer: It is the highest networking layer responsible for receiving and delivering the data to the final application.

In this work, we propose to tackle both the scheduling and congestion-control problem by focusing on the MAC and network layers. Moreover, performing the congestion control at the transport layer provides good performance in the wired Internet, where packet losses are mostly due to buffer overflows and can be seen as a sign of congestion in the network. Wireless multi-hop networks are fundamentally different, because of the variability of the wireless channel that leads to packet losses and high delays. Indeed, TCP communications running on multi-hop wireless links show relatively poor performance with low throughput [GSK04].

Performing congestion control in a hop-by-hop manner instead of end-to-end has been shown to have the potential to improve the performance in a wireless multi-hop network [YS07]. Therefore, an interesting alternative for the backhaul of wireless mesh networks is to move the congestion-control feature from the transport to the network layer. An additional advantage of such a change is that it also covers the cases of voice and video traffic that are typical in emergency situations and that run over the UDP protocol. For the case of data communication requiring to keep the in-order reliable delivery of TCP without congestion-control, new transport or application protocols can be designed. Therefore, as the in-order reliable delivery of the transport layer is out-of scope of this thesis, we mostly base our study on UDP traffic that requires us to provide congestion control at a lower layer.

2.3 The IEEE 802.11 Protocol

The set of standards in the IEEE 802.11 family includes different modulation techniques that operate in the 2.4 GHz and 5 GHz frequency bands and that use the same basic protocol [IEE99]. The first release includes the IEEE 802.11b and the IEEE 802.11a protocols. The IEEE 802.11b protocol runs on the 2.4 GHz band by using Direct-Sequence Spread Spectrum modulation (DSSS) and it provides data rates up to 11 Mb/s. This protocol divides the frequency into 13 channels of 22 MHz, which are spaced 5 MHz apart, thus making only up to three orthogonal channels. The IEEE 802.11a protocol runs on the 5 GHz band by using Orthogonal Frequency-Division Multiplexing (OFDM) and it provides data rates up to 54 Mb/s. After that, new flavors of the IEEE 802.11 protocol appeared; they run on one of these two frequency bands and increase the achievable data rate by using

Multiple Input Multiple Output (MIMO) techniques. The common point among all the variations of the IEEE 802.11 family is the main scheduling protocol based on Carrier Sense Multiple Access (CSMA) to decide when to transmit a packet. In the subsequent section we describe the key components of the IEEE 802.11 protocol and we point to [IEE99] for the technical details.

2.3.1 Description in Single-Hop Environments

The IEEE 802.11 protocol was originally designed for single-hop communications, where the only existing problem is contention (i.e., how to efficiently avoid collisions) and not congestion (i.e., how to avoid buffer overflow).
In order to solve the contention problem, a node i that has a packet to send starts the procedure by uniformly selecting a backoff value β_i in the interval $[0; cw_i - 1]$, where the contention window cw_i starts with the minimal value CW_{min} ($CW_{min} = 2^5$ for 802.11b and 2^4 for 802.11a/g). Then, node i monitors the medium to assess whether communications take place or not. For each idle time slot (consisting of $20\mu s$ for 802.11b and $9\mu s$ for 802.11a/g), the backoff β_i is decremented by one, otherwise it remains frozen if the medium is busy. Eventually, when the backoff reaches zero, node i sends its packet over the wireless medium and it verifies the successful transmission of the packet to the destination through an acknowledgment scheme in which the destination sends an ACK packet after the error-free reception of a data packet.
In the case of a successful transmission, the exactly same procedure is repeated for the next packet in the queue of node i. However, if an ACK is not received at node i, the data packet is considered as lost due to contention (i.e., collisions) and thus the contention window cw_i is doubled before uniformly picking β_i. An unsuccessful transmission is repeated at most seven times and at each time, cw_i is doubled until it reaches the maximal value $CW_{max} = 2^{10}$. If the packet is not successfully received after the 7^{th} trial, this packet is dropped and the system starts the process again with $cw_i = CW_{min}$ for the next packet in the queue of node i. Due to the exponential backoff of IEEE 802.11, this contention-avoidance scheme has been shown to deliver long-term fairness between the nodes, but not short-term fairness.

Nevertheless, the assumption behind the above process is that all the nodes can correctly detect when the medium is idle or busy (i.e., all nodes are within the same collision domain) and therefore collisions only occur when at least two nodes randomly pick the same backoff β. Even in single-hop communications, this assumption can be violated if two clients are in the transmission range of the gateway but not in the sensing range of each other (e.g., due to walls). This situation, known as the *hidden-node terminal*, leads to a serious performance degradation in CSMA schemes due to repeated collisions between the clients that cannot detect transmissions from each other. The cause of this problem is that the channel condition seen at the receiver is not known by the sender, and the solution in single-hop scenarios is to add a collision avoidance scheme. In IEEE 802.11, it is implemented through the optional use of Request-To-Send (RTS) and Clear-To-Send (CTS) messages.

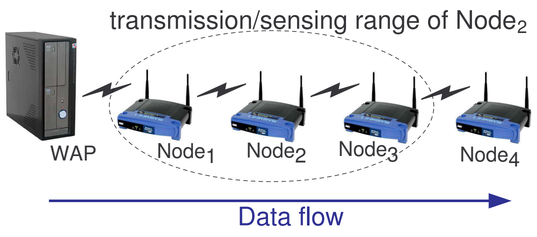


Figure 2.3: Simple 4-hop linear scenario leading to problems with IEEE 802.11.

These messages inform all the nodes in the neighborhood of the source and the destination of the duration of the data exchange. With RTS/CTS, a node starts by sending a RTS message when its backoff expires and it waits to receive a CTS from the destination before sending the data packet. In the meanwhile, all the nodes that hear the RTS or CTS set their Network Allocation Vector (NAV), which ensures that they remain silent for the entire duration of the transmission. Despite its collision avoidance property, the RTS/CTS mechanism also leads to message overhead and performance losses therefore it is rarely used in practice (i.e., RTS/CTS is turned off by default).

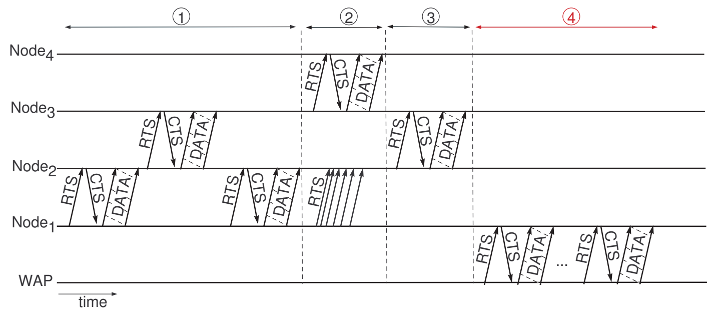
2.3.2 Challenges in Multi-Hop Environments

The IEEE 802.11 protocol was designed for single-hop communications, even though it is currently heavily used in multi-hop scenarios such as in mesh networks. Unfortunately, there are fundamental differences between a single-hop and multi-hop environment, which make the protocol behave poorly in the latter case. Below, we discuss two features of the protocol that lead to poor performance in a simple linear 4-hop scenario as the one depicted in Figure 2.3. This problem takes the form of network instability where the queue of the first relay builds-up indefinitely.

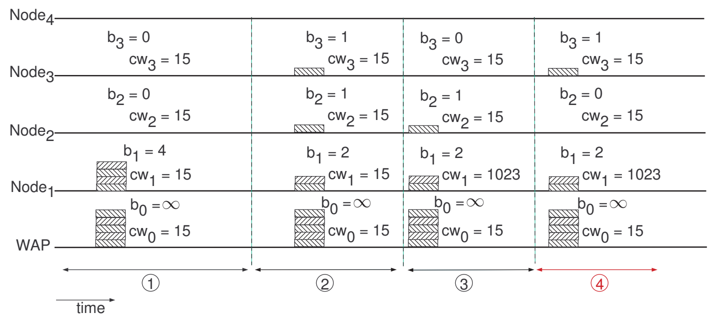
Exponential Backoff and Collision Avoidance

In [AKT08], we show that the CSMA/CA mechanism as it is implemented in IEEE 802.11 leads to poor performance, because nodes may be silent upon the reception of an RTS, even though no complete data exchange takes actually place.

To better understand this problem, we illustrate it through an example that builds upon four phases. Figure 2.4(a) depicts the transmissions as a function of the time, whereas Figure 2.4(b) shows the corresponding queues (b_1, b_2, b_3) and the values of the contention window (cw_0, cw_1, cw_2, cw_3) for the topology depicted in Figure 2.3. We assume that the WAP always has traffic to send, so that its buffer is full ($b_0 = \infty$), and we start with node 1 having already 4 packets buffered. The 4-phase scenario leading to the build-up of node 1 is then:



(a) Link activity (ACK messages omitted).



(b) Queue size and *cw* evolution at the beginning of each phase.

Figure 2.4: Illustration of the perturbation creation due to the exponential backoff of the MAC IEEE 802.11.

1. **Phase 1:** Packets are sent from node 1 to node 2 and node 3. At the end of this phase, each queue contains at least one packet.

2. **Phase 2:** Node 3 transmits a packet to node 4. Node 1 is out of the sensing range of node 3: it is therefore unaware of this transmission, and sends unsuccessful RTS. These RTS messages make the WAP set its NAV properly, and increase the contention window cw_1 up to its maximal value $CW_{max} = 2^{10}$.

3. **Phase 3:** Node 2 transmits a packet to node 3. As the WAP is unaware of this transmission, its backoff counter is not frozen and will eventually reach zero. But, the NAV of node 1 is set by the RTS, which prevents it from decrementing its contention window. Therefore, the contention window of node 1 remains at a high value (around CW_{max}).

4. **Phase 4:** The transmission of node 2 terminates. Node 1 and the WAP still have packets to send and thus compete for the channel. However their competition is not fair, because the contention window of node 1 is much larger than that of the WAP ($\sim 2^{10}$ compared to 2^5 for 802.11b (or 2^4 for

802.11a) in our example, a ratio factor of 32 (or even 64)!). This unfair advantage implies that the WAP will win the competition for the channel many times in a row. As a result, the queue of node 1 builds up.

This example shows that the exponential backoff with RTS/CTS as implemented in IEEE 802.11 exacerbates the instability problem in multi-hop networks. However, we stress that these mechanisms are not the root cause of instability. Indeed, in Chapter 3 we analytically prove that the instability problem already occurs in more fundamental schemes such as CSMA.

2.3.3 IEEE 802.11s

In July 2004, a Task Group was created to develop a new amendment to the IEEE 802.11 standard (called IEEE 802.11s) in order to address the challenges relative to the multi-hop environment of mesh networks. The goal of the IEEE 802.11s protocol is to perform routing at the MAC layer and also to bring security and congestion control to mesh networks.

In order to deal with congestion control, the task group proposes to use an explicit congestion notification message that is sent by a node to its neighbors so that they can adapt their sending rate. In Chapter 5, we will show that it is possible to eliminate these explicit notification messages by taking advantage of the broadcast nature of the wireless medium.

2.4 Desired Properties of Mesh Networks

Having described the goals, the architecture, and the challenges of IEEE 802.11 wireless mesh networks, we now clearly state the system properties that are required for the large-scale adoption of WMNs.

- **High Throughput:** Mesh networks need to be efficient and thus they should be able to deliver an end-to-end throughput that is as close as possible to the theoretical network capacity. To achieve this, WMNs need to avoid suffering from the typical sources of throughput degradation such as packet collisions and buffer overflows.

- **Low Delays:** Another important metric is the end-to-end delay that has to be maintained as low as possible in order to support real-time services such as voice traffic and video on demand. To reach this objective, the network needs to be stable with small queues at each of the relay nodes.

- **Fairness:** The two previous criterion focus on the performance within a single flow. In a real mesh deployment, there are typically multiple flows concurrently present in the network. Therefore it is important that the network delivers a certain level of fairness in order to avoid the complete starvation of certain flows.

- **Adaptability:** Wireless mesh networks have to cope with two sources of variability. The first source comes from the variability of the traffic matrix with the dynamic arrival or departure of new sources in the network. The second is due to the intrinsic variability of the shared wireless medium that is vulnerable to a large number of environmental changes (such as people passing by, electronic devices turned on and doors being opened/closed).

- **Robustness:** As already discussed in Chapter 1, the capacity of a wireless network significantly differs from a wired network, because it is usually unknown and difficult to measure due to the time-variability. Therefore it is important that the network automatically adapts and that the performance does not degrade (or even collapse) when the sources receive packets at a rate above the network capacity.

Chapter 3

Modeling the Instability of Mesh Networks

3.1 Background

3.1.1 Problem Statement

We consider a linear topology such as the one in the backbone of mesh networks, where each node hears only its direct neighbors (cf. Figure 3.1). A common scenario in these networks occurs when the gateway, the WAP, needs to send packets to an end-user that is beyond its direct coverage range. To achieve this goal, the packets transit through the wireless backbone, forwarded by multiple TAPs. As the backbone part is a key element of any WMN, we are interested in optimizing its performance as far as throughput and delay are concerned.

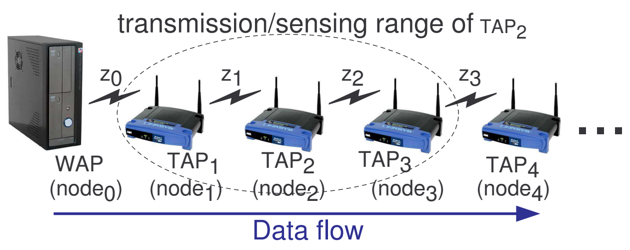


Figure 3.1: Linear topology of a WMN backbone.

In this setting we show through measurements on a testbed, an unexpected behavior of IEEE 802.11 multi-hop networks, which is the striking difference in stability between 3-hop and 4-hop networks. Our testbed is composed of two laptops that act as the source and sink of the traffic and five wireless routers that act as the WAP (node 0) and TAP_i (node i with $1 \leq i \leq 4$). Each laptop runs on Linux with

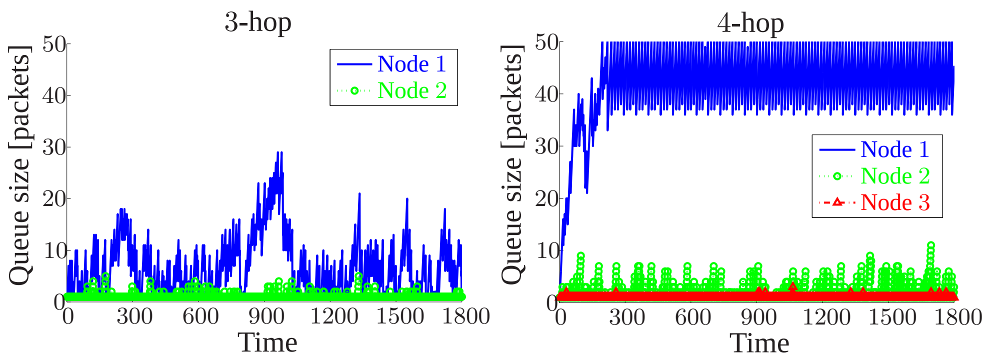


Figure 3.2: Experimental results for the queue evolution of each relay node in 3-hop and 4-hop topologies. A time slot corresponds to an event when the queue size is recorded, that is every time a packet arrives at a node.

the software *Iperf*[1] used to generate saturated UDP traffic with a payload size of 1470 bytes. Each laptop is then connected through a wired cable to either the WAP or the last TAP (i.e., node 4). The wireless routers are Asus WL-500gP running the version *Kamikaze 7.07* of the OpenWRT firmware[2]. We change the mini-PCI WiFi cards to Atheros cards in order to benefit from the flexibility of the MadWifi driver [MAD]. This allows the modifications of the driver source code to perform both queue monitoring and the modification of the contention window. We then set the routers to run in ad-hoc mode on channel 13 of IEEE 802.11b at the data rate of 1Mb/s and without RTS/CTS. To avoid interference from neighboring networks, we perform our measurements in the basement of the BC building at EPFL, where no other wireless networks can be sensed. Finally, we set our topology to match our theoretical study: direct neighbors can communicate together, but nodes separated by two hops or more cannot hear each other.

The major finding is the drastically different stability behavior of 3-hop and 4-hop topologies, which appears to be counter-intuitive. Figure 3.2 shows that the 3-hop topology is stable, but the 4-hop network is unstable. Furthermore, the 4-hop instability is due to node 1, whose queue length exhibits a transient behavior, i.e., which grows indefinitely until it reaches the hardware limit (50 packets for our routers).

In this chapter we seek to better understand the root causes behind this experimental stability result. Toward this goal, we introduce an analytical model that is inspired from the behavior of CSMA/CA protocols (e.g., 802.11-like protocols) with some necessary simplifications for the sake of tractability. We emphasize that, given the mathematical assumptions, our analysis is exact.

[1]Iperf - The TCP/UDP bandwidth measurement tool: http://dast.nlanr.net/Projects/Iperf/
[2]OpenWRT firmware: http://openwrt.org/

3.1.2 Related Work

The unarguable success of the IEEE 802.11 [IEE99] protocol in WiFi communications has lead to the current development of a new draft focusing on multi-hop networks such as WMNs, 802.11s [CK, DBvdVH08]. However, until the release of 802.11s, 802.11b/g remains the standard and it is therefore essential to understand its behavior. Towards this goal, previous works [AKT08, NL07, DBvdVH08] present drawbacks of the current protocol in a multi-hop environment. In [GSK08], Garetto et al. present a model to derive the throughput of flows in a multi-hop network. Furthermore, Ng et al. identify the existence of an optimal offered load and propose source-rate limiting at the application layer as a solution [NL07]. Our approach differs in the sense that we introduce an analytical model that focuses on queue stability and therefore gives insight into the existence of a maximal feasible load. Furthermore our stabilization strategy uses solely the MAC layer and therefore does not impose any limiting requirements on the client side.

Tackling the congestion problem at the transport layer, e.g. TCP, is studied in [SGM$^+$08, RJJP08]. Shi et al. focus on inter-flow competition by studying the starvation occurring when a one-hop flow competes with a two-hop flow and propose a counter-starvation technique that solves the problem. Similarly Rangwala et al. propose another rate-control protocol that achieves better fairness and efficiency than TCP. Our work differs as we focus on the link competition that takes place within a single flow and study the factors leading to the transition from stability to instability. Thus, rather than relying on the transport layer, we throttle congestion at the MAC (link) layer (in the OSI model, this functionality of the link layer is referred to as flow control [GK80]). Analytical and simulation results showing that a hop-by-hop congestion algorithm outperforms an end-to-end version are presented in [YS07] by Yi et al. Their findings reinforce the need to implement congestion control at the MAC layer.

In [LE99] Luo et al. study the system stability of random access protocols in single-hop settings. Our work goes further by analyzing the multi-hop scenario where the queue states of successive nodes are dependent.

The distributed scheduling problem, which aims at ensuring stability and maximal throughput, has witnessed growing interest in the research community. The seminal work of Tassiulas [TE92] introduces a back-pressure algorithm that uses global network queue information to derive an optimal routing/scheduling policy and achieve stability and maximal throughput. Several extensions of this work have been conducted, e.g., Ying et al. reduce the number of queues maintained at each node to enhance scalability [YST08]. Further work on throughput and fairness guarantee can be found in [CKLS08], where Chapokar et al. introduce a distributed scheduling strategy that attains a guaranteed ratio of the maximal throughput. A more complete review of the stability problem in scheduling is presented by Yi et al. in [YPC08], where the tradeoff between complexity, utility and delay is discussed in depth. Finally a scheduling policy based on the queue length is presented and studied analytically by Gupta et al. in [GLS07]. These works propose conceptual scheduling solutions that keep the network stable, but depart from IEEE 802.11 protocols to various extents, and for which no practical implementation exists to date. Our work differs from this previous body of work, as we focus on the stability of *existing* CSMA protocols, e.g. IEEE 802.11. To the best of our knowledge, we are the first to identify the key factors (network size and stealing effect) that affect the network stability. Furthermore, following our analytical study, we develop a practical stabilization strategy and validate experimentally our results with off-the-shelf hardware.

3.2 Analytical Model

3.2.1 MAC Layer Description

The first common assumption [CKLS08, ES05, LE99, TE92, YST08] is that of a slotted discrete-time axis, in other words, each transmission takes one time slot and all the transmissions occurring during a given slot start and finish at the same time. We consider a greedy source model, i.e., the WAP (gateway) always has new packets ready for transmission. Assuming a K-hop system, the packets flow from the WAP to TAP_K, via TAP_1, TAP_2, $\ldots$, TAP_{K-1}. TAPs do not generate packets of their own. Each TAP is equipped with an infinite buffer.

We assume that the system evolves according to a two-phase mechanism: a *link competition phase* and a *transmission phase*. The link competition phase, whose length is assumed to be negligible, occurs at the beginning of each slot. During this phase, all the nodes with a non-empty queue compete for the channel and a pattern of successful transmissions emerges, referred to as *transmission pattern* in this chapter. Given the current state of queues, the link competition process is assumed to be independent of competitions that happened in previous slots. This assumption is similar to the commonly used assumption of exponentially (memoryless) distributed backoffs. During this phase, non-empty nodes are sequentially chosen at random and added to the transmission pattern if and only if they do not interfere with already selected communications (with the notable exception of the *stealing effect* described in Section 3.2.3). The final pattern is obtained when no more nodes can be added without interfering with the others.

The second phase of the model is fairly straightforward as it consists in applying the transmission pattern from the previous phase in order to update the queue status of the system. This queue status information is of utmost importance for our analysis because it is the parameter that indicates whether the network remains stable (no queue explodes) or suffers congestion (one or more queues build up).

3.2.2 Discrete Markov Chain Model

We now formalize the model previously described mathematically. All packets are generated by the WAP (node 0), and are forwarded to the last TAP (node K) by successive transmissions via the intermediate nodes (TAPs) 1 to $K-1$. A time step

$n \in \mathbb{N}$ corresponds to the successful transmission of a packet from some node i to its neighbor $i+1$, or if K is large enough, of a set of packets from different non-interfering nodes $i, j, \ldots$ to nodes $i+1, j+1, \ldots$, provided these transmissions overlap in time (the transmitters and receivers must therefore not interfere with each other). We assume that node 0 always has packets to transmit (infinite queue), and that node K consumes immediately the packets, as it is the exit point of the backbone (its queue is always 0). We are interested in the evolution of the queue sizes b_i of relaying nodes $1 \leq i \leq K-1$ over time, and therefore we adopt, as a state variable of the system at time n, the vector

$$\vec{b}(n) = [b_1(n)\; b_2(n)\; \ldots\; b_{K-1}(n)]^T,$$

with T denoting transposition. We also introduce a set of K auxiliary binary variables z_i, $0 \leq i \leq K-1$, representing the i^{th} link activity at time slot n: $z_i(n) = 1$ if a packet was successfully transmitted from node i to node $i+1$ during the n^{th} time slot, and $z_i(n) = 0$ otherwise. Observing that

$$b_i(n+1) = b_i(n) + z_{i-1}(n) - z_i(n),$$

we can recast the dynamics of the system as

$$\vec{b}(n+1) = \vec{b}(n) + A * \vec{z}(n) \tag{3.1}$$

where

$$\begin{aligned} \vec{z}(n) &= [z_0(n)\; z_1(n)\; z_2(n)\; \ldots\; z_{K-1}(n)]^T \\ A &= \begin{bmatrix} 1 & -1 & 0 & \ldots & 0 \\ 0 & 1 & -1 & 0 & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & 1 & -1 \end{bmatrix}. \end{aligned}$$

Finally, the activity of a link z_i depends on the queue sizes of all the nodes, which we cast as $z_i = g_i(\vec{b})$ for some random function $g_i(\cdot)$ of the queue size vector, or in vector form as

$$\vec{z}(n) = g(\vec{b}(n)). \tag{3.2}$$

The specification of $g = [g_0, \ldots, g_{K-1}]^T$ is the less straightforward part of the model, as it requires entering in some additional details of the CSMA/CA protocol, which we defer to the next sections. We will first expose it in Section 3.3 for a $K = 3$ hops network, and then move to the larger networks with $K = 4$ and $K \geq 5$ in the subsequent section, as the specification of g comes with some level of complexity as K gets larger. Nevertheless, we can already mention here two simple constraints that g must verify:

1. Node i cannot transmit if its queue is empty, and therefore we have $z_i = g_i(\vec{b}) = 0$ if $b_i = 0$;

2. Nodes that successfully transmit in the same time slot must be at least 2 hops apart, as otherwise the packet from node i would collide at node $i+1$ with the packet from node $i+2$. Hence
$$z_i z_{i+k} = 0 \text{ for } k \in \{-2,-1,1,2\}. \tag{3.3}$$

We observe that (3.1) and (3.2) make the model a discrete-time, irreducible Markov chain. The (in)stability of the network coincides with its (non-)ergodicity.

3.2.3 Stealing Effect Phenomenon

The stealing effect phenomenon is a result of the well-known hidden node problem that occurs in multi-hop topologies. Indeed, the existence of directional multi-hop flows in the backbone of mesh networks, from node 0 to node K may induce unfairness in a way that does not arise in single-hop scenarios. Figure 3.3 illustrates an example where the stealing effect occurs. When node i first enters the link competition phase, node $i+2$ may be unaware of this transmission attempt. Because it senses the medium to be idle, node $i+2$ may therefore start a concurrent transmission to node $i+3$ even though it lost the competition phase (i.e. node $i+2$ selected a larger backoff than node i). As a collision occurs at node $i+1$ (due to the broadcast nature of the wireless medium), node i will experience an unsuccessful transmission, whereas the transmission from node $i+2$ will succeed. We refer to this unfairness artifact as the stealing effect, which differs from the classical capture effect. The latter pertains to packets transmitted to the same destination.

Definition 3.1 (Stealing Effect) *The stealing effect occurs when a node $i+2$ successfully captures the channel from an upstream node i, even though it accesses the medium later. We define p to be the probability of the occurrence of the stealing effect.*

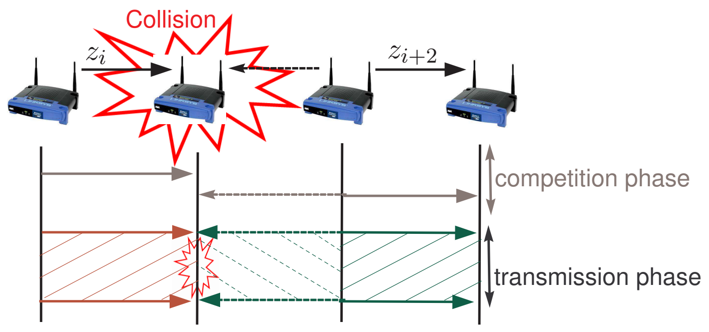


Figure 3.3: Stealing effect scenario.

In IEEE 802.11, the stealing effect corresponds to the event where node $i + 2$ captures the channel, even though it has a larger backoff value than node i. The probability of this event occurring depends on the specific protocol implementation. If the optional RTS/CTS handshake is disabled, then $p \to 1$. If RTS/CTS is enabled, then p is typically much smaller, but still non-zero because RTS messages may collide [SGM$^+$08]. Indeed, the transmission time of a control message (e.g., the RTS transmission time at the 1Mb/s basic rate is $352\mu s$) is non-negligible compared to the duration of a backoff slot ($20\mu s$).

In our model, the stealing effect is captured by having the function $g(\cdot)$ in (3.2) depend on p. As revealed by our analysis, a positive and somewhat counter-intuitive consequence of the stealing effect is the promotion of a laminar packet flow, specifically, a smooth propagation of packets. Indeed, by favoring downstream links over upstream ones, this creates a form of virtual back-pressure that prevents packets from being pushed too quickly into the network.

3.2.4 Stability Definition

A queue is stable when its occupancy does not tend to increase indefinitely. More formally, we adopt the usual definitions of stability (see e.g. Section 2.2 of [BJL08]).

Definition 3.2 (Stability) *A queue is stable when its evolution is ergodic (it goes back to zero almost surely in a finite time). A network is stable when the queues of all forwarding nodes (i.e., all TAPs) are stable.*

3.3 Stability of 3-Hop Networks

Let us first analyze the 3-hop topology, which remains relatively simple because only one link can be active at a given time slot. Indeed, the only three possible transmission patterns are

$$\vec{z} \in \{[1\ 0\ 0]^T, [0\ 1\ 0]^T, [0\ 0\ 1]^T\}.$$

We can now complete the description of the function $g(\cdot)$, before analyzing the ergodicity of the Markov chain.

3.3.1 System Evolution

The role of the stochastic function $g(\cdot)$ is to map a queue status $\vec{b}$ to a transmission pattern $\vec{z}$ with a certain probability.

First, in the case of an idealized CSMA/CA model without the stealing effect ($p = 0$), all non-empty nodes have exactly the same probability of being scheduled. That is, if only node 0 and node 1 (or, respectively, node 2) have a packet to send, both patterns $[1\ \ 0\ \ 0]^T$ and $[0\ \ 1\ \ 0]^T$ (resp., $[0\ \ 0\ \ 1]^T$) happen with a probability

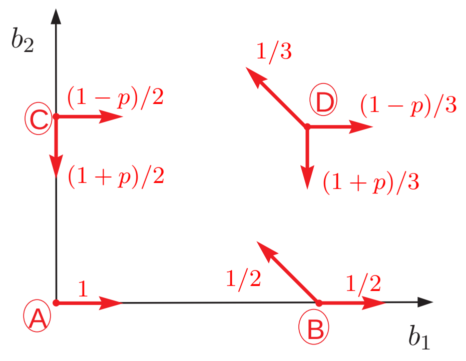


Figure 3.4: Random walk in $\mathbb{N}^2$ modeling the 3-hop network. where the 4 regions are: (A) $\{0; 0\}$, (B) $\{b_1 > 0; 0\}$, (C) $\{0; b_2 > 0\}$ and (D) $\{b_1 > 0; b_2 > 0\}$.

of $1/2$. Similarly, when all three nodes have a packet to send, each of the three possible transmission patterns happens with a probability of $1/3$.

More generally, when we include the stealing effect, we capture the bias towards downstream links that are two hops away. When only node 0 and node 1 compete for the channel, nothing is changed and the probability of success remains $1/2$ as they are only separated by one single hop. However, when node 0 and node 2 compete together, there is a probability p that node 2 steals the channel.

This leads us to define function $g(\cdot)$ differently for each region of $\mathbb{N}^2$ as shown in Figure 3.4. First, in region $A = \{b_1(n) = 0, b_2(n) = 0\}$,

$$g([b_1(n)\ b_2(n)]^T) = [1\ 0\ 0]^T .$$

In region $B = \{b_1(n) > 0, b_2(n) = 0\}$ we have that

$$g([b_1(n)\ b_2(n)]^T) = \begin{cases} [1\ 0\ 0]^T & \text{with probability } 1/2 \\ [0\ 1\ 0]^T & \text{with probability } 1/2. \end{cases}$$

In region $C = \{b_1(n) = 0, b_2(n) > 0\}$,

$$g([b_1(n)\ b_2(n)]^T) = \begin{cases} [1\ 0\ 0]^T & \text{with probability } (1-p)/2 \\ [0\ 0\ 1]^T & \text{with probability } (1+p)/2. \end{cases}$$

Finally, in region $D = \{b_1(n) > 0, b_2(n) > 0\}$, all three nodes compete, and node 2 can still steal the channel from node 0, hence

$$g([b_1(n)\ b_2(n)]^T) = \begin{cases} [1\ 0\ 0]^T & \text{with probability } (1-p)/3 \\ [0\ 1\ 0]^T & \text{with probability } 1/3 \\ [0\ 0\ 1]^T & \text{with probability } (1+p)/3. \end{cases}$$

3.3.2 Stability Analysis

The queue evolution from (3.1) is a random walk in $\mathbb{N}^2$, as depicted in Figure 3.4. Theorem 3.1 shows the stabilizing influence of the stealing effect.

Theorem 3.1 *A* 3*-hop network is unstable for the case* $p = 0$ *and it is stable for all* $0 < p \leq 1$.

Proof: The instability of the case $p = 0$ is readily proved with the non-ergodicity theorem (see Appendix A) using the Lyapunov function

$$h(b_1, b_2) = b_1, \tag{3.4}$$

and setting the constants

$$c = d = 1.$$

Indeed, it is easy to note that in this case

$$\mathbb{E}\left[h(\vec{b}(n+1)) \mid h(\vec{b}(n))\right] - h(\vec{b}(n)) \geq 0,$$

is verified for all the three regions of the space satisfying $h(\vec{b}(n) > 1$ (i.e., the regions B, C and D).

Next we prove the stability of the cases $0 < p \leq 1$ by using Foster's theorem (see Appendix) with the Lyapunov function

$$h(b_1, b_2) = b_1^2 + b_2^2 - b_1 b_2,$$

the finite set

$$F = \{0 \leq b_1, b_2 < 5/p\},$$

the function $k = 1$ and the notations

$$\begin{aligned}
\mu_{b_1,b_2}(n) &= \mathbb{E}\left[h(\vec{b}(n+1)) \mid h(\vec{b}(n)) = h(b_1, b_2)\right] \\
\epsilon_{b_1,b_2}(n) &= \mu_{b_1,b_2}(n) - h(b_1, b_2),
\end{aligned}$$

where $\epsilon_{b_1,b_2}(n)$ can be interpreted as the drift of the random walk at time n. Then we verify Foster's theorem for all the three regions of $\mathbb{N}^2 \setminus F$.
After some computations, we find that for Region $B \setminus F$,

$$\begin{aligned}
\epsilon_{b_1,0}(n) &= \frac{1}{2}(b_1+1)^2 + \frac{1}{2}((b_1-1)^2 + 1 - (b_1 - 1)) - b_1^2 \\
&= 2 - b_1(n)/2 < 0.
\end{aligned}$$

Likewise, for region $C \setminus F$, we get

$$\begin{aligned}
\epsilon_{0,b_2}(n) &= \frac{1+p}{2}(b_2-1)^2 + \frac{1-p}{2}(1 + b_2^2 - b_2) - b_2^2 \\
&= 1 - (3+p)b_2(n)/2 < 0.
\end{aligned}$$

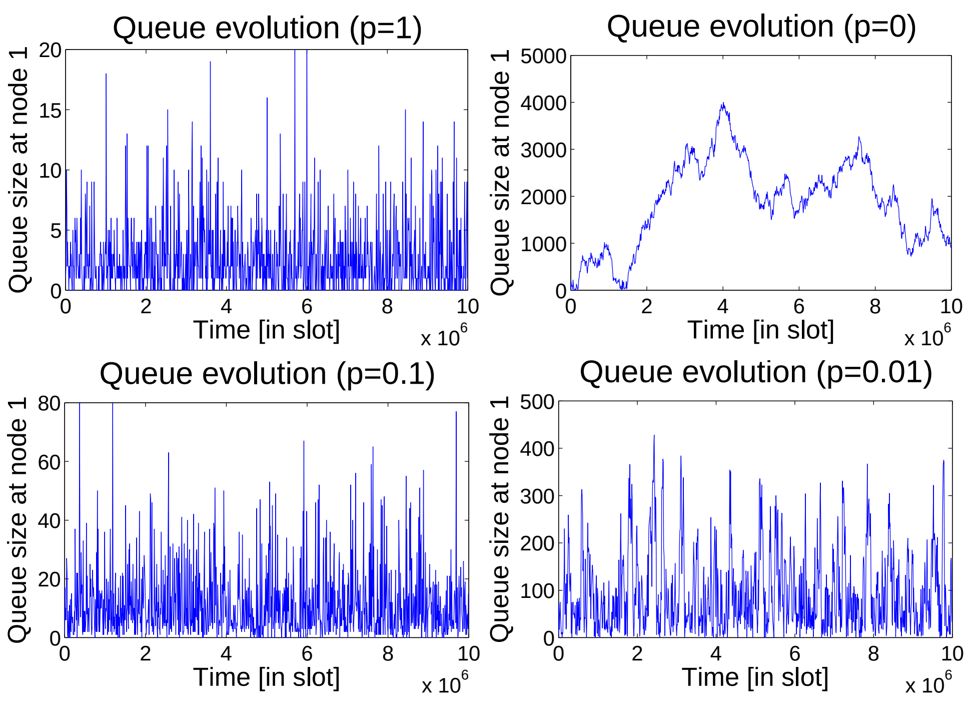


Figure 3.5: Queue evolution for 3-hop with different p values.

Finally, for region $D \setminus F$, we have

$$\begin{aligned}
\epsilon_{b_1,b_2}(n) &= \frac{1-p}{3}((b_1+1)^2+b_2^2-(b_1+1)b_2)+ \\
&\quad \frac{1}{3}((b_1-1)^2+(b_2+1)^2-(b_1-1)(b_2+1))+ \\
&\quad \frac{1+p}{3}(b_1^2+(b_2-1)^2-b_1(b_2-1))+ \\
&\quad b_1b_2-b_1^2-b_2^2 \\
&= \frac{1-p}{3}(1+2b_1-b_2)+\frac{1}{3}(3-3b_1+3b_2)+\frac{1+p}{3}(1+b_1-2b_2) \\
&= 5/3-p(b_1(n)+b_2(n))/3<0.
\end{aligned}$$

Consequently, the two conditions of the theorem are satisfied and stability is proved. $\square$

Finally, we simulate our Markov chain model in MATLAB and in Figure 3.5 we present the effect of p on the queue evolution. We also mention, that our theoretical results give insight into monitoring the queue of node 1 in order to assess the stability of the system (i.e., the Lyapunov function of (3.4) only considers b_1 to prove instability).

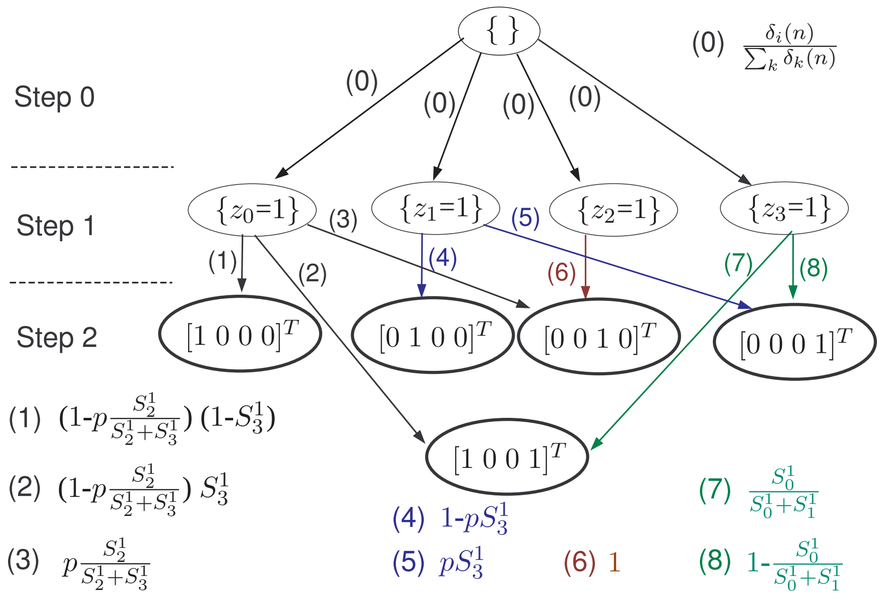

Step 0
Step 1
Step 2
$\{\}$
$\{z_0=1\}$
$\{z_1=1\}$
$\{z_2=1\}$
$\{z_3=1\}$
$[1\ 0\ 0\ 0]^T$
$[0\ 1\ 0\ 0]^T$
$[0\ 0\ 1\ 0]^T$
$[0\ 0\ 0\ 1]^T$
$[1\ 0\ 0\ 1]^T$
(0) $\frac{\delta_i(n)}{\sum_k \delta_k(n)}$
(1) $(1\text{-}p\frac{S_2^1}{S_2^1+S_3^1})\ (1\text{-}S_3^1)$
(2) $(1\text{-}p\frac{S_2^1}{S_2^1+S_3^1})\ S_3^1$
(3) $p\frac{S_2^1}{S_2^1+S_3^1}$
(4) $1\text{-}pS_3^1$
(5) pS_3^1
(6) 1
(7) $\frac{S_0^1}{S_0^1+S_1^1}$
(8) $1\text{-}\frac{S_0^1}{S_0^1+S_1^1}$

Figure 3.6: *Decision tree* to obtain $\vec{z} = g(\vec{b})$ for the 4-hop model.

3.4 Instability of 4-Hop Networks

The 4-hop system is relatively similar to the 3-hop, except that the function $g(\cdot)$ becomes more complex to derive. Indeed the five possible patterns become

$$\vec{z} \in \{[1\ 0\ 0\ 0]^T, [0\ 1\ 0\ 0]^T, [0\ 0\ 1\ 0]^T, [0\ 0\ 0\ 1]^T, [1\ 0\ 0\ 1]^T\}.$$

3.4.1 System Evolution

The drastic difference when moving to 4-hop topologies is that nodes that can transmit concurrently (i.e., node 0 and node 3) will *reinforce* each other and will increase their transmission probability [DT06, Dou07].

This interdependence makes the determination of $g(\cdot)$ less straightforward than in the 3-hop case. We capture this complexity by a *decision tree*, depicted in Figure 3.6, which maps all the sequential events that can occur for the selection of the transmission pattern (one of the states in bold in Figure 3.6).

Before describing the exact mechanisms behind our decision tree, we introduce some necessary notations. First, we define the iteration step m that represents the step between two sequential events (an event corresponds to either the inclusion of a node in the transmission pattern or the removal of a node from the competition). As shown in Figure 3.6, the decision-tree process ends in two iterations ($m \in \{0, 1, 2\}$) and this is due to the fact that at most two links can be active concurrently

in the transmission pattern of a 4-hop network.
Secondly, we introduce the two indicator vectors $\vec{\delta}(n)$ and $\vec{S}^m$. The four entries

$$\delta_i(n) = \mathbb{1}_{\{b_i(n)>0\}}$$

indicate which queues are occupied ($\delta_i(n) = 1$) or empty ($\delta_i(n) = 0$). The vector

$$\vec{S}^m = [S_0^m \ \ldots \ S_3^m]^T ,$$

obtained through an iterative process, indicates the set of nodes that are still in competition for the channel at iteration step m. Initially, all the nodes with a non-empty queue compete for the channel at step 0 and therefore

$$\vec{S}^0 = \vec{\delta}(n).$$

Then the indicator vector at step m, $\vec{S}^m$, is obtained by removing from $\vec{S}^{m-1}$ the node that was selected at iteration step m and its direct neighbors.
For example, if we start from the fully-occupied case

$$\vec{S}^0 = [1\ 1\ 1\ 1]^T$$

and follow the path where node 1 is selected (z_1 is set to 1), the nodes 0, 1 and 2 are removed from the competition and the new indicator vector becomes

$$\vec{S}^1 = [0\ 0\ 0\ 1]^T$$

for this path.

The exact probabilities of each link of the decision tree are denoted in Figure 3.6. The intuition behind these probabilities is that at step m all nodes i that are still competing for the channel (i.e., $S_i^m = 1$) have an equal probability of being selected for transmission. Furthermore, if z_{i-2} is already set to 1 at step m, the selected node i has a probability p of successfully stealing the channel, in which case z_{i-2} is set to 0 and z_i is set to 1 instead. Otherwise, z_i is set to 0.

The computation of the different transmission pattern probabilities (i.e., the determination of the function $g(\cdot)$) is obtained by summing up the *path probability* of each of the paths leading to one of the five possible transmission patterns (state circled in bold in Figure 3.6).
In other words, the probability of the pattern

$$[1\ 0\ 0\ 0]^T$$

is the probability of having z_0 set to 1 at step 0, multiplied by the probability of keeping this selection at step 1 (i.e., no additional active link or stealing effect).
Similarly, the probability of the pattern

$$[0\ 1\ 0\ 0]^T$$

is the probability of having z_1 set to 1 at step 0, multiplied by the probability of keeping this selection at step 1.
Then, the probability of the pattern

$$[0\ 0\ 1\ 0]^T$$

is obtained by adding: (i) the probability of having z_2 set to 1 at step 0, multiplied by the probability of having this selection maintained at step 1 and (ii) the probability of having z_0 set to 1 at step 0, multiplied by the probability of having the stealing effect at step 1.
Likewise, the probability of the pattern

$$[0\ 0\ 0\ 1]^T$$

is obtained by adding: (i) the probability of having z_3 set to 1 at step 0, multiplied by the probability of having this selection maintained at step 1 and (ii) the probability of having z_1 set to 1 at step 0, multiplied by the probability of having the stealing effect at step 1.
Finally, the probability of the pattern

$$[1\ 0\ 0\ 1]^T$$

is obtained by adding: (i) the probability of having z_0 set to 1 at step 0, multiplied by the probability of having z_3 set to 1 at step 1 and (ii) the probability of having z_3 set to 1 at step 0 multiplied by the probability of having z_0 set to 1 at step 1.

As in Figure 3.4, Figure 3.7 summarizes the probabilities of the transmission patterns (i.e., $g(\cdot)$) for each of the 8 regions of $\mathbb{N}^3$:

$$\begin{aligned}
A &= \{0,0,0\}\\
B &= \{b_1(n)>0,0,0\}\\
C &= \{0,b_2(n)>0,0\}\\
D &= \{b_1(n)>0,b_2(n)>0,0\}\\
E &= \{0,0,b_3(n)>0\}\\
F &= \{b_1(n)>0,0,b_3(n)>0\}\\
G &= \{0,b_2(n)>0,b_3(n)>0\}\\
H &= \{b_1(n)>0,b_2(n)>0,b_3(n)>0\}.
\end{aligned}$$

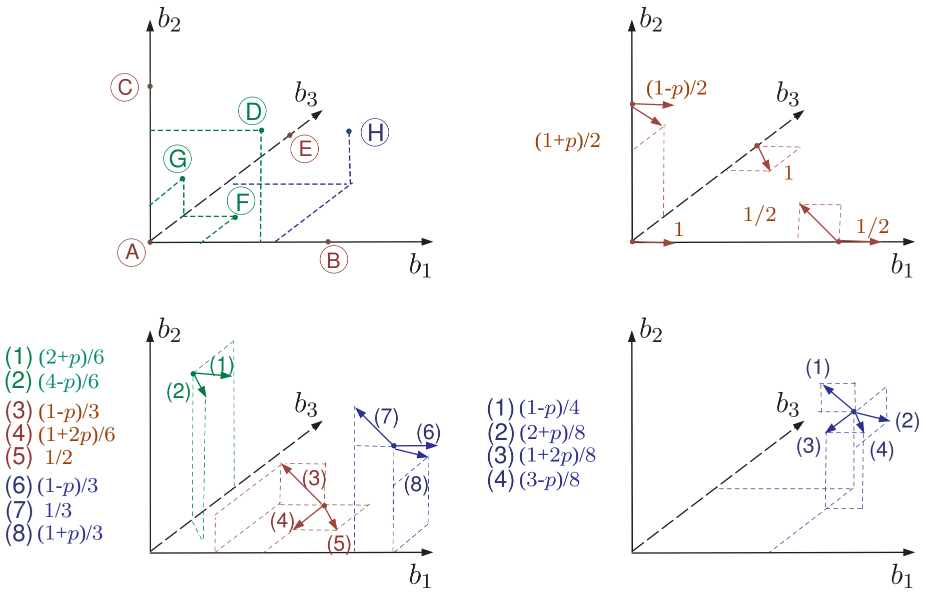


Figure 3.7: Random walk in $\mathbb{N}^3$ for a 4-hop network.

3.4.2 Stability Analysis

Similarly to the 3-hop network, we model the queue evolution by the random walk in $\mathbb{N}^3$ depicted in Figure 3.7. However, contrary to the 3-hop case, the 4-hop case presents a structural factor that makes the system unstable either with or without the stealing effect as stated in Theorem 3.2.

Theorem 3.2 *A 4-hop network is unstable for all* $0 \leq p \leq 1$.

Proof: Starting with $p \neq 1$, we introduce the Lyapunov function

$$h(b_1, b_2, b_3) = b_1 + \frac{p}{1+p} b_3, \tag{3.5}$$

the constants $c = 3$, $d = 1$, $\epsilon = (1-p)/36$ and the function

$$k(i) = \begin{cases} 3 & \text{if } i \in \text{region } B \\ 2 & \text{if } i \in \text{region } D \\ 1 & \text{otherwise} \end{cases}, \tag{3.6}$$

Furthermore we introduce the notation

$$\begin{aligned} \mu_{k,b_1,b_2,b_3}(n) &= \mathbb{E}[h(\vec{b}(n+k))|h(\vec{b}(n) = h(b_1, b_2, b_3))] \\ \epsilon_{k,b_1,b_2,b_3}(n) &= \mu_{k,b_1,b_2,b_3}(n) - h(b_1, b_2, b_3), \end{aligned}$$

where $\epsilon_{k,b_1,b_2,b_3}(n)$ is the drift of the k-step random walk. Having

$$S_c = \{\vec{i} : h(\vec{i}) > c\},$$

we verify condition 2 of the transience theorem (see Appendix A) in all regions of S_c, starting with the regions having $k(i) = 1$.
For region $A \cap S_c$, we obtain

$$\epsilon_{1,0,0,0} = 1 \geq \epsilon.$$

For region $C \cap S_c$, we obtain

$$\epsilon_{1,0,b_2,0} = \frac{1-p}{2} + \frac{1+p}{2}\frac{p}{1+p} = \frac{1}{2} \geq \epsilon.$$

For region $E \cap S_c$, we obtain

$$\epsilon_{1,0,0,b_3} = 1 - \frac{p}{1+p} = \frac{1}{1+p} \geq \epsilon$$

For region $F \cap S_c$, we obtain

$$\begin{aligned}\epsilon_{1,b_1,0,b_3} &= -\frac{1-p}{3} - \frac{1+2p}{6}\frac{p}{1+p} + \frac{1}{2}(1 - \frac{p}{1+p}) \\ &= \frac{1-p}{6(1+p)} \geq \epsilon.\end{aligned}$$

For region $G \cap S_c$, we obtain

$$\begin{aligned}\epsilon_{1,0,b_2,b_3} &= \frac{2+p}{6}\frac{p}{1+p} + \frac{4-p}{6}(1 - \frac{p}{1+p}) \\ &= \frac{4+p+p^2}{6(1+p)} \geq \epsilon.\end{aligned}$$

For region $H \cap S_c$, we obtain

$$\begin{aligned}\epsilon_{1,b_1,b_2,b_3} &= -\frac{1-p}{4} + \frac{2+p}{8}\frac{p}{1+p} - \frac{1+2p}{8}\frac{p}{1+p} + \frac{3-p}{8}(1 - \frac{p}{1+p}) \\ &= \frac{p^2+1}{8(1+p)} \geq \epsilon.\end{aligned}$$

Following the same methodology, we obtain after some computations the 2-step drift (i.e., $k(i) = 2$) for region $D \cap S_c$

$$\epsilon_{2,b_1,b_2,0} = \begin{cases} \frac{1-p}{18} \geq \epsilon & \text{for } b_2 = 1 \\ \frac{1-p}{24} + \frac{p^2}{12} \geq \epsilon & \text{otherwise} \end{cases},$$

And the 3-step drift (i.e., $k(i) = 3$) for region $B \cap S_c$

$$\epsilon_{3,b_1,0,0} = \frac{1-p}{36} \geq \epsilon.$$

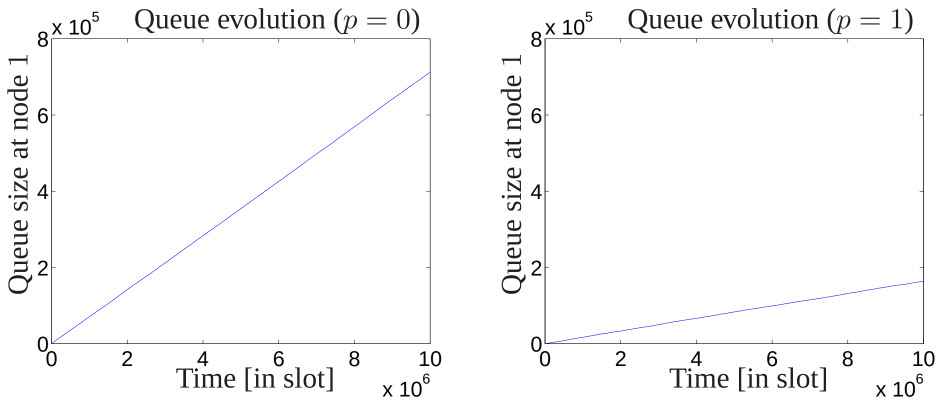


Figure 3.8: Queue evolution for 4-hop with different p values.

Consequently, as conditions 1 and 3 are trivially satisfied, the system is unstable for $p \neq 1$.

In the case $p = 1$, we prove the instability of the network by using the non-ergodicity theorem (see Appendix A) with the Lyapunov function

$$h(b_1, b_2, b_3) = 2b_1 + b_3, \tag{3.7}$$

and setting the constants $c = d = 2$ in that theorem. Indeed, by computing the drift $\epsilon(\vec{b}(n)) = \epsilon_{1,b_1,b_2,b_3}(n)$, we obtain

$$\epsilon(\vec{b}(n)) = \begin{cases} 0 & \text{if } \vec{b}(n) \in \text{region } B, D, F \\ 1 & \text{if } \vec{b}(n) \in \text{region } C, E, G \\ 2/8 & \text{if } \vec{b}(n) \in \text{region } H. \end{cases} \tag{3.8}$$

Therefore, as we have non-negative values for all the regions of the space such that $h(\vec{b}(n)) > c$ and as the drift is upper-bounded by d, we end our proof for $p = 1$ by applying the non-ergodicity theorem.
□

Finally, we present in Figure 3.8 the simulation results showing instability independently of the p value. These results are fundamental for real networks as they reveal the tendency of CSMA to naturally produce instability for 4-hop topologies.

3.4.3 Extension to Larger K-Hop Topologies

In the case without the stealing effect ($p = 0$), we can easily prove the network instability for $K = 2$, as we did in the previous sections for $K = 3, 4$. When $p = 0$, the instability of a K-hop topology for any $K > 4$ follows then from the following lemma.

Lemma 3.1 (K-hop Instability) *If $p = 0$, a sufficient condition for a linear K-hop network to satisfy the conditions of the non-ergodicity theorem and thus to be unstable is that both the $(K-1)$ and $(K-3)$ hop networks satisfy the conditions of the non-ergodicity theorem.*

Proof: Let us denote the next step expectation of a K-hop network by

$$\mu_i^K(n) = \mathbb{E}[h(\vec{b}(n+1)) \mid h(\vec{b}(n)) = h(\vec{i})].$$

Here $h(\vec{b}) = b_1$ and therefore we can write

$$\mu^K(n) = \alpha\mu_0^K(n) + (1-\alpha)\mu_1^K(n) \tag{3.9}$$

where

$$\alpha = \mathbb{P}(z_{K-1}(n) = 0)$$

and

$$\begin{aligned}
\mu_0^K(n) &= \mathbb{E}\left[b_1(n+1) \mid b_1(n) = b_1, z_{K-1}(n) = 0\right] \\
&= \mu^{K-1}(n) \\
\mu_1^K(n) &= \mathbb{E}\left[b_1(n+1) \mid b_1(n) = b_1, z_{K-1}(n) = 1\right] \\
&= \mathbb{E}\left[b_1(n+1) \mid b_1(n) = b_1, z_{K-3}(n) = z_{K-2}(n) = 0\right] \\
&= \mu^{K-3}(n)
\end{aligned}$$

where we have used (3.3) and the independence of $b_i(n+1) - b_i(n)$, $1 \le i \le K-3$, from $b_{K-2}(n)$ and $b_{K-1}(n)$, conditionally to $z_{K-3}(n) = z_{K-2}(n) = 0$. Therefore (3.9) becomes

$$\mu^K(n) = \alpha\mu^{K-1}(n) + (1-\alpha)\mu^{K-3}(n),$$

which implies that $\mu^K(n)$ verifies the inequalities of the non-ergodicity theorem if $\mu^{K-1}(n)$ and $\mu^{K-3}(n)$ do.
□

This structural instability of the MAC layer for linear topologies is of utmost importance for the design of future wireless mesh networks. Indeed, queue instability means delay and packet drops. These artifacts are undesirable and a mechanism to prevent instability should be designed. Fortunately, some simple *static stabilization strategies* are implementable with off-the-shelf hardware, as described in the next section.

3.5 Static Stabilization Strategy

3.5.1 Source Throttling

In the previous sections, we showed that system instability is caused by the node next to the source (i.e., node 1). The main reason for this problem is that, over the

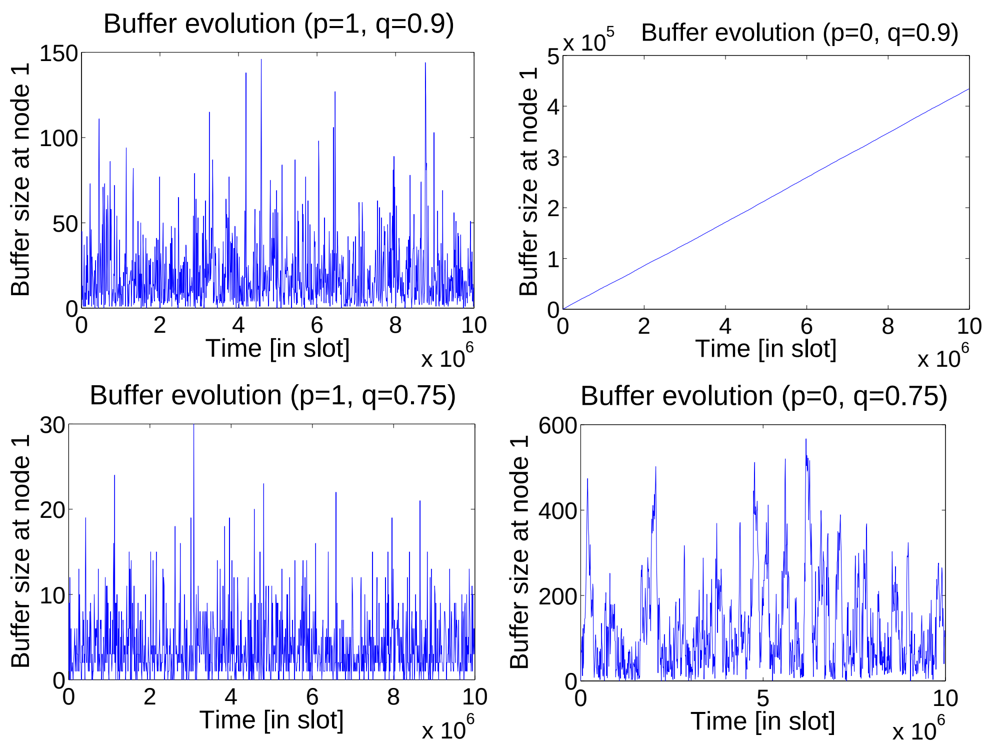


Figure 3.9: Queue evolution for 4-hop with different p and q values.

long run, node 0 is given more chances than node 1 to successfully transmit, which results in a queue build-up at node 1.

Our MAC layer solution, called *penalty strategy*, works by reducing the odds of channel access by node 0. This strategy, besides its useful stabilization property, is easily implementable with off-the-shelf hardware. For instance, the odds of channel access for an IEEE 802.11 node can be affected using the minimum contention window parameter (CW_{min}). In particular, the higher the CW_{min}, the lower the odds of channel access. Therefore our penalty strategy can be deployed by setting node 0 with a higher CW_{min} (i.e., cw_0) than the other nodes (i.e., cw_i for $i > 0$).

3.5.2 Penalty Model

We model the penalty strategy by introducing a *throttling factor* $q \in [0, 1]$ that shows the degree at which the input rate is throttled at the source. That is, $q = 1$ means the input rate is not constrained and then node 0 is not penalized (similar to our previous model). On the contrary, $q = 0$ means node 0 is completely starved and the input rate is null.

A useful analogy to this strategy is a water pipe with the tap aperture being q.

The higher the value of q, the higher the inflow into the network, but also the more likely the flow will be turbulent. Mapping this back to our system, we deduce that there exists a throttling factor threshold q^* below which the system is stable.

The impact of the throttling factor is captured in our analytical model by choosing node 0 with probability

$$\frac{qS_0^m}{qS_0^m + \sum_{j>0} S_j^m},$$

whereas node $i \neq 0$ is selected with probability

$$\frac{S_i^m}{qS_0^m + \sum_{j>0} S_j^m}.$$

For instance, consider a K-hop network where at a given time slot only nodes 0 and 1 have packets to transmit: then node 0 wins with probability

$$\mathbb{P}(z_0) = q/(1+q)$$

and node 1 wins with complementary probability

$$\mathbb{P}(z_1) = 1/(1+q).$$

3.5.3 Theoretical Analysis

Next we analyze 3-hop and 4-hop networks, after implementing the penalization strategy.

Beginning with the previously unstable 3-hop case ($p = 0$), we prove the stabilization effect of the throttling factor and the existence of the threshold $q^* = 1$ by applying a similar methodology than in Section 3.3.2. That is, using the function

$$h(b_1, b_2) = b_1^2 + b_2^2,$$

and the finite set

$$F = \left\{0 \leq b_1 < \frac{3+q}{2(1-q)}\right\} \bigcap \left\{0 \leq b_2 < \frac{1+q}{2}\right\},$$

we verify Foster's theorem (see Appendix A) and therefore prove the stability of the 3-hop case for all $q < 1$.

Next, we consider 4-hop topologies. We observe that the stealing effect has an important impact on stability. Indeed, as depicted on Figure 3.9, the threshold q^* is highly dependent on the stealing effect probability p. The higher p, the higher q^*.

We derive theoretical bounds on q^* as follows. First, we obtain a lower-bound through a stability argument. The proof is obtained by a generalized version of Foster's theorem (Theorem 2.2.2, [FMM95]) that uses k-step expectation instead of 1-step expectation. We apply it with the Lyapunov function

$$h = b_1 + b_2 + b_3,$$

taking $k = 18$.
The upper-bound is similarly obtained by the non-ergodicity theorem (see Appendix A) with the function

$$h = b_1 + \frac{p}{1+p} b_3$$

and the parameter $k = 15$.
Following this procedure for $p = 0$, we obtain

$$0.37 < q^* \leq 0.884.$$

Similarly, when $p = 1$, we find

$$0.76 < q^* \leq 0.964.$$

It should be noticed that the bounds on q^* could be made tighter by either investigating different Lyapunov functions or by increasing the value k (i.e., increasing the number of steps of the k-steps expectation at the expense of higher computational cost).
Nevertheless, the current bounds already validate some results of Figure 3.9. For instance, $q = 0.75 < 0.76$ is stable for $p = 1$ and $q = 0.9 > 0.884$ is unstable for $p = 0$.

3.5.4 Experimental Analysis

We test our stabilization strategy on our experimental testbed described in Section 3.1. We implement the factor q in our testbed by setting a different value of CW_{min} at the WAP (i.e., cw_0) than at the other nodes (i.e., $cw_{i>0}$). Thus, to implement $q = 1/2$, we use $cw_0 = 2^6$ and $cw_{i>0} = 2^5$. Note that our hardware forced us to set the contention window as a power of 2. The results of Figure 3.10 confirm experimentally the efficiency of the throttling factor in stabilizing the network. Indeed, a value of $q = 1/2$ suffices to change the network behavior from unstable to stable.

Table 3.1 presents mean throughput statistics for different network sizes and values of q. The results are obtained by running experiments for 400 s and measuring throughput over non-overlapping 5 s intervals. Each entry in the table thus represents an average of 80 samples. Corresponding standard deviations are also provided. We make the following observations. First, there is a significant throughput gap between 3-hop and 4-hop networks, which can be attributed to the distributed nature of IEEE 802.11 that does not always make the best spatial reuse of the channel. Second, for the case of 4-hop networks, the mean throughput decreases with q, but only when q becomes very small (i.e., $q = 1/16$ or smaller). Up to that point, the overhead caused by using a larger contention window for the WAP is not too significant.

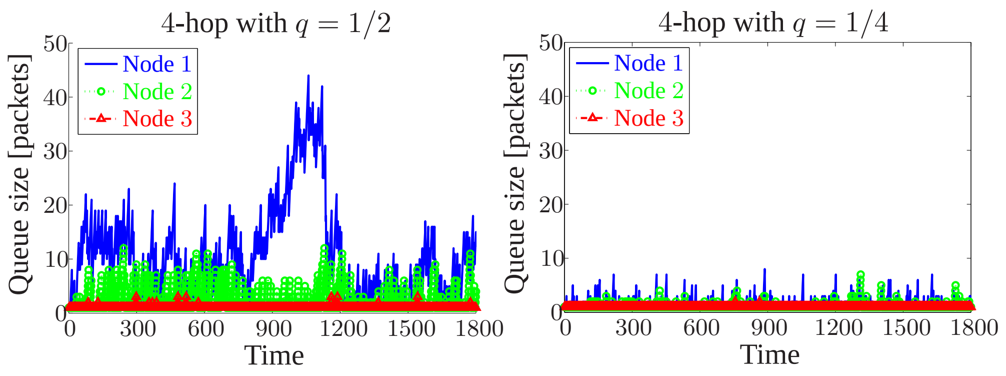


Figure 3.10: Experimental results for the queue evolution for the 4-hop topologies with different throttling factors q (we recall $q = 1$ is unstable).

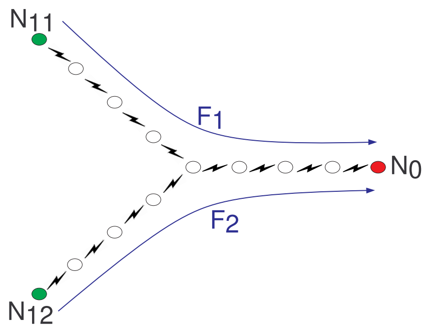
	Mean throughput	Standard deviation
3-hop with $q = 1$	240.5 kb/s	15.8
4-hop with $q = 1$	146.0 kb/s	16.7
4-hop with $q = 1/2$	157.5 kb/s	16.3
4-hop with $q = 1/4$	141.5 kb/s	14.5
4-hop with $q = 1/8$	143.3 kb/s	17.4
4-hop with $q = 1/16$	108.7 kb/s	11.8
4-hop with $q = 1/32$	96.9 kb/s	11.1
4-hop with $q = 1/64$	81.8 kb/s	13.4

Table 3.1: Mean throughput and standard deviation for networks of different sizes and for different throttling factors q.

3.6 Simulations on Multi-Flow Topologies

Up to this point, we have focused on single flow linear topologies as they are the building blocks of more general mesh topologies. However, to show that the stability problem also arises in more complex topologies, in this section we present the simulation results obtained with the ns-2 simulator.

We analyze the two multi-flows topologies depicted in Figure 3.11, where scenario 1 sees two concurrent flows and scenario 2 sees three. Furthermore, we set the simulator to use the standard parameters of IEEE 802.11 ad-hoc networks (RTS/CTS disabled, Tx range: 250 m, Cs range: 550 m) and let the simulations run for respectively 1100 s and 1600 s.



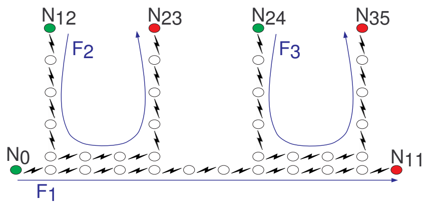


Figure 3.11: Illustration of the two topologies tested by simulation.

The two performance metrics we focus on are: (i) end-to-end delay (low delays means that the network is stable, whereas high delay is a symptom of saturated queues) and (ii) throughput. Figure 3.12 ignores the first 100 s of the simulation and shows the average performance achieved by the network as a function of the throttling factor q. We compute the throughput by measuring the average on disjoint 50 seconds intervals, thus obtaining 20 (30) measurement points. Then we plot the median value with the 95%-confidence intervals.

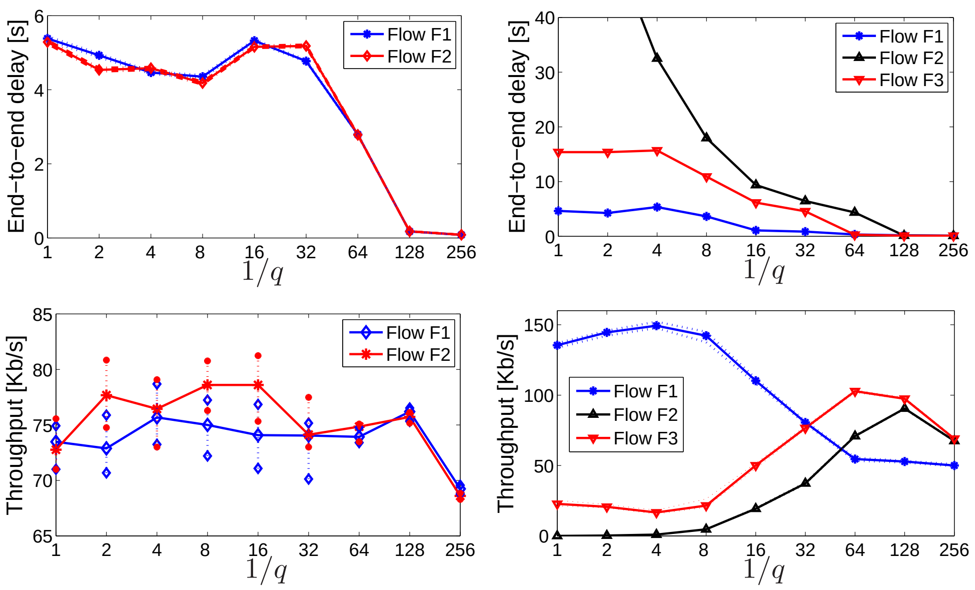


Figure 3.12: Illustration of the evolution of the delay and the median of the averaged throughput (with confidence interval) depending on the throttling factor q for scenario 1 (left) and scenario 2 (right). We note that in both scenarios the value $q = 1/128$ stabilizes the network (i.e. low delay), while achieving a good throughput performance.

We note that, for both scenarios, the standard IEEE 802.11 protocol (i.e. $q = 1$) performs poorly as expected, with high variance in throughput and high end-to-end delays. Furthermore, using an appropriate throttling factor that is larger than for the single-flow case (here $q = 1/128$), performances are significantly improved by achieving both negligible delays and higher global throughput due to a lower packet loss rate (as no buffer overflow in stable regime).

3.7 Instability Problem at Higher Rates

Our analytical model allows us to explain why a stable 3-hop network becomes unstable when a 4^{th} hop is added (see Figure 3.2). Nevertheless, the results from Figure 3.2 are obtained with a fixed data rate of 1 Mb/s, a buffer size limit of 50 packets, and a small-scale testbed where the routers are used without their external antennas (better control on the experimental environment). In order to validate our results on a different setting, we modify the MadWifi driver to unlock the buffer size limit and to allow for the modification of its value at run time through a simple command (see Appendix B for details). We then set the buffer limit to 100 packets and repeat the experiment from Figure 3.2 on the real-scale deployment of Figure 3.13, for different data rate settings.

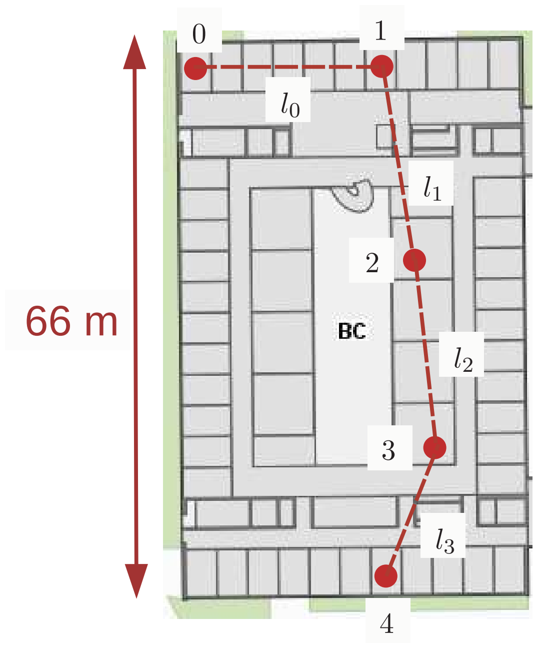


Figure 3.13: Illustration of the deployment used in Section 3.7.

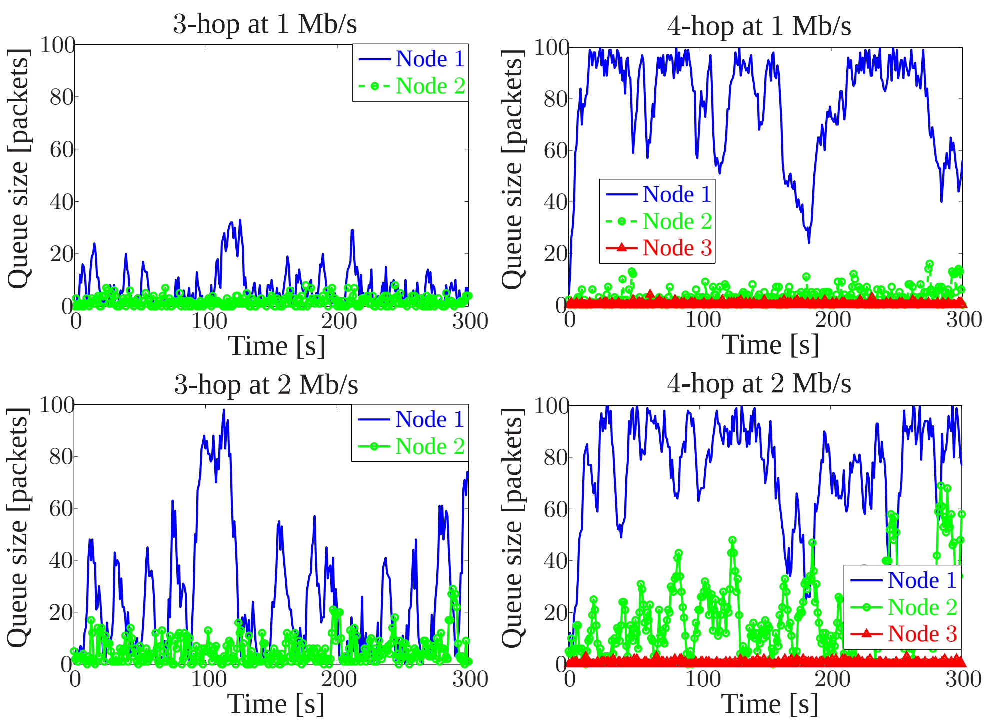


Figure 3.14: Validation of the experimental results from Figure 3.2 on a different setup running at various data rate.

Figure 3.14 and 3.15 show the queue evolution of a 3-hop network (node 0 to 3 in Figure 3.13) and a 4-hop network (node 0 to 4 in Figure 3.13) at data rates of: 1 Mb/s, 2 Mb/s, 11 Mb/s and auto-rate. Additionally, Table 3.2 presents the link throughputs and the end-to-end throughputs achieved at the different data rates.

throughput\rate	1 Mb	2 Mb	11 Mb	auto-rate
l_0	894 kb/s	1.67 Mb/s	6.71 Mb/s	5.79 Mb/s
l_1	858 kb/s	1.52 Mb/s	5.82 Mb/s	2.03 Mb/s
l_2	754 kb/s	1.28 Mb/s	4.23 Mb/s	1.95 Mb/s
l_3	813 kb/s	1.6 Mb/s	5.98 Mb/s	5.49 Mb/s
3-hop	241 kb/s	493 kb/s	1.05 Mb/s	373 kb/s
4-hop	194 kb/s	354 kb/s	791 kb/s	260 kb/s

Table 3.2: Measurements of the links throughput and the end-to-end throughput of a 3-hop and 4-hop linear topology for different data rates.

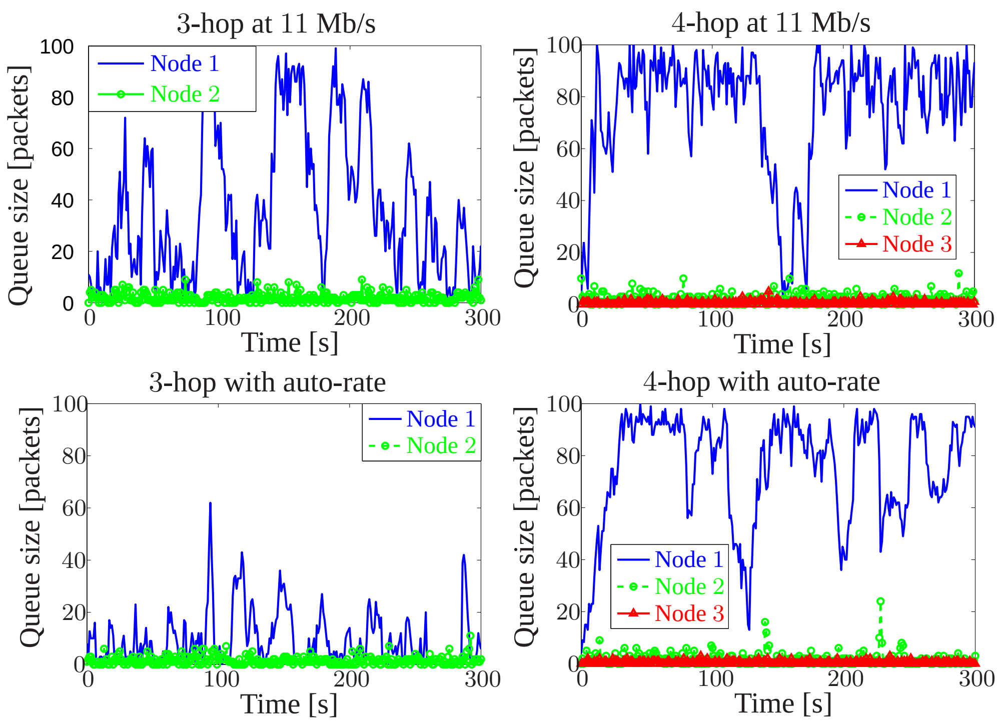


Figure 3.15: Validation of the experimental results from Figure 3.2 on a different setup running at various data rate.

Our results show that even though l_2 is the bottleneck link (i.e., the link delivering the smallest throughput when transmitting alone) for all the data rates, it does not result in network instability in the 3-hop case, due to the stealing effect described in Section 3.2.3. Moreover, the simple addition of a 4^{th} hop turns the network from stable to unstable (i.e., the queue remains close to the buffer limit). We note that the queue size variations are larger than in Figure 3.2. This is because the real-scale deployment is a less controlled environment more prone to changing channel conditions. Nevertheless, we stress that, despite these variations, the change in stability between a 3-hop and 4-hop network is seen for all the different data rates that we tested, as predicted by our analytical model.

Our experimental results at higher rates also provide an interesting finding that is worth mentioning. Indeed, when observing the 3-hop results, we see that the queue variation of node 1 increases at higher rates.
To understand this finding, we need to recall that the transmission duration decreases at higher rates. This means that the period of vulnerability to the stealing effect decreases at higher rates and, therefore the probability of stealing effect p decreases as a function of the data transmission rate.
Finally, once we understand the relation between the data rate and the probability p, we note that our experimental results at higher rates confirm the simulation re-

sults of our analytical model presented in Figure 3.5. In other words, we see that the higher the rate (i.e., the smaller p), the closer the queue evolution gets to a null recurrent system.

3.8 Concluding Remarks

In this chapter, we addressed the problem of network stability in CSMA-based linear wireless mesh networks and we provided three main contributions. First, we identified two key factors impacting the stability: the network size and an artifact that we call the stealing effect. Second, we proved analytically and showed experimentally that 3-hop networks are stable when we account for the stealing effect, but 4-hop networks (and presumably larger topologies) are not. Third, we devised and proved the effectiveness of a simple static stabilization strategy that throttles the source at the MAC layer, preventing packets from being injected too quickly into the network. We note that this strategy penalizes only the source when other nodes also have packets to transmit. This desirable property allows to ensure both stability and high throughput. Our analysis and experiments have shown that selecting a value CW_{min} four to eight times larger at the source than at the relay nodes (i.e., $q = 1/4$ or $q = 1/8$) in the 4-hop case effectively achieves these goals. We also showed that multi-flow networks can also be stabilized by using higher values of q. As the optimal parameter q is topology-dependent, in the next chapters we will present some techniques to automatically adapt the throttling factor q according to the network environment.

Chapter 4

Building an Indoor Wireless Testbed

4.1 Problem Statement

Wireless mesh networks were designed to become a commercial product to deliver high-speed Internet access to regions that were still unserved. In their beginning, in 1994 under the name of *massive array cellular systems*, wireless mesh networks were thought to become a cost-effective alternative to replace the last-mile infrastructure in large metropolitan cities [Pie94]. With the explosion of wired broadband connectivity in developed countries, the business case has evolved to provide access to places where the infrastructure is inexistent or unavailable, such as in developing countries or in emergency situations. It remains that we need to demonstrate that the theoretical solutions, indeed work well in practice.

Research in mesh networks thus requires us to complete a full cycle consisting of analysis, simulation and experimental evaluation in order to make true progress (cf. Figure 4.1). As a result, we decided to build a large-scale wireless mesh network to support our research. Similar infrastructures exist, such as the MIT

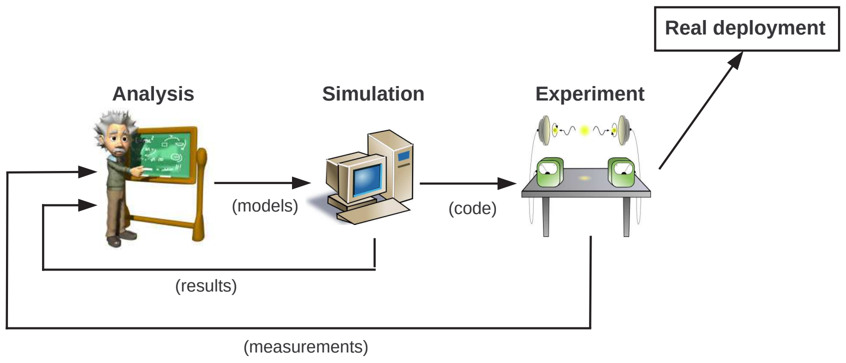


Figure 4.1: Scientific progress in WMNs requires a full cycle

roofnet [ROO] or the Magnets [KZP06] (currently renamed to BOWL[1]) testbeds. Nevertheless, as we did not have easy day-to-day access to these testbeds and none existed at EPFL, we decided to build from scratch our own multi-hop testbed on campus. During this deployment, we faced multiple challenges and design questions. In this chapter, we share some of the experiences and lessons we learned in this process.

4.2 Choice of Hardware and Software

4.2.1 Requirements and Challenges

The first step when deploying our indoor mesh network was to set the requirements for our testbed. In our case, we were interested in matching the following constraints.

- **Adaptability:** Even though our main interest lies in the scheduling problem of IEEE 802.11 mesh networks, we were interested in building a more general platform that would enable practical investigations of a wide range of research scenarios such as routing, user tracking, mobility, etc.

- **Programmability:** Due to the research-orientation of our deployment, we chose our hardware and software in order to provide adaptability even if it is at the cost of a small drop in relative performance. The first effect of this requirement appears in our choice of the firmware (openWRT[2]) and of the driver (MadWifi[3]) that are both open-source and thus enable us to modify the source code in order to meet our research needs.

- **Large-scale:** Having already performed small-scale experiments (e.g., in the basement of our school building), we were interested in providing a platform that is closer to a real deployment. Toward this goal, we designed our testbed to cover six buildings on our campus with different node densities between the buildings.

- **Price constraints:** Finally, the last constraint we needed to meet was a budget constraint that forced us to evaluate different alternatives, in order to find the hardware that provides the best tradeoff between the different requirements. We also stress that due to budget reasons, we did not consider the use of Software Defined Radio (SDR) in our architecture, even though it was the option offering the highest level of programmability.

[1]Berlin Open Wireless Lab (BOWL): http://bowl.net.t-labs.tu-berlin.de/
[2]OpenWRT firmware: http://openwrt.org/
[3]MadWifi driver: http://madwifi-project.org/

4.2.2 Hardware Description

In order to decide on the hardware forming our wireless testbed, we screened the different options and tested the four following routers as potential candidates (from most expensive to cheapest):

- **Mikrotik RouterBoard 532** was the first router we tested experimentally in [AHKT08]. It was available at a price around 250 $. This router is based on a CPU of 400 MHz, with 64 MB of DDR RAM. It provides two mini-PCI connectors that allows the use of 2 independent wireless interfaces.

- **Asus WL-500g Premium v1** is a small and efficient residential router that cost around 90$. It is based on a 266 MHz CPU (that can safely be over-clocked to 300 MHz) with 32 MB of RAM and 8 MB of flash. It provides a mini-PCI slot that initially contains a Broadcom wireless card (which can easily be replaced by any other mini-PCI card) and two USB ports that are very useful for extending the storage capacity with USB memory sticks.
  We note that today a new version of this router (v2) replaces the original version. Even though this second version is less expensive, it is also less interesting for us as researchers. Indeed, the mini-PCI slot has been replaced by a built-in wireless interface and this modification prevents us from easily testing new protocols by changing the wireless card.

- **Linksys WRT54GL** is a popular home router that cost around 60 $. It is based on a 200 MHz CPU with 16 MB of RAM and 4 MB of flash. It has a built-in wireless card that cannot be removed and thus does not offer the flexibility that a mini-PCI gives.

- **La Fonera** is the smallest and cheapest router we tested with a price of around 20 $. It is based on a 180 MHz CPU with 16 MB of RAM and 4 MB of flash.

After testing the routers under different settings, we found that the Asus WL-500g Premium offered the best price vs. performance tradeoff. The three major points that convinced us to select the Asus instead of the Linksys or La Fonera are:

1. It provides the flexibility to change the wireless card (i.e. it uses a mini-PCI instead of a built-in wireless interface), which is useful to easily adapt to changes in the MAC technology.

2. It has two USB slots that can be used to extend the storage memory. This feature is particularly useful, because it allows us to easily log arbitrarily large trace files on the router without any restrictions.

3. It does not have the 4 MB-flash limitation. Indeed, 4 MB is pretty limited when some useful softwares, such as Click [KMC+00], are included in the OpenWRT firmware image.

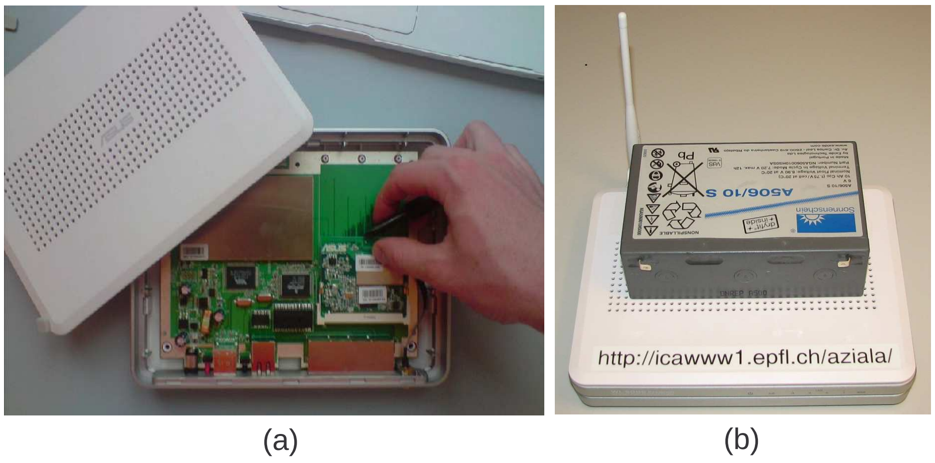


Figure 4.2: The Asus WL-500g Premium that we used in our testbed and where (i) we changed the WiFi mini-PCI card to an Atheros-based one, and (ii) we designed a technique to power our router through a battery, which enabled us to support mobile scenarios.

When purchasing our hardware in 2007, we chose the Asus over the Routerboard for two main reasons. First, the USB capabilities were a useful feature provided only by the Asus. Second, the small CPU gain of the Routerboard did not justify the price difference, which was an important factor as we were planning to buy around 60 routers (i.e., around $10,000$ $ difference in the total price). We note that nowadays, at the end of 2010, an interesting option that we are considering for upgrading our testbed are the Alix boards from PC Engines, which have the advantage of (i) running over an x86 architecture instead of mipsel, and (ii) having space for two WiFi interfaces and three antennas (that can for example be used to perform MIMO in IEEE 802.11n).

The Asus routers were provided with a power supply and a standard Broadcom mini-PCI WiFi card. These settings are good for a standard home user, but we needed to operate two modifications/enhancements in order to support the research scenario we had in mind (i.e., modifying/monitoring the MAC parameters and supporting mobile scenarios).

- **Allow modification/monitoring of MAC parameters:** In order to use the open-source MadWifi driver, we needed to change the mini-PCI card to an Atheros-based one as depicted in Figure 4.2.a. We used the NMP-8602 Atheros card that runs on IEEE 802.11a/b/g.

- **Enable mobility:** In order to eliminate the requirement to plug our router to the electrical grid (which prevents mobility), we equipped some of our devices with 6 V batteries as shown in Figure 4.2.b. To be able to meet the high amperage requirement to support the wireless interface of our router, we used A 506/10 batteries. Moreover, as the Asus routers work on an input voltage of 4.5 V, we added two inductances to the cabling connecting the battery to the router.

In addition to these modifications, we also equipped each router with a 2 GB memory stick that is automatically mounted during the boot-up of the router. After more than one year of constant use, we note that the power supply is the main source of failure of our routers. Indeed, we experienced almost no problems with the router hardware itself, but we had to replace around 10 power supplies that broke down.

4.2.3 OpenWRT Firmware

OpenWRT is a free open-source and Linux-based firmware. We decided to flash our routers with this distribution, because of the facility with which we can (i) install existing program and (ii) deploy our own C code. In Appendix B, we provide a detailed explanation of the methodology that needs to be followed in order to (i) build an image that is ready to be flashed on the routers, (ii) cross-compile our own program to run on the mipsel architecture of the Asus, and (iii) flash the new image to the routers and make the first boot-up configuration procedure.

A particularly interesting software to install in the OpenWRT image is the Click modular router [KMC$^+$00]: it allows us to completely control the processing/routing of a packet since it arrives to the router until it leaves it.

4.2.4 MadWiFi Driver

MadWifi is an interesting, but challenging wireless driver. On the one hand, its advantage comes from its open-source code that leaves room for easy modification and enhancement of the protocol. On the other hand, it suffers from one of the typical drawbacks of open-source software, as opposed to industrial products. For example, we noted that some of the supposedly working commands were only partially implemented (or even not implemented at all).

One of the key advantages that convinced us to adopt MadWifi is that it provides an easy access to multiple MAC parameters of IEEE 802.11, such as the CW_{min}, CW_{max}, RTS mode on/off, etc.
Nevertheless, we needed to modify ourselves the source code in order to achieve three objectives:

1. We unlocked the modification, via the *iwpriv* command, of the MAC layer parameters for Best Effort traffic (BE), the standard type of traffic in the network.

2. We enabled the use of multiple MAC queues (up to 4 or 8) as it is proposed in the IEEE 802.11e standard to achieve Quality of Service. Moreover, the possibility of using different queues is very useful in congestion-control schemes as it allows us to use different class of service depending on the nature of the next-hop.

3. We coded three new functions to the *wlanconfig* command in order to access the instantaneous queue occupancy and to access and modify the maximal MAC buffer length (usually locked to 50 packets).

As the technical details behind each modification are rather complex, we defer their description to Appendix B. Furthermore, we stress that nowadays new open-source drivers such as ath5k[4] and ath9k[5] have been launched as follow-ups of MadWifi in order to support the new generations of the IEEE 802.11 family, such as IEEE 802.11n.

4.3 Testbed Deployment

Once we finished with flashing and configuring our 60 Asus routers, we had to plan our deployment to be able to cover as many scenarios as possible. We initially planned our deployment for a flexible design in which the routers would be rapidly deployed and collected back directly after a round of measurements (or at the end of the day). But, our initial measurements appeared to suffer the same problems as the ones reported in [PYC06] and we had to reconsider our design. Indeed, we found that simple displacements of a mesh node by as little as a fraction of a centimeters leads to significant changes in the connectivity, from high SNRs to basically no connection at all. This issue was a real limitation for the reproducibility of the results as it is impossible to guarantee the exact same positioning of a router between two rounds of deployment.

Therefore, we finally decided to have a mostly fixed mesh network testbed with 41 routers installed and we kept the others as mobile devices that can rapidly be deployed for flexibility if needed. For the safety of the hardware, we deployed most of the routers within private offices. Concerning the routers that needed to be deployed in public spaces, we modified them to add a locking system in order to attach them to the fixed infrastructure.

4.3.1 Topology

The map of the final topology that we are currently using in our indoor testbed is depicted in Figure 4.3. We use 41 wireless routers that we spread among the 6 buildings of the I&C Faculty. We designed our deployment to form a fully-connected network, with a different node density among different buildings. Among the 41 routers, 37 are connected to the wired infrastructure by using the *wpa_supplicant* software for authentication with the IT infrastructure of the department. The 4 remaining nodes could not be directly connected to the wired infrastructure due to their location in the inter-building corridors that do not contain any wired connectivity plug.

[4]ath5k driver: http://linuxwireless.org/en/users/Drivers/ath5k
[5]ath9k driver: http://linuxwireless.org/en/users/Drivers/ath9k

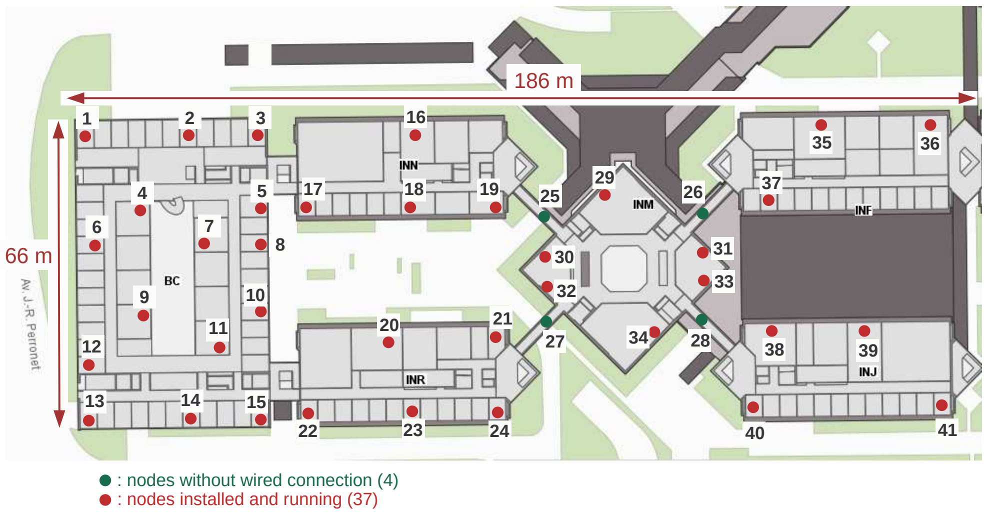


Figure 4.3: Map of our wireless mesh network deployed within EPFL. It covers 6 buildings over $10,000\ m^2$ and it contains 37 Asus routers that are accessible through a wired Ethernet connection and 4 nodes that are only accessible via wireless.

Even though the node positions remained fixed, this real-scale deployment enabled us to derive some interesting findings about the variability of the environment. Indeed, we found that both (i) people passing by, and (ii) windows and doors being opened/closed, led to drastic changes in the performance. These practical findings have two consequences for our work:

- It confirmed the need to perform long-term traces in order to obtain statistically meaningful results that are not impacted by the natural variability of the wireless environment.

- It supported our approach to propose dynamic solutions that automatically adapt to the changes of the environment and that does not require the a priori knowledge of the capacity region.

4.3.2 Network Control Tool

Although the offline processing of long-term traces is needed in the final step of a practical evaluation, it is very useful to be able to rapidly obtain live visualization of some performance metrics in other situations. Examples of cases where real-time plot are useful include (i) the early implementation/evaluation phase when looking for the sources of a problem; (ii) the final evaluation phase when finely tuning the system; and (iii) administrative and demonstration purposes where real-time visualization helps.

After some investigation, we could not find a software that matches our requirements of: (i) allowing the visualization of statistics such as the link throughput and the MAC queue occupancy (obtained after hacking the driver as described in Appendix B); (ii) allowing to control (i.e., launch commands) in all or part of the network from a central server; and (iii) being open-source and allowing simple modification of the source code for the addition of new visualization modules depending on the research needs.

Therefore, we developed our own program, called *Net-Controller*, that can create nice dynamic plots representing data collected on-the-fly from the network and that allows us to send arbitrary commands to any set of nodes. We mention that some tools, such as Jigsaw [CBB^{+}06], have been proposed in prior work to allow for the creation of a packet-level trace by merging the data obtained from multiple sniffers. Net-Controller significantly differs from Jigsaw-like tools in three ways:

- it is both very user-friendly with a graphical interface and easily expandable to monitor arbitrary metrics of interest in the network;
- it does not require dedicated monitoring nodes in the network and thus it is also able to display internal data that cannot be sniffed on the air, such as the queue occupancy or the variation in contention window;
- it is not only a monitoring tool. Indeed, Net-Controller allows to modify the parameters of any node in the network, directly from the graphical interface. Moreover, it also enables us to generate traffic from any set of nodes in the network to any other node with only a few mouse clicks (see video[6]).

Design Description of Net-Controller

We designed Net-Controller to work as two separated sub-programs that communicate together through a network socket (see Figure 4.4).

- **The server part** runs on a computer (desktop, laptop, or any other device with a display). It consists of a simple graphical interface that allows for dynamic plots to be generated, for the commands to be launched on the nodes, and for traffic to be started (or stopped). Requests are sent periodically to the nodes in order to retrieve the data to plot. The commands are launched through `ssh`. This program is written in Python and is multi-platform. It simply needs to know the IP addresses of the routers to interact with them.
- **The client part** runs on the wireless routers. Its role is to answer the requests sent by the server with the appropriate values. These values can be instantaneously obtained at any time (e.g., a queue occupancy), or aggregated (e.g., the number of bytes received since the last request, in order to compute a throughput). The current version of this module is implemented in C and

[6]Net-Controller - network management & visualization: http://icawww1.epfl.ch/NetController/

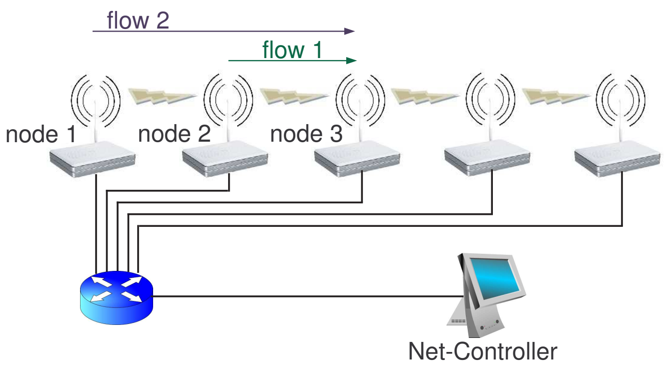


Figure 4.4: Illustration of the design structure of Net-Controller. The flows 1 and 2 are used to illustrate through an example the plots resulting from Net-Controller.

can answer requests for: (i) the link level throughput, for each IP flow going through this router; (ii) the occupancy of the IP and MAC layer queues; and (iii) the value of the IEEE 802.11 CW_{min} parameter. We designed our program to facilitate the addition of new parameters.

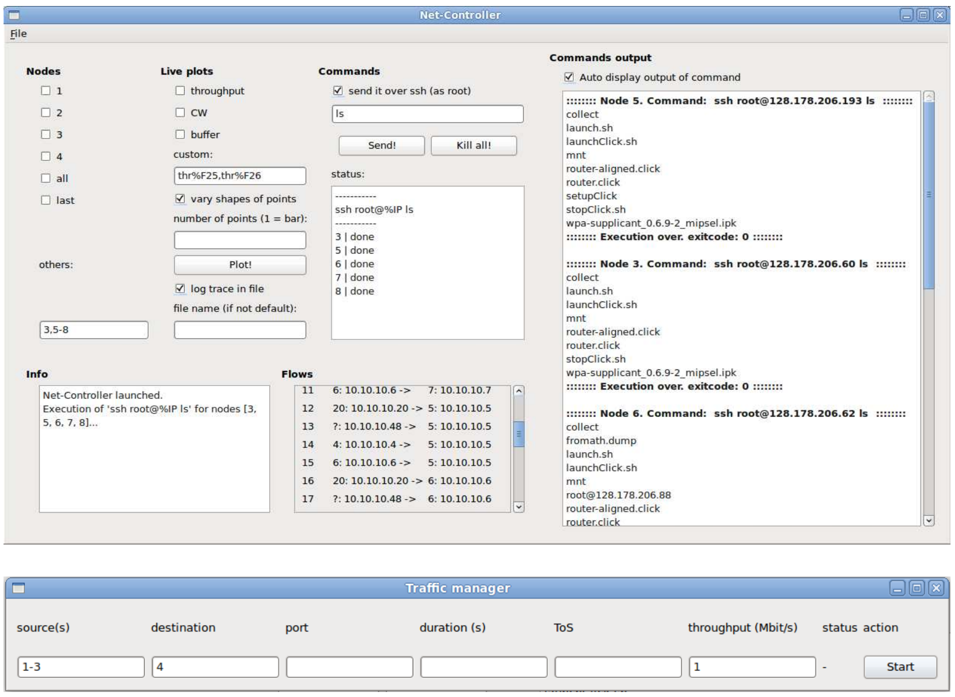
Having described the high-layer architecture of Net-Controller, we defer to Appendix B the technical discussion about the detailed installation procedure and use of the program.

Graphical Interface of Net-Controller

The control graphical interface of Net-Controller is depicted in Figure 4.5, where we see two active windows.

The window entitled ”Net-Controller” is the main control window that can be horizontally divided into three regions. In the top left, we have a space dedicated to the nodes and parameters selection. We note that Net-Controller also allows for logging in a trace file of the live statistics that are plotted. In the bottom left, there is a text field that summarizes the last commands that have been executed by the program.
The rightmost part of the window is a text field that is dedicated to displaying the output of the commands that have been launched in the different nodes.
Finally, in the middle of the window, we find at the bottom a list of all the flows that have been sensed by any of the router in the mesh network (our notion of flows corresponds to the tuple $< IP_{src}; IP_{dst} >$). In the middle top, we find the command section that allows the user to send and see the status of the command execution at the different nodes of the mesh.

The second window labeled ”Traffic Manager” is an add-on module that we created in order to easily launch traffic between any two pair of nodes. The way it operates is by using the parameters provided by the user in order to launch the



Net-Controller

Nodes: ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ all ☐ last
others: 3,5-8

Live plots: ☐ throughput ☐ CW ☐ buffer
custom: thr%F25,thr%F26
☑ vary shapes of points
number of points (1 = bar):
☑ log trace in file
file name (if not default):

Commands: ☑ send it over ssh (as root)
ls

status:

```
-----------
ssh root@%IP ls
-----------
3 | done
5 | done
6 | done
7 | done
8 | done
```

Info:

```
Net-Controller launched.
Execution of 'ssh root@%IP ls' for nodes [3,
5, 6, 7, 8]...
```

Flows:

```
11   6: 10.10.10.6 ->    7: 10.10.10.7
12   20: 10.10.10.20 ->  5: 10.10.10.5
13   ?: 10.10.10.48 ->   5: 10.10.10.5
14   4: 10.10.10.4 ->    5: 10.10.10.5
15   6: 10.10.10.6 ->    5: 10.10.10.5
16   20: 10.10.10.20 ->  6: 10.10.10.6
17   ?: 10.10.10.48 ->   6: 10.10.10.6
```

Commands output
☑ Auto display output of command

```
:::::::: Node 5. Command:  ssh root@128.178.206.193 ls  ::::::::
collect
launch.sh
launchClick.sh
mnt
router-aligned.click
router.click
setupClick
stopClick.sh
wpa-supplicant_0.6.9-2_mipsel.ipk
:::::::: Execution over. exitcode: 0 ::::::::

:::::::: Node 3. Command:  ssh root@128.178.206.60 ls  ::::::::
collect
launch.sh
launchClick.sh
mnt
router-aligned.click
router.click
stopClick.sh
wpa-supplicant_0.6.9-2_mipsel.ipk
:::::::: Execution over. exitcode: 0 ::::::::

:::::::: Node 6. Command:  ssh root@128.178.206.62 ls  ::::::::
collect
fromath.dump
launch.sh
launchClick.sh
mnt
root@128.178.206.88
router-aligned.click
router.click
```

Traffic manager

source(s)	destination	port	duration (s)	ToS	throughput (Mbit/s)	status	action
1-3	4				1	-	Start

Figure 4.5: User-friendly graphical interface of Net-Controller that allows us to (i) select the parameters and node to monitor; (ii) launch arbitrary command on any set of nodes; and (iii) launch traffic between any set of nodes.

corresponding *iperf* commands at the different nodes. Its major advantages lays in considerably speeding the experimental process by allowing us to rapidly launch or stop any flows in the network with a simple mouse click.

Example of Result Display

In order to illustrate an example of live graphical plots returned by Net-Controller, we consider the interesting example of Fair Queuing [DKS89].

Indeed, currently almost all off-the-shelf routers use a single queue with a FIFO policy (First-In, First-Out), but other policies have been proposed. In fair queuing, each node uses a separate queue for each individual flow and these queues are then scheduled in a round-robin manner. We implement fair queuing in Click [KMC+00] and capture its gain by having a fully backlogged UDP 1-hop flow (flow 1) and a 2-hop flow (flow 2); both go through node 2 as depicted in Figure 4.4. Figure 4.6 shows that the standard FIFO policy completely starves the 2-hop flow, whereas fair queuing fairly shares the throughput between the two flows (max-min fairness). The starvation problem in FIFO occurs because the single queue of node 2 is always full of packets from flow 1, and thus drops most packets from flow 2.

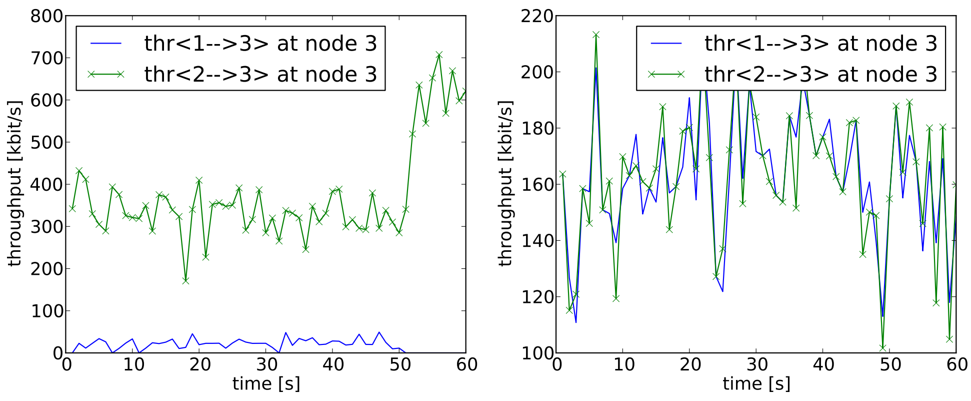


Figure 4.6: Throughput received at the destination for the two flows. With the standard FIFO policy (left) and with fair queuing (right). The plots are produced by Net-Controller. In the legend, "thr$\langle x \rightarrow y \rangle$" denotes the throughput of the flow from the source node x to the destination node y.

4.4 Concluding Remarks

In this chapter we described some of the challenges we faced and some of the experiences we gained while building from scratch our indoor wireless multi-hop network testbed.

In addition to the work presented in this thesis, our wireless multi-hop network[7] supported multiple other research work. Some examples of projects that used our testbed for their experimental evaluation cover the fields of

- Indoor user localization and tracking using IEEE 802.11 [Epi10].
- Epidemic forwarding protocols and the effect of mobility [EFLBA10].
- Correlation of erasure between links [JSPF$^+$10].
- Opportunistic routing protocols for multi-hop networks [Bec11].

The usage of our wireless testbed in a large range of research scenarios is an indicator that we successfully reached our goal of adaptability in the design and deployment of this infrastructure.

Furthermore, we presented the tool that we developed to manage and monitor a wireless testbed. Net-Controller was used for our demo at Infocom in 2010 [HAT]. There, we could launch experiments on our EPFL testbed and visualize the network performance in live from San Diego, USA. We provide Net-Controller as an open-source software in our website[8].

[7] Aziala - a wireless multi-hop testbed for research purposes: http://icawww1.epfl.ch/aziala
[8] Net-Controller - network management & visualization: http://icawww1.epfl.ch/NetController/

Chapter 5

MAC Layer Congestion-Control

5.1 Background

5.1.1 Problem Statement

We consider the case of a wireless multi-hop topology such as the one existing in the backhaul of a mesh network. As depicted in Figure 5.1, the backhaul of a wireless mesh is composed of three types of nodes: (i) a Wired Access Point (WAP) that plays the role of gateway and is connected to the Internet, (ii) Access Points (APs) that ensure the access part of the WMN by having the end-users connected to it (note that usually the backhaul and access part of a WMN run on independent channels to avoid interferences) and (iii) Transit Access Point (TAPs) that transports the data packets through multiple hops from the WAP to the AP and back.

In this settings, we recall the results from Chapter 3, where we showed that (i) IEEE 802.11 networks with more than 3 hops are intrinsically unstable and that (ii) a simple static stabilization strategy can solve this problem by correctly choosing the throttling factor $q \in [0, 1]$. Nevertheless, we also recall that the throt-

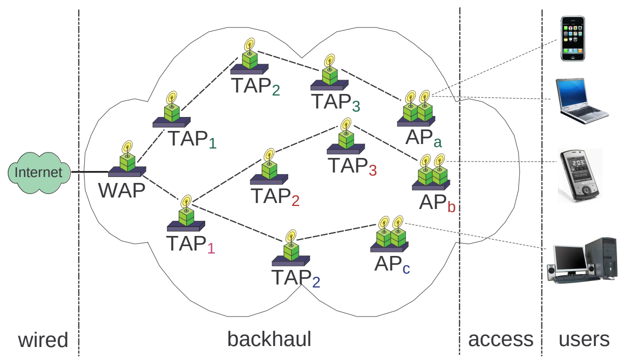


Figure 5.1: A wireless mesh network consists of a backhaul and an access part.

tling parameter q is topology-dependent. This is a serious limitation for wireless mesh networks because (i) the traffic matrix is likely to vary over time and (ii) the exact topology might even not be known in advance (e.g., in the case of an emergency requiring a rapid deployment). Hence, there is still a need to develop self-adaptive algorithms that can stabilize the network independently of the topology. In this chapter, we satisfy this requirement by introducing EZ-flow, a hop-by-hop congestion-control mechanism that automatically adapts to the network status.

5.1.2 System Requirements

In the design of our mechanism we focus on developing a practical, stabilizing solution that is compatible with current equipments and protocols used in IEEE 802.11 wireless mesh networks. Toward this goal, we set four main requirements:

- **Network stabilization**: EZ-flow is designed mainly to ensure network stability, where a network is stable if all the relay nodes have finite queues when equipped with infinite buffers (see Definition 3.2). In practice, when buffers are finite, this means that no queue builds up. Furthermore, as the environment changes in real networks, we require EZ-flow to automatically adapt itself to changes in the traffic matrix.

- **End-to-end delay reduction**: The first implication of network stability is a reduced end-to-end delay that should be maintained low with EZ-flow, compared to IEEE 802.11 alone. Such a requirement of low delays is of utmost importance in cases where a mesh network supports real-time, multimedia services such as VoIP, video-on-demand or online-gaming.

- **Unmodified MAC layer**: We require that the IEEE 802.11 MAC layer remains unmodified in order to ensure the compatibility of our solution with the mesh networks already deployed. To meet this objective, we propose to implement EZ-flow as a separate program that interacts with the MAC layer solely through the contention window CW_{min} parameter of IEEE 802.11.

- **Backward compatibility**: We ensure the backward compatibility of EZ-flow by having each node derive the needed information without message passing. This approach allows for the possibility of an incremental deployment of EZ-flow in an already existing mesh.

In addition to these requirements we add two properties that are not primary goals of EZ-flow, but that are still desired properties that appeared in all our simulations and experimental deployments:

- **Fairness improvement**: We take IEEE 802.11 as a baseline for EZ-flow and note that fairness is improved through a higher Jain's fairness index value

$$FI = \frac{(\sum x_i)^2}{(n_{flows} \cdot \sum x_i^2)} \tag{5.1}$$

where n_{flows} is the total number of flows and x_i is the throughput achieved by flow i.

- **Fair throughput improvement**: Low delays and high throughput are often seen erroneously as antagonist goals to be pursued. Of course, reducing the application throughput to a very low value will always ensure low delay. Nevertheless, we do not want our mechanism to limit itself to ensuring delay at the price of throughput. For the same level of fairness, we therefore require that EZ-flow achieves a global throughput that is higher than with IEEE 802.11 alone.

5.1.3 Related Work

Much effort has been put into understanding how 802.11 behaves in a multi-hop environment. Previous work show the inefficiency of the protocol in providing optimal performance, as far as delay, throughput and fairness are concerned [GSK04]. In [NNCA06], Nandiraju et al. propose a queue management mechanism to improve fairness. However, as they mention in their conclusion, a solution to the inherent unfairness of the IEEE 802.11 MAC layer is needed for their mechanism to work properly. In [JP09], Jindal et al. claim that the performance of IEEE 802.11 in multi-hop settings is not as bad as could be expected. For instance, they show an example through simulation where IEEE 802.11 achieves a max-min allocation that is at least 64% of the max-min allocation obtained with a perfect scheduler. Our experiments, in Section 5.3, show that the performance may actually be much worse. We believe that the cause of the discrepancy is that [JP09] assumes that flows are source-rate limited, whereas we do not make such an assumption.

A first analytical solution to the stability problem in multi-hop networks is discussed in the seminal work of Tassiulas et al. [TE92], which introduces a back-pressure algorithm. Their methodology uses a centralized scheduler that selects for transmission the link with the greatest backlog difference, i.e. the greatest difference in queue occupancy between the MAC destination node and the MAC source node. Such a solution works well for a wired network, but is not adapted to a multi-hop wireless network where decentralized schedulers are needed due to the synchronization problem. Extensions from this work to distributed scheduling strategies have been discussed in works such as [CKLS08], where Chapokar et al. propose a scheduler that attains a guaranteed ratio of the maximal throughput. Another effort to reduce the complexity of back-pressure is presented in [YST08], where Ying et al. propose to enhance scalability by reducing the number of queues that need to be maintained at each node. The interaction between an end-to-end congestion controller and a local queue-length-based scheduler is discussed by Eryilmaz et al. in [ES05]. The tradeoff that exists in each scheduling strategy between complexity, utility and delay is discussed in depth in [YPC08]. One of the drawbacks of these previous methods is that they require queue information from other nodes. The usual solution is to use message passing, which produces an overhead and is thus costly even if it is limited to the direct neighbors.

Some recent work propose schedulers that do not require buffer information from other nodes. In [GLS07], Gupta et al. propose an algorithm that uses the maximum node degree in the network. Proutiere et al. [PYC08] propose another algorithm, where each node makes the scheduling decision based solely on its own buffer. Finally, most recently Shin et al. propose an algorithm that achieves stability and where each node uses its own buffer occupancy with a $\log\log$ function to make the scheduling decision [SSR09]. Nevertheless, even though their algorithm is efficient for the case of a perfect CSMA, it requires a very large buffer size (i.e., in the order of thousands of packets). Such a requirement presents two drawbacks: First, large buffers imply a large end-to-end delay; second, the requirement of such large buffers does not match with current hardware, which usually have a standard MAC buffer of only 50 packets. To sum up, despite recent and significant progress on the theoretical side, almost all the existing solutions are still far from being compatible with the current IEEE 802.11 protocol. One exception, which was developed in parallel with our work, is the hop-by-hop congestion control scheme in [WJHR09]. In their paper, Warrier et al. propose and deploy DiffQ, which is a protocol implementing a form of backpressure (i.e., prioritizing links with large backlog differential). To achieve this implementation, DiffQ makes each node inform its neighbors of its queue size by piggybacking this information in the data packet (i.e., modifying the packet structure by adding an additional header) and then schedules the packets in one of the four MAC queues (each with different CW_{min} value) depending on the backlog difference. Our approach differs in two ways: (i) we use the next-hop buffer information instead of the differential backlog, which results in an implicit congestion signal being pushed back more rapidly to the source; (ii) as opposed to DiffQ, we do not modify the packet structure in any way as we passively derive the next-hop buffer occupancy without any form of message passing. To the best of our knowledge, EZ-flow is the first implementation that solves the turbulence and instability problem in real 802.11-based multi-hop testbed without modifying the packets and without any form of message passing. Our approach differs from all the previous works in the sense that we propose a practical solution, implemented with off-the-shelf hardware, where we take advantage of the broadcast nature of the wireless medium to derive the buffer information of neighboring nodes. We also highlight that the novel passive buffer derivation methodology of our BOE module is potentially compatible with new algorithms such as DiffQ, and it could allow them to eliminate the need to piggy-back the buffer information (resulting in unmodified packet structure).

Another line of work, parallel to ours, tackles congestion at the transport layer rather than the MAC (link) layer. In [RJJP08], Rangwala et al. present limitations of TCP in mesh networks and propose a new rate-control protocol named WCP that achieves performances that are both more fair and efficient. Similarly, Shi et al. focus on the starvation that occurs in TCP when a one-hop flow competes with a two-hop flow and they propose a counter-starvation policy that solves the problem for this scenario [SGM$^+$08]. Garetto et al. also tackle the starvation problem at an upper layer [GSK08]. They propose a rate-limiting solution and evaluate it by simulation. Their major motivation for not using MAC-based approach is to ensure compatibility with 802.11-based mesh network currently deployed. EZ-flow is also fully compatible with the existing protocol since it only varies the contention window CW_{min}, a modification allowed by the standard. Our approach differs from previous work in the sense that we tackle the problem at the MAC layer and that our methodology solves the problem both for bi-directional traffic (e.g TCP) or uni-directional traffic that cannot count on feedbacks from the final destination to adapt its rate (e.g. UDP).

Finally, another kind of work, which is similar to ours in the idea of exploiting the broadcast nature of the wireless medium, is found in cooperative diversity and network coding. In [KRW$^+$08], Katti et al. propose that relay nodes listen to packets that are not necessarily targeted for them in order to code the packets together later on (i.e. XOR them together) and thus increase the channel capacity. In [BM05], Biswas et al. present a routing mechanism named ExOR that takes advantage of the broadcast nature to achieve cooperative diversity and thus increase the achievable throughput. Furthermore, in [HRGD05] Heusse et al. also use the broadcast nature of IEEE 802.11 to improve the throughput and fairness of single-hop WLANs by replacing the exponential backoff with a mechanism that adapts itself according to the number of slots that are sensed idle. Our work follows the same philosophy of taking advantage of the "free" information given by the broadcast nature. Apart from that, our approach is different, because we do not use cooperation and network coding techniques at relay nodes, but instead in a competitive context we derive and use the next-hop buffer information to tackle the traffic congestion occurring in multi-hop scenarios.

To work in combination with routing solutions such as ExOR, our approach could be extended . Truly, the fact that the forwarded packets are not all sent to the same successor node implies that the forwarding process may not be FIFO (First-In, First-Out) anymore and thus the information derived by the BOE becomes more noisy. Nevertheless, by using a larger averaging period to smoothen the noise, this information could still be useful for congestion control. Moreover, to perform congestion control, a node does not always precisely need to know which successor (i.e., which next-hop relay) gets its packets: it just needs to keep to a low value the *total* number of packets that are waiting to be forwarded at all of its successors. This could be done using a similar methodology to the one presented in this paper for the unicast case. A similar extension of a congestion-control from unicast to multicast is discussed by Scheuermann et al. in [STL$^+$07].

5.2 EZ-Flow: a Scheme without Message Passing

5.2.1 EZ-Flow Description

First, we introduce the notion of flow, where a flow is a directed communication between a source and a destination. In the multi-hop case, the intermediate nodes

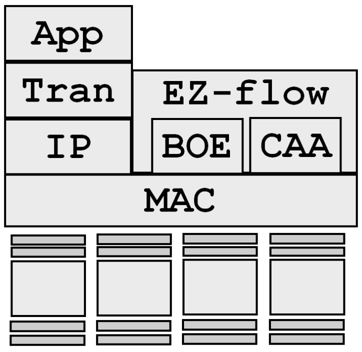


Figure 5.2: EZ-flow works as two modules implemented above the MAC layer. The Buffer Occupancy Estimator (BOE) passively derives the next-hop queue occupancy and transmits this information to the Channel Access Adaptation (CAA) module, which acts on a IEEE 802.11 parameter (i.e., the CW_{min}) to perform congestion control.

act as relays to transport the packets to the final destination. A node $i+1$ is the successor node of node i along a given flow if it is the next-hop relay in the multi-hop flow. We denote the queue occupancy of node i by b_i and its minimal contention window (CW_{min}) by cw_i. In order, not to starve forwarded traffic, each node that acts both as a source and relay should maintain 2 independent queues: one for its own traffic and the other for the forwarded traffic. Furthermore, a node that has multiple successors should maintain 1 queue per successor (2 if it acts as source and relay). Indeed, different successors may encounter different congestion levels and thus EZ-flow performs best if it can adapt the channel access probability per successor. Note that, this requirement is scalable as EZ-flow does not need queuing per destination, but per successors and the number of successors is typically limited to a single digit in the case of a WMN.

Second, we describe the two modules forming EZ-flow that work as two independent programs that collaborate to achieve congestion-control above the MAC layer (see Figure 5.2).

- A Buffer Occupancy Estimator (BOE) that derives the queue status of the successor node along a flow.

- A Channel Access Adaptation (CAA) that uses the information from the BOE to adapt the channel access probability through cw_i.

5.2.2 Buffer Occupancy Estimation

One of the major novelties of EZ-flow lies in the BOE that passively derives the queue occupancy at the successor node b_{i+1} without requiring any type of message passing. We emphasize that our BOE works differently than estimation approaches, such as [Jia07], that send probe packets to estimate the total queue size. Instead, in our approach each node i passively computes how many of its own packets are queued at node $i+1$. Using this information, instead of the total queue size, EZ-flow aims to keep the number of packets at a successor's queue small. This design choice prevents a node from starving itself due to non-cooperative neighbors (not performing congestion control).

To perform its task, the BOE keeps in memory a list L of the identifiers of the last 1000 packets it sent to a successor node. In our deployment we use the 16-bit checksum of the TCP or UDP packet as an identifier so as not to incur any computational overhead due to processing the packet. We note that this identifier, present in the packet header, could be used by any mesh network based on TCP/UDP and IP, and this is clearly the standard in currently deployed networks. We stress, however, that this design choice is used without any loss of generality. Even if, in the future, the standard would be to run IPsec or to use non-TCP/UDP packets, our mechanism would instead simply need to use a lightweight hash of the packet payload as an identifier.

Algorithm 1 EZ-flow mechanism at node i

```
  BOE module:
  if transmission of packet $p$ to node $i+1$ then
    Store checksum of $p$ in $PktSent[]$   (overwrite oldest entry if needed)
    $LastPktSent$ = checksum of $p$
  else if sniffing of packet $p$ from $i+1$ to $i+2$ then
    if checksum of $p \in PktSent[]$ then
      $b_{i+1}$ = number of packets in $PktSent[]$ between $p$ and $LastPktSent$
      return $b_{i+1}$ to CAA module
    end if
  end if

  CAA module:
Require: Reception of $50$ $b_{i+1}$ samples from BOE
  $\bar{b}_{i+1}$ = Average of $50$ $b_{i+1}$ samples
  if ($\bar{b}_{i+1} > b_{max}$) then
    $count_{down} \leftarrow 0$; $count_{up} \leftarrow count_{up} + 1$
    if ($count_{up} >= log(cw_i)$) then
      $cw_i \leftarrow cw_i \cdot 2$; $count_{up} \leftarrow 0$
    end if
  else if ($\bar{b}_{i+1} < b_{min}$) then
    $count_{up} \leftarrow 0$; $count_{down} \leftarrow count_{down} + 1$
    if ($count_{down} >= 15 - log(cw_i)$) then
      $cw_i \leftarrow cw_i/2$; $count_{down} \leftarrow 0$
    end if
  else
    $count_{up} \leftarrow 0$; $count_{down} \leftarrow 0$
  end if
```

The second information needed is the identifier of the packet that is actually forwarded by the successor node. This piece of information can be obtained by taking advantage of the broadcast nature of the wireless medium. Indeed, node i is on the range of $i + 1$ and is thus able to hear most of the packets that are sent by node $i + 1$ to $i + 2$. In the usual settings, the MAC layer at each node transmits to the upper layer only the messages that are targeted to it and ignores the messages targeted to other nodes. However, by setting a node in the monitoring mode, it is possible to sniff packets that are targeted to other nodes through a raw socket (as tcpdump does[1]). Using such a methodology, it is then possible for a node to track which packets are being forwarded by its successor node without it requiring any form of message passing.

Finally, as the standard buffering policy is ”First In, First Out” (FIFO), node i can accurately compute the number of its packets stored at node $i + 1$ each time it hears a packet from node $i + 1$.
Indeed, it only needs to compare the identifier of the packet it hears with the identifiers of the sent packets it has in the list L. The number of packets between the corresponding match (the packet that node $i + 1$ forwards) and the last packet that node i sent (the last entry in the list L) corresponds to b_{i+1}.
It is important to note that, in order to perform its task, the BOE module does not need to overhear all the packets forwarded by node $i + 1$. Instead, it is enough for it to be able to overhear some packets. Each time node i overhears a forwarded packet from node $i + 1$ (which happens most of the time, experimentally), it can precisely derive the queue occupancy and transmit it to the CAA that will react accordingly. Obviously, the more forwarded packets node i can overhear, the faster it can detect and react to congestion. Nevertheless, even in the hypothetical case where node i is unable to hear most of the forwarded packets, it will still adapt to the congestion and eventually set its contention window to the right value. This robustness of EZ-flow to forwarded packets that are not overheard is a crucial property, as some packets may be missed due to the variability of the wireless channel or hidden node situations.

5.2.3 Channel Access Adaptation

The second module of EZ-flow is the CAA that adapts the channel access probability according to $\bar{b}_{i+1}$, which is the 50-sample average of the b_{i+1} derived by the BOE. The intuition behind EZ-flow is that in the case a successor node already has many packets to forward, it is useless to send it more packets. Even worse, sending more packets degrades the performance. Indeed, every time node i sends a new packet to be forwarded, node $i + 1$ looses a chance to transmit.

Following this result, we propose a simple policy for the CAA that uses solely two thresholds: (i) b_{min} and (ii) b_{max}. Then it adapts the channel access of each node by changing its value of the contention window cw_i. Indeed, every time the

[1]The Tcpdump Manual Page: http://www.tcpdump.org/

node i needs to send a packet when the channel is not idle, it randomly chooses a backoff value that is inside the interval $[0, cw_i - 1]$ and it waits for this amount of time before retrying to transmit (see Section 2.3 for more details on IEEE 802.11). Therefore, we note that the higher the cw_i is, the lower the channel access probability is.

Our policy makes the decision based on a time average of the queue occupancy at the successor node ($\bar{b}_{i+1}$). We set the time average parameter to be 50 samples and then one of three cases may occur:

- $\bar{b}_{i+1} < b_{min}$: the average queue at node $i + 1$ is below the lower threshold. This shows that the buffer is underutilized. Thus node i should increase its channel access probability by dividing cw_i by a factor of two.

- $\bar{b}_{i+1} > b_{max}$: the average queue at node $i + 1$ is above the upper threshold. This shows that the buffer is overutilized (or is even overflowing). Thus node i should decrease its channel access probability, which it does by doubling cw_i.

- $b_{min} < \bar{b}_{i+1} < b_{max}$: this is the desired situation as the buffer is correctly utilized by being neither empty most of the time nor saturated. In this case, node i concludes that it has a correct channel access probability and thus keeps cw_i unchanged.

Other policies than multiplicative-increase, multiplicative-decrease could be used to update cw_i in order to have a higher range of possible values. Yet, we chose this policy due to the hardware constraint that requires setting cw_i at powers of 2.

Furthermore, we provide a better inter-flow fairness in EZ-flow by using two parameters:

- $count_{up}$ counts the number of successive times the condition ($\bar{b}_{i+1} > b_{max}$) happens (overutilization).

- $count_{down}$ counts the number of successive times the condition ($\bar{b}_{i+1} < b_{min}$) happens (underutilization).

These two pieces of information are then used to update the contention window parameter according to the current cw_i value, where nodes with a high cw_i react both more quickly to underutilization signals and more slowly to overutilization signals than nodes with a low cw_i.

Finally, the selection of the parameters b_{min} and b_{max} can affect the reactivity and the speed of convergence of EZ-flow, depending on the topology. Indeed, the smaller the gap between these two values, the higher the reactivity of EZ-flow to slight variations, whether due to variations of the traffic load or not. These parameters can thus be fine tuned depending on the desired behavior, but fortunately the general values of b_{min} and b_{max} already significantly improve the situation compared to standard IEEE 802.11. Indeed, the most important parameter to set is

b_{min}, which has to be very small (i.e., $\sim 10^{-1}$) in order to avoid that the nodes too often become too aggressive and reach unsupportable rates. The parameter b_{max} can then be set with more flexibility, depending on the desired reactivity.

5.3 Experimental Evaluation

We implement EZ-flow and evaluate it on 9 off-the-shelf wireless nodes of our testbed. First, we describe the environment and hardware used in our experiment and discuss the practical details for the implementation of EZ-flow. Then, we present the measurement results that confirm the efficiency of EZ-flow in improving the performance in a real WMN environment.

5.3.1 Hardware and Software Description

We use 4 laptops running Linux, which act as source and sink of the traffic, and 9 wireless nodes equipped with an omni-directional antenna that represent the multi-hop backhaul of a mesh network. We recall that the wireless routers are Asus WL-500gP, in which we change the mini-PCI WiFi card to an NMP-8602 Atheros card. Each router runs the OpenWRT firmware [OPE] with the MadWifi driver [MAD] modified to perform both queue monitoring and the modification of the contention window. The wireless cards operate in 802.11b at a fixed transmission rate of 1 Mb/s and with the RTS/CTS mechanism disabled. Finally, we set the routing to be static.

We implement the two modules of EZ-flow, the BOE and CAA, in C code as described in Section 5.2. Two practical constraints need to be accounted for. Both of them are not required in other implementations with different hardware.

1. **Sniffer constraint:** We initially intended to deploy both the BOE and CAA module within the same wireless card (i.e., the same router), but we had to reconsider our design. Indeed, the BOE acts mostly as a sniffer that collects the packets sent either by a node itself or its direct forwarder. The problem is that a WiFi card cannot transmit and receive at the same time and therefore is unable to truly sniff its own packet on the air. Instead the best a sniffer can do is to capture the packet before it is sent to the MAC layer to be actually transmitted on the air. The drawback of this technique, however, is that packets can be sniffed as sent by a node, even though they are dropped by the MAC layer (for example a buffer overflow), and thus are never really physically transmitted. To overcome this limitation, we use two WiFi interfaces per wireless node (i.e., two routers connected through an Ethernet cable). One interface is responsible for sending the traffic and running the CAA. The other interface does not transmit any packets and acts only as a sniffer that implements the BOE. We use this approach to simplify the practical deployment. EZ-flow does not require the use of two interfaces. Indeed, another approach could be to use only one interface and to directly

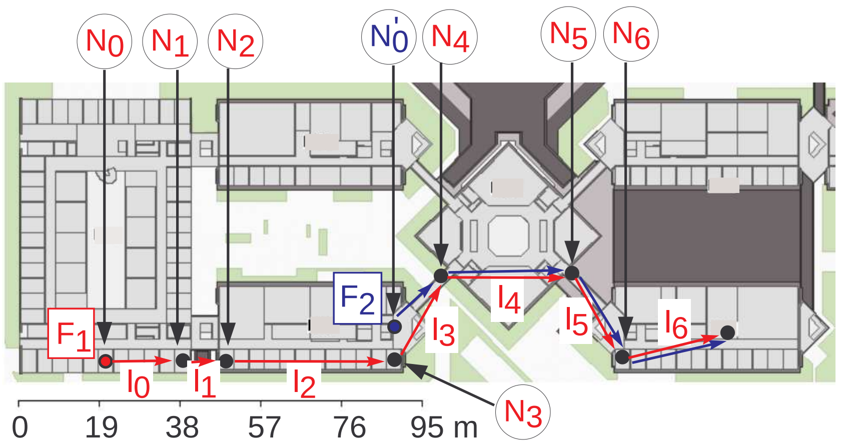


Figure 5.3: Illustration of the testbed topology. The hardware used are Asus WL-500gP routers with an Atheros-based wireless card.

implement EZ-flow at the kernel level of the wireless driver (and not the application level) in order for the BOE to capture only the packets that are truly sent at the physical layer.

2. **MadWifi constraint:** The second practical constraint comes from the *iwconfig* command of the Madwifi driver to increase the contention window CW_{min}. Indeed, it has no effect above 2^{10} (even though the driver allows the command to execute up to 2^{15}). We noticed this flaw in the implementation of the MadWifi command by checking a single-link capacity for different CW_{min} values and observing that it significantly varies up to 2^{10}, but it remains unchanged between 2^{10} and 2^{15}.

5.3.2 Topology Description

We deploy our testbed over 4 buildings of the university campus where at most 2 flows are concurrently active. Figure 5.3 presents the exact map of our mesh network deployment. On the one hand, the flow F_1 is a 7-hop flow for which the bottleneck link is l_2 as shown in Table 5.1. On the other hand, the flow F_2 is a shorter flow of 4 hops that shares the same path than F_1 and produces a typical parking-lot scenario. For the sake of comparability, we avoid the effect of interference from other networks by running our experiments on channel 12 during the night (1 am - 5 am), but we stress that the instability problem remains also during daytime as shown in our demo[2]. Finally, we use the values from Table 5.1 to obtain the theoretical optima from Table 5.2 that assume a k-hop interference

[2]Demo available at: http://icawww1.epfl.ch/NetController/ (Video 2)

	Mean throughput	Standard deviation
l_0	845 kb/s	23 kb/s
l_1	672 kb/s	49 kb/s
l_2	408 kb/s	67 kb/s
l_3	748 kb/s	42 kb/s
l_4	746 kb/s	28 kb/s
l_5	805 kb/s	27 kb/s
l_6	648 kb/s	43 kb/s

Table 5.1: Illustration of the capacity of each link of flow F_1. The means are obtained through measurements over 1200 s.

effect between the links with $k = 2$ and $k = 3$ (the experimental setup is somewhere between this two ranges). To do so, we compute the capacity of all paths of interfering links

$$C_j^{j+k} = 1/(\sum_{i=j}^{j+k} \frac{1}{C_i}) \text{ , for } 0 \leq j \leq 6 - k,$$

and where C_i is the capacity of link l_i. The theoretical optimum is then obtained by taking the capacity $C_{j'}^{j'+k}$ of the bottleneck path of interfering links within a flow.

5.3.3 Measurement Results

The first scenario we consider is when F_1 is alone in the network. Figure 5.4 shows the queue evolution with standard IEEE 802.11 and with EZ-flow turned on. We note that for IEEE 802.11 both nodes N_1 and N_2 saturate and overflow, due to the bottleneck link l_2 (between N_2 and N_3), whereas the queue occupancy of all the other nodes is negligibly small, similarly to N_3. This results in an end-to-end throughput of 119 kb/s as shown in Table 5.2 (note that a similar throughput degradation for the backlogged case has been observed through simulation in [LBDC$^+$01]). In contrast, EZ-flows detects and reacts to the bottleneck at link l_2 by increasing cw_1 up to 2^8. This action stabilizes the queue of N_2 by reducing the channel access of link l_1. Similarly, EZ-flow detects that the queue of N_1 builds up and forces N_0 to increase cw_0 until it reaches our hardware limit of 2^{10} (see Section 4.1). This hardware limitation prevents EZ-flow from reducing the queue occupancy of N_1 to a value as low as N_2. However, we stress that despite this hardware limitation, EZ-flow still significantly improves the performance by reducing the turbulence of the flow and increasing the throughput to 148 kb/s (close to the 3-hop interference range theoretical optimum and mapping to a 41% reduction in the gap to the 2-hop optimum). Furthermore, we show through simulation in Section 5.4 that EZ-flow completely stabilizes the network once this limitation is removed.

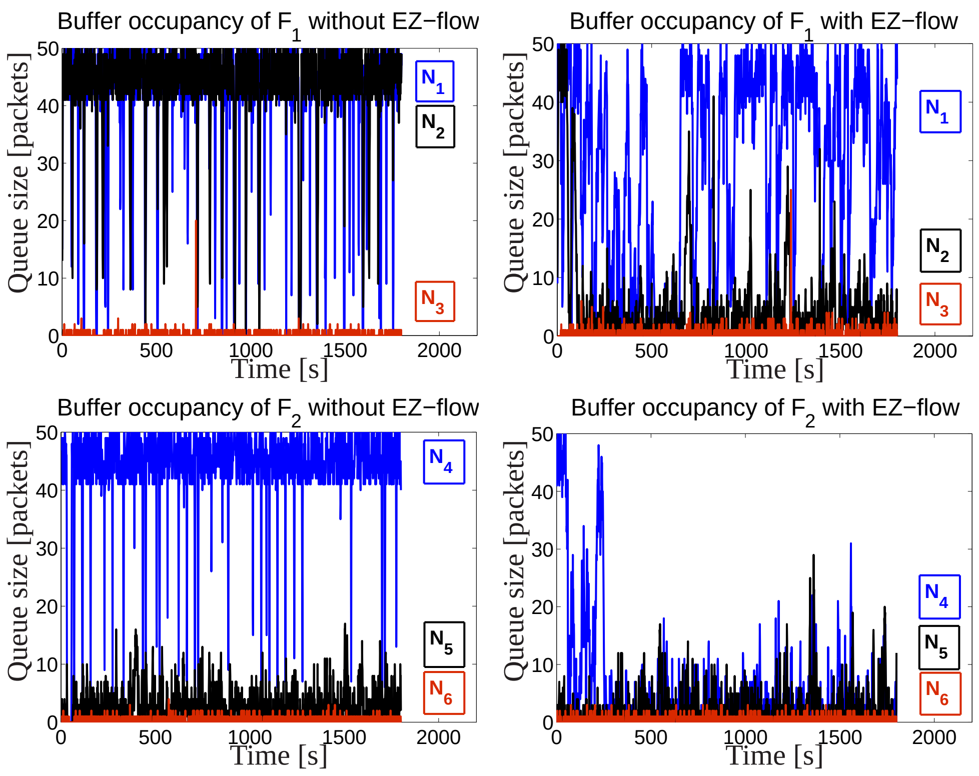


Figure 5.4: Experimental results for the queue evolution of the relay nodes when flow F_1 or F_2 are active. The average number of buffered packets are: (i) without EZ-flow 41.6 (N_1), 43.1 (N_2) and 43.7 (N_4) and (ii) with EZ-flow 29.5 (N_1), 5.2 (N_2) and 5.3 (N_4). The remaining queues are very small.

In the second scenario, we consider F_2 alone. Similarly to our mathematical analysis of Section 3.4, we note that for IEEE 802.11 the queue of the first relay node of F_2 (i.e., N_4) builds up and overflows, resulting in a throughput of 157 kb/s. However, EZ-flow completely stabilizes the network for all the relay nodes (no queue builds up) by making the source node $N_0^{'}$ increase $cw_0^{'}$ up to 2^8. Thus EZ-flow works even better in this scenario where it is not blocked by the hardware limitation and it achieves a throughput of 185 kb/s.

Finally the last scenario is a parking-lot scenario where both F_1 and F_2 are simultaneously active. Similarly to what is also reported in [SGM$^+$08] between a 1- and 2-hop flow, Table 5.2 shows that IEEE 802.11 performs very poorly: the long flow F_1 is completely starved in favor of the short flow F_2, because $N_0^{'}$ is too aggressive (even for its own flow) and thus prevents the packets from the longer flow F_1 from being relayed by the intermediate nodes N_1, N_2, N_3. However, by its nature, EZ-flow solves the problem by making the two source nodes, $N_0^{'}$ and N_0, become less aggressive in order to stabilize their own flow. This approach

	Mean throughput	Theoretical optima		Jain's Fairness
		$k = 3$	$k = 2$	
F_1	119 kb/s	151 kb/s	190 kb/s	
F_2	157 kb/s	183 kb/s	242 kb/s	
F_1 F_2	7 kb/s 143 kb/s			0.55
F_1^{EZ}	148 kb/s	151 kb/s	190 kb/s	
F_2^{EZ}	185 kb/s	183 kb/s	242 kb/s	
F_1^{EZ} F_2^{EZ}	71 kb/s 110 kb/s			0.96

Table 5.2: Measurements over 1800 s with and without EZ-flow. Theoretical optima are obtained assuming a 3-hop (2-hop) interference range. The sub-division in the table shows the results for: (i) one single flow, or (ii) two simultaneous flows.

thus solves the starvation problem and significantly increases both the aggregate throughput of F_1 and F_2 and the Jain's fairness index.

5.3.4 Effect of Bi-Directional Traffic

EZ-flow is designed to stabilize the queues within a flow independently of the interferences caused by other flows. In the previous sub-section, we investigated the effect of having multiple flows by looking at a setting where two separate flows share part of their path to reach the same destination (e.g., the gateway).

We now focus on a different scenario, where two flows take exactly opposite paths (i.e., the destination of a flow is the source of the other flow). Toward this goal we use the experimental setting depicted in Figure 5.5, where the two 4-hop flows are $F_{0\to4}$ (from node 0 to node 4) and $F_{4\to0}$ (from node 4 to node 0). The measurements show a serious throughput asymmetry in this setting. Indeed, we set the data rate of all nodes to 2 Mb/s and when first launching each flow by itself, we

	w/o, RTS, w/o EZ	w/o RTS, EZ	RTS, w/o EZ	RTS, EZ
b_1	93	37	37	2
b_2	40	2	2	1
b_3	0	0	0	0
$F_{0\to4}$	102 kb/s	140 kb/s	53 kb/s	62 kb/s
$b_{3'}$	1	1	1	1
$b_{2'}$	0	0	0	0
$b_{1'}$	0	0	0	0
$F_{4\to0}$	26 kb/s	68 kb/s	34 kb/s	44 kb/s

Table 5.3: Measurements of the effect of EZ-flow on: (i) the median queue occupancy at the relay nodes and (ii) the end-to-end throughput of the 4-hop flows $F_{0\to4}$ and $F_{4\to0}$.

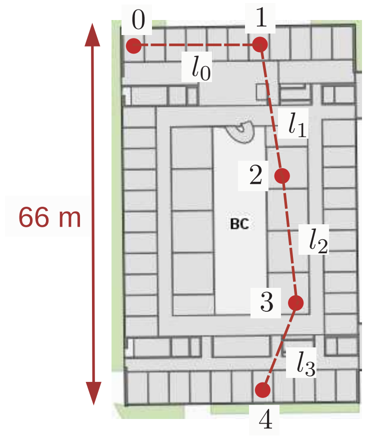


Figure 5.5: Illustration of the deployment used in Section 5.3.4.

obtain a throughput of: (i) 411 kb/s for flow $F_{0\to4}$ (379 kb/s with RTS); and (ii) 206 kb/s for flow $F_{4\to0}$ (172 kb/s with RTS). We then launch both flows simultaneously for 600 s and our results are summarized in Figure 5.6 and Table 5.3.

Table 5.3 shows the following for each flow: (i) the median queue occupancy from measurements taken each second (we set the buffer size limit to 100 packets); and (ii) the average end-to-end throughput. The results indicate that, either with or without RTS, the use of EZ-flow reduces the queue size and increases the end-to-end throughput for both flows. Moreover the results show that, in our setting, the performances are better without the use of RTS, and this also corresponds to the case where EZ-flow provides the largest performance gain. We show, in Figure 5.6, the evolution through time of the queue b_2 with and without EZ-flow. Finally, we stress that the Madwifi constraint is the reason that the queue b_1 does not reach a lower value with EZ-flow (i.e., the contention window of node 0 is set to the maximal working value of 2^{10}).

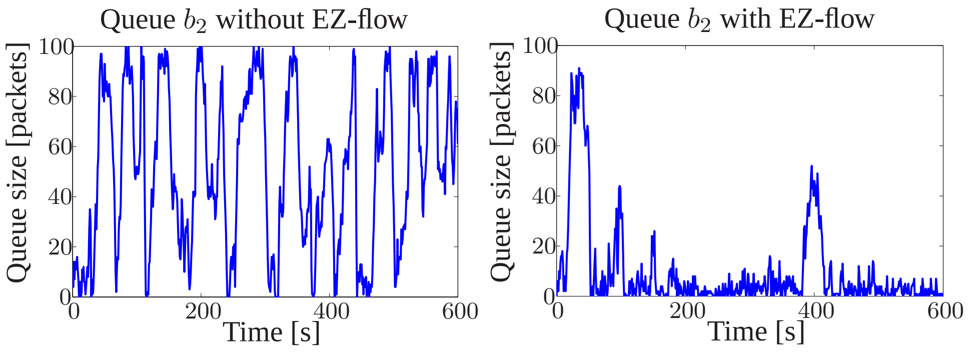


Figure 5.6: Effect of EZ-flow on the queue evolution through time of b_3.

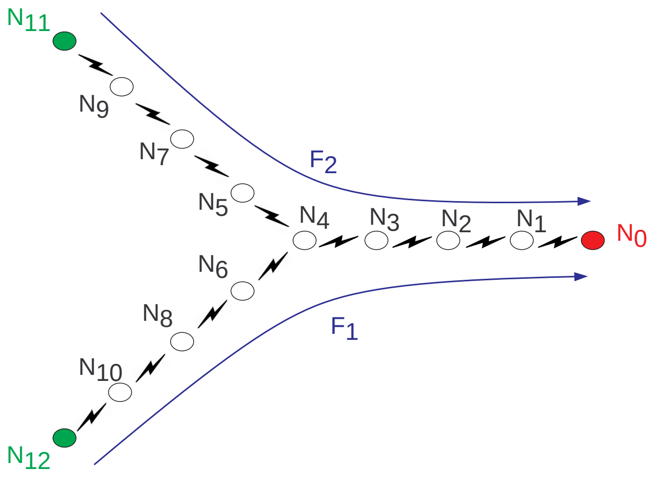


Figure 5.7: Scenario 1: 2-flows topology.

5.4 Simulation Results

To confirm our statement that the EZ-flow mechanism successfully achieves network stability and adapts to changing traffic matrices, in this section we present simulation results on two different scenarios with varying traffic loads.

5.4.1 System Description

We implemented the two modules of EZ-flow, the BOE and CAA, in ns-2 simulator version 2.33 [MF]. Our implementation closely follows the description of Section 5.2, where each node does not use any global information, but uses only the information it can hear by sniffing the channel.

Beside the inclusion of EZ-flow, we keep the standard parameters of IEEE 802.11. Therefore we use a transmission range of 250 m, a sensing range of 550 m and the RTS/CTS mechanism turned off. The reason we do not use RTS/CTS is twofold: (i) the current implementations of the protocol disable the mechanism by default and (ii) enabling the RTS/CTS is useless in the standard case we consider where the area covered by the sensing range (550 m) is larger than the maximal area covered by RTS and CTS ($2 \cdot 250$ m). We also keep the default data rate of 1 Mb/s and the propagation model to be two-ray ground. To ensure that the systems run in a saturated mode, we generate at the source a Constant Bit Rate (CBR) traffic at a rate of 2 Mb/s. Finally we use the NOAH routing agent [J.Wb], which is a static routing agent, in order to focus on the influence of the MAC layer and to remove from our study the effect of route link failure and the overhead of routing messages. The parameters of EZ-flow are $b_{min} = 0.05$, $b_{max} = 20$ and $max_{cw} = 2^{15}$.

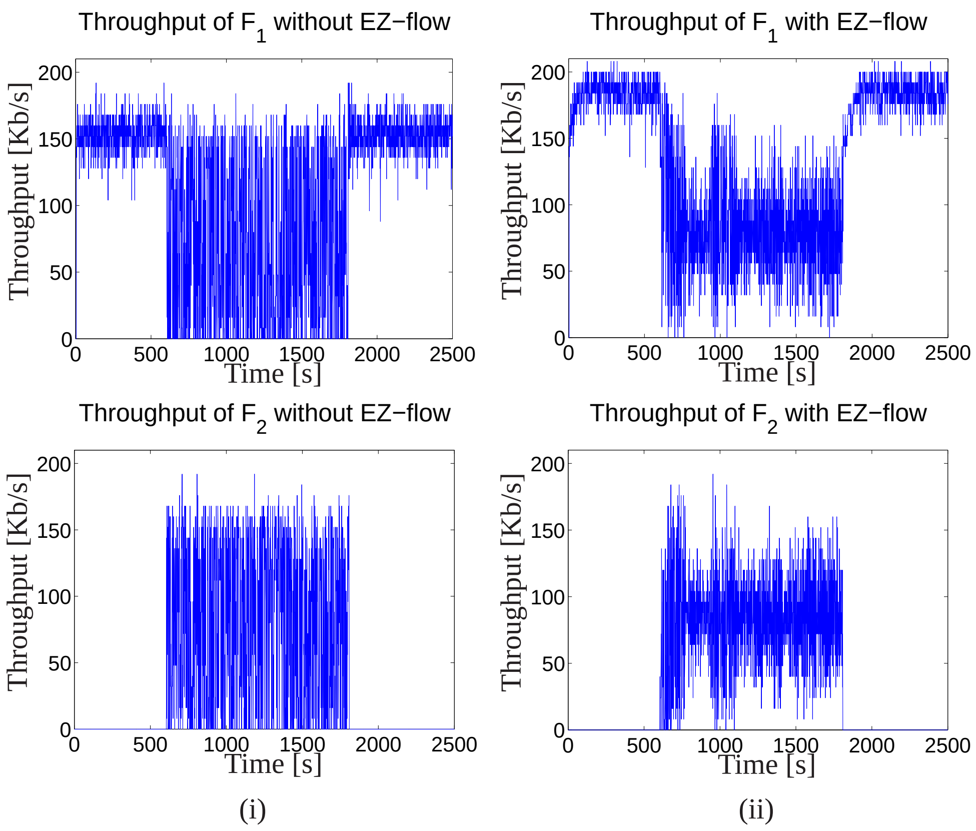


Figure 5.8: Throughput results for flow F_1 and F_2 in scenario 1: (i) with standard 802.11 and (ii) with EZ-flow turned on.

5.4.2 Scenario 1: 2-Flow Topology

The topology we study in our first scenario is depicted in Figure 5.7 and corresponds to two 8-hop flows that merge together to access a gateway. This situation corresponds to the uplink scenario happening in the backbone of WMNs, where different flows merge together to reach the gateway that delivers access to the Internet.

The flow F_1 is active for the entire duration of the simulation, i.e., from 5 s to 2504 s. Flow F_2 is active between 605 s and 1804 s. The throughput and delay results are shown respectively in Figures 5.8 and 5.9.

During the first period, the flow F_1 is alone in the network ($5 - 604$ s). We note that in the case of standard IEEE 802.11 without EZ-flow, the network already suffers from congestion. Indeed, the end-to-end delay reaches a value of 4.1 s, which is unacceptable for delay-sensitive traffic, and the throughput reaches only 153.2 kb/s. But when EZ-flow is turned on, the network is stabilized. Indeed, the end-to-end delays drop to a value as low as 0.2 s. Interestingly, this reduction in delay does not happen at the cost of a reduced throughput as it increases up to an av-

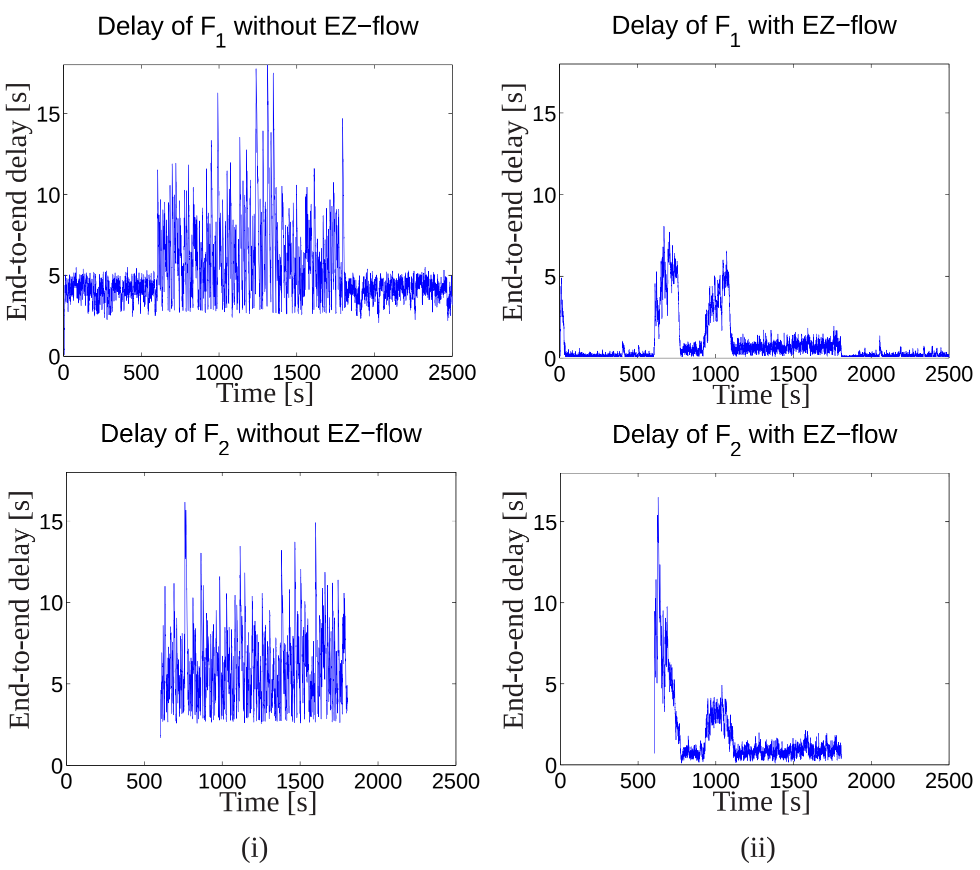


Figure 5.9: Delay statistics for flows F_1 and F_2 in scenario 1: (i) with standard 802.11 and (ii) with EZ-flow turned on.

erage of 183.9 kb/s, which corresponds to a throughput gain of 20% over standard 802.11. To understand why EZ-flow achieves this performance, Figure 5.10 shows how the contention windows are automatically adapted at the different nodes. The stable regime is reached once the relay nodes set their contention window to the minimal value of 2^4 and the source node, N_{12}, sets it to $cw_{12} = 2^7$. Therefore, we highlight that for the single-flow topology, EZ-flow reaches distributively the static solution that was proven to be stable (proposed in Section 3.5).

During the second period, both flows F_1 and F_2 are concurrently active ($605 - 1804$ s). Obviously, for IEEE 802.11 the congestion problem becomes worse with average delays as high as 5.8 s, an average throughput reduced to 76.5 kb/s and a high throughput variation. Enabling EZ-flow once again improves these three metrics, and most importantly solves the problem of congestion. Indeed, the end-to-end delay rapidly drops to negligible values, which shows no queue build-up in the network. Furthermore, the average throughput is also increased to 82.1 kb/s. The explanation for the two peaks in delay at around 600 s and 1000 s is found in

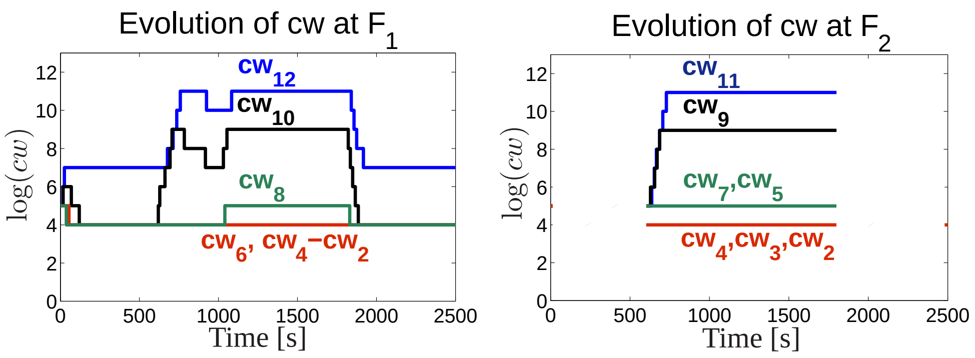


Figure 5.10: Illustration of how EZ-flow modifies the CW_{min} values at the different nodes of the network.

Figure 5.10. The first peak corresponds to the transient incurred by the arrival of flow F_2. Up to 605 s only flow F_1 exists in the network, and EZ-flow adapted the contention windows to stabilize the network for a single-flow topology. At 605 s the second flow F_2 appears in the network and therefore the previous contention windows are too small for this new traffic load. Thus, the queue starts to build up at some nodes and this is reflected by the sudden increase in end-to-end delay. Fortunately, EZ-flow rapidly adapts the contention windows to solve the problem and converges once again to a stable state. However, we note that after this first peak, the contention windows of the nodes in F_1 and F_2 are different as $cw_8 = 2^4$, whereas $cw_7 = 2^5$. This difference is the cause of the second peak. Indeed, due to the small cw_8, N_{10} and then N_{12} sense their successor node underutilized and thus become more aggressive. Unfortunately, this increase leads to a rate that is not supportable at the junction node N_4, and the queues of N_5 and N_6 start to build up. Both N_7 and N_8 detect this increase, but following the algorithm of the CAA, N_8 is more likely to react as $cw_8 < cw_7$. Therefore N_8 increases its cw_8, N_{10} and N_{12} react to it and reach a steady state. Interestingly, once the stable regime is reached, the source nodes set cw_{11} and cw_{12} at the value of 2^{11}, which is once again similar to the optimal static solution proposed in Section 3.5 ($q = 2^4/2^{11} = 1/128$).

During the last period, the flow F_1 is again alone in the network ($1805-2504$ s). As expected, IEEE 802.11 achieves performances similar to the first period. More importantly, the results show a particularly interesting property of EZ-flow: its adaptability to changes in the traffic load. Indeed, as soon as the flow F_1 leaves the network the buffer of some nodes becomes under-utilized. EZ-flow detects this and becomes more aggressive by decreasing the cw_{12}, cw_{10} and cw_8 until it reaches the same stable state as in the first period. Therefore improvements in throughput and delay similar to the first period are found for this last period.

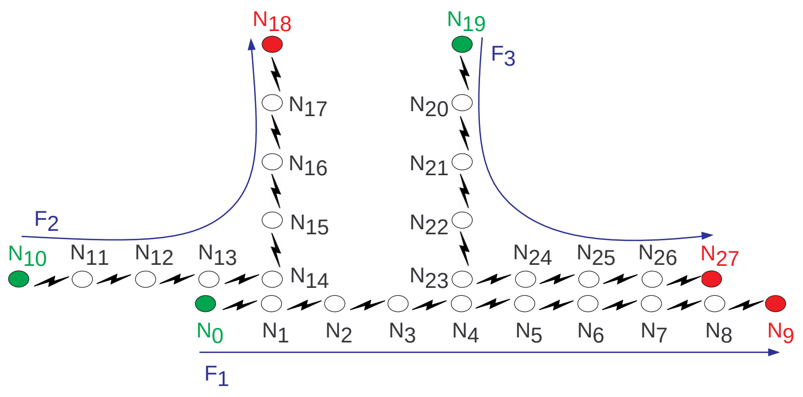


Figure 5.11: Scenario 2: 3-flows topology.

5.4.3 Scenario 2: 3-Flow Topology

The second scenario we consider is a 3-flow topology as depicted in Figure 5.11. This situation corresponds to the multi-hop scenario where multiple sources reach different destinations, but share the wireless resource with other flows on some parts of their paths. Furthermore, this topology illustrates what happens when the source of one flow (i.e., N_0) is a hidden node from another source (i.e., N_{10}). The simulation starts with flows F_1 and F_2 present in the network from 5 s to 1805 s. Then flow F_3 is added and the three flows share the resources from 1805 s to 3605 s. Finally, we remove flows F_2 and F_3 and let F_1 alone in the network from 3605 s to 4500 s, in order to check that the system stabilizes once again to a performance similar to what we find in the single-flow topology of scenario 1. The throughput and delay statistics are shown respectively in Figure 5.13 and Table 5.4. Furthermore, Figure 5.12 illustrates how EZ-flow adapts the contention windows over time.

During the first period, $[5, 1805)$, we see that IEEE 802.11 drastically suffers from the hidden node situation, with F_2 experiencing a particularly high delay (~ 15 s) and low throughput. The fairness index is 0.75. On the contrary, when EZ-flow is turned on, the contention window of the source of F_2 cw_{10} is increased up to a value of 2^{10} to provide a smooth flow. We note that this increase delivers negligible delays to both flows and does not penalize F_2 as it has a throughput that is even slightly higher than F_1. The reason F_2 achieves a higher throughput with a larger contention window ($cw_{10} = 2^{10}$ and $cw_0 = 2^5$) is that N_{10} only directly competes with two nodes (N_{11} and N_{12}), whereas N_0 competes with seven other nodes.

During the second period, $[1805, 3605)$, we see that IEEE 802.11 starves flow F_2 and F_3 in favor of F_1 and that all flows suffer from high delays. The reason that F_1 shows better performances than F_3 is that N_0 has many neighbors and it

	Mean throughput	Standard deviation	FI (Eq. (5.1))
F_1	145.6 kb/s	27.4 kb/s	0.75
F_2	39.9 kb/s	36.7 kb/s	
F_1	129.9 kb/s	45.3 kb/s	0.64
F_2	31.0 kb/s	32.5 kb/s	
F_3	27.3 kb/s	39.9 kb/s	
F_1	150.0 kb/s	13.0 kb/s	
F_1^{EZ}	89.9 kb/s	41.3 kb/s	1.00
F_2^{EZ}	100.3 kb/s	42.6 kb/s	
F_1^{EZ}	29.5 kb/s	22.9 kb/s	0.80
F_2^{EZ}	139.7 kb/s	23.0 kb/s	
F_3^{EZ}	135.4 kb/s	26.9 kb/s	
F_1^{EZ}	179.9 kb/s	13.5 kb/s	

Table 5.4: Mean throughput, standard deviation and Jain's fairness index (FI) with and without EZ-flow for the three periods: (i) F_1 alone, (ii) F_1 and F_2 active and (iii) all three flows active.

naturally reduces the source access rate and thus the queue build-up problem. IEEE 802.11 achieves a cumulative throughput of 188.2 kb/s and a fairness index of 0.64. In contrast, EZ-flow increases the cumulative throughput to 304.6 kb/s (a 62% throughput gain over 802.11), increases the fairness index to 0.8, and drastically reduces the end-to-end delay by an order of magnitude, at least. We note that F_1 has its throughput reduced even though the source of F_1, N_0, has cw_0 that is lower than cw_{10} and cw_{19} ($cw_0 = 2^7$ and $cw_{10} = cw_{19} = 2^9$). This reduction is due to the higher competition that F_1 experiences and it allows both F_2 and F_3 to have higher throughputs and all the flows to have negligible delays and thus, a stable network.

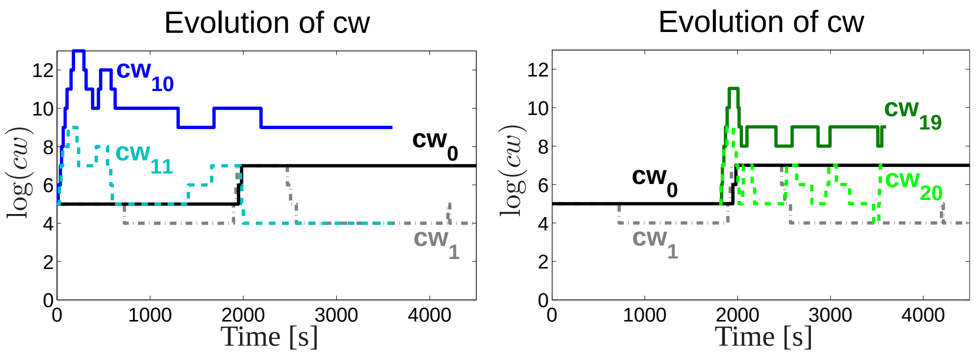


Figure 5.12: Illustration of how EZ-flow modifies the CW_{min} values at the two first nodes of each flow.

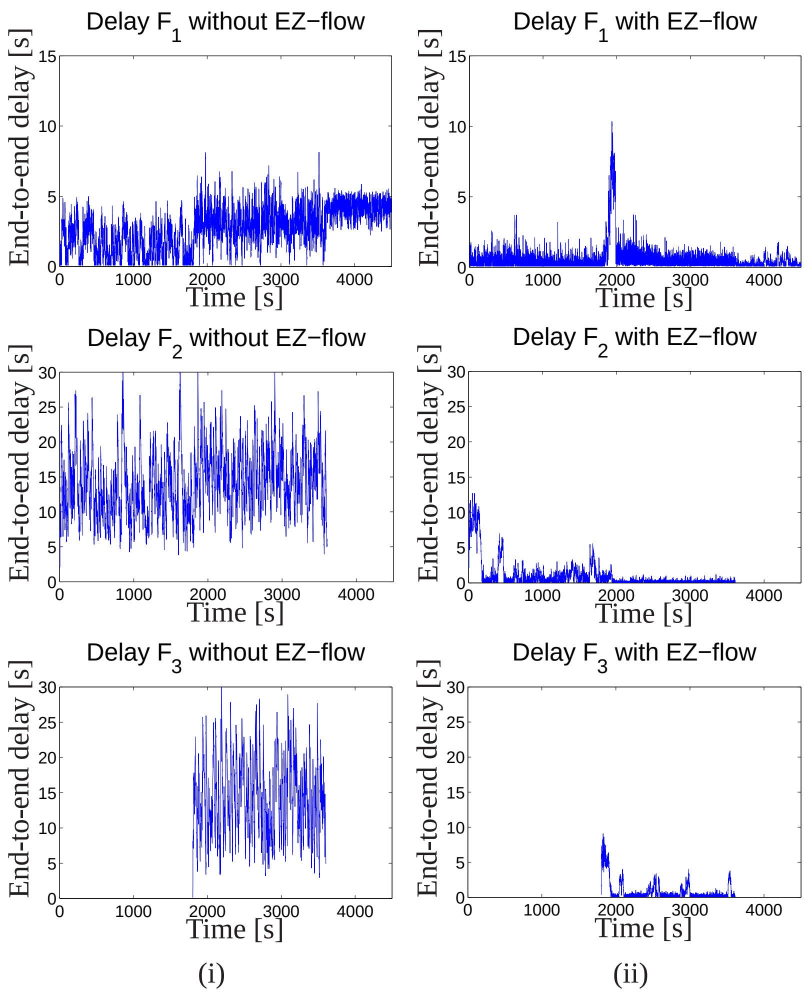


Figure 5.13: Delay statistics for flow F_1, F_2 and F_3 in scenario 2: (i) with standard 802.11 and (ii) with EZ-flow turned on.

Finally, during the last period we see that once again EZ-flow successfully detects the variation in traffic load and adapts the contention windows to achieve results similar to those in the single-flow case of scenario 1.

5.5 Dynamical Model

5.5.1 EZ-Flow Dynamics

Using the same notation as in Chapter 3, the dynamics of a network using EZ-flow are captured by the recursive equations

$$cw_i(n+1) = f(cw_i(n), b_{i+1}(n)) \tag{5.2}$$
$$b_i(n+1) = b_i(n) + z_{i-1}(n) - z_i(n), \tag{5.3}$$

where $f(\cdot,\cdot)$ is defined by

$$f(cw_i(n), b_{i+1}(n)) = \begin{cases} \min(cw_i(n)\cdot 2, max_{cw}) & \text{if } (b_{i+1}(n) > b_{max}) \\ \max(cw_i(n)/2, min_{cw}) & \text{if } (b_{i+1}(n) < b_{min}) \\ cw_i(n) & \text{otherwise,} \end{cases}$$

with b_{max} and b_{min} being, respectively, the maximal and minimal threshold values for the queue and $min_{cw} = 2^m$ and $max_{cw} = 2^M$ being the bounds between which the contention windows can evolve. Practical values are $m = 4$ and $M = 15$, thus we always take

$$M > m + 1.$$

This discrete-time model is a Markov chain with the tuple

$$\{\vec{b}(n), \vec{cw}(n)\}$$

as state, where

$$\vec{b}(n) \in \mathbb{N}^{K+1}$$

and where $\vec{cw}(n)$ satisfies both

$$cw_i(n) \in \{2^m, 2^{m+1}, \cdots, 2^M\}$$

and

$$cw_i(n) \geq 2^{m+\min(l, M-m)} \text{ when } b_{i+1}(n) > b_{max} + l, \tag{5.4}$$

where $l > 0$. The lower-bound condition (5.4) comes from the recursive application of (5.2) for the last l time slots. Indeed, $b_{i+1}(k) > b_{max}$ for $n - l < k \leq n$ implies that $cw_i(k+1) = \min(cw_i(k)\cdot 2, 2^M)$.

The state space is divided in 2^{K-1} regions, which differ by the entries of $\vec{b}$ that are zero and non-zero (i.e., the queues that are empty or not). Figure 3.7 illustrates these 8 regions for a 4-hop network (denoted A-H). In each region, one can compute first the possible outcomes of the back-off timers that depend on the contention values $\vec{cw}(n)$, and next the resulting transmission patterns that depend also on the possible collisions due to hidden terminals. The possible outcomes are obtained by following the same reasoning as in Section 3.4. We summarize them in Table 5.5 for the 4-hop network with a stealing effect $p = 1$ (i.e. no RTS/CTS).

Region	$\vec{z}$	$\mathbb{P}(\vec{z})$
A	$[1,0,0,0]$	1
B	$[1,0,0,0]$	$cw_1/(cw_0+cw_1)$
	$[0,1,0,0]$	$cw_0/(cw_0+cw_1)$
C	$[0,0,1,0]$	1
D	$[0,1,0,0]$	$\frac{cw_0cw_2}{\sum_{i=0,1,2}\prod_{j\neq i}cw_j}$
	$[0,0,1,0]$	$1-\frac{cw_0cw_2}{\sum_{i=0,1,2}\prod_{j\neq i}cw_j}$
E	$[1,0,0,1]$	1
F	$[0,0,0,1]$	$cw_0/(cw_0+cw_1)$
	$[1,0,0,1]$	$cw_1/(cw_0+cw_1)$
G	$[0,0,1,0]$	$cw_3/(cw_2+cw_3)$
	$[1,0,0,1]$	$cw_2/(cw_2+cw_3)$
H	$[0,0,1,0]$	$\frac{cw_0cw_1cw_3}{\sum_{i=0,1,2,3}\prod_{j\neq i}cw_j}+\frac{cw_1cw_2cw_3}{\sum_{i=0,1,2,3}\prod_{j\neq i}cw_j}\frac{cw_3}{cw_2+cw_3}$
	$[0,0,0,1]$	$\frac{cw_0cw_2cw_3}{\sum_{i=0,1,2,3}\prod_{j\neq i}cw_j}+\frac{cw_0cw_1cw_2}{\sum_{i=0,1,2,3}\prod_{j\neq i}cw_j}\frac{cw_0}{cw_0+cw_1}$
	$[1,0,0,1]$	$\frac{cw_1cw_2cw_3}{\sum_{i=0,1,2,3}\prod_{j\neq i}cw_j}\frac{cw_2}{cw_2+cw_3}+\frac{cw_0cw_1cw_2}{\sum_{i=0,1,2,3}\prod_{j\neq i}cw_j}\frac{cw_1}{cw_0+cw_1}$

Table 5.5: Probability of occurrence of the transmission pattern $\vec{z}$ for the different regions of the space $\mathbb{N}^3$.

5.5.2 Proof of Stability

Equipped with the model described above, we now formally prove the efficiency of EZ-flow in stabilizing the network. We give a proof, which holds when

$$b_{min} > M - m + 1. \tag{5.5}$$

This condition further reduces the state space of our model as, following a similar recursive argument than for (5.4), it implies that

$$cw_i(n) = 2^m \text{ when } b_{i+1}(n) = 0. \tag{5.6}$$

When $b_{min} \leq M - m + 1$, the proof uses computer-assisted computations, and is given in [ASTEF09].

Theorem 5.1 *EZ-flow stabilizes a* 4*-hop network by maintaining almost surely finite the queues of all the relaying nodes.*

Proof: We apply Foster's theorem (see Appendix A) with the Lyapunov function

$$h(b_1, b_2, b_3, cw_0, cw_1, cw_2, cw_3) = b_1 + b_2 + b_3,$$

and the finite set

$$S = \{cw_0, cw_1, cw_2, cw_3 \leq 2^M; 0 \leq b_1, b_2, b_3 \leq b_{max} + M - m + 3\}.$$

We need to verify that both conditions (A.1) and (A.2) of this theorem are verified for all points $\{\vec{b}(n), \vec{cw}(n)\}$ within the state space.

We note first that (A.1) is satisfied by the definition of h and the non-zero transition probabilities of the random walk.

It takes some more work to verify (A.2). One needs to compute

$$\epsilon_{k,\vec{b}}(n) = \mathbb{E}\left[h(\vec{b}(n + k(\vec{b}(n))))|\vec{b}(n)\right] - h(\vec{b}(n))$$

for all possible $\vec{cw}$ and with $\vec{b}(n)$ in each of the 7 regions B-H outside S, similarly to the proof of Theorem 2.

First, we note that the transition probabilities from Table 5.5 imply that:

$$\epsilon_{1,\vec{b}}(n) > 0 \text{ for } \vec{b}(n) \in B,$$

$$\epsilon_{1,\vec{b}}(n) < 0 \text{ for } \vec{b}(n) \in F \cup H,$$

$$\epsilon_{1,\vec{b}}(n) = 0 \text{ otherwise.}$$

Then, we find that after some computations that for all $\vec{cw}$, (A.2) is verified. In regions F and H, we directly have from Table 5.5 that

$$k(\vec{b}(n)) = 1 \text{ when } \vec{b}(n) \in F \cup H.$$

In regions D and E, we note that there is a strictly positive probability of having $\vec{b}(n+1) \in F \cup H$ and a zero probability of having $\vec{b}(n+1) \in B$. Therefore, we derive that

$$k(\vec{b}(n)) = 2 \text{ when } \vec{b}(n) \in D \cup E.$$

In region G, we see that there is a strictly positive probability of having $\vec{b}(n+1) \in D \cup H$ and a zero probability of having $\vec{b}(n+1) \in B$. Thus, this gives us that

$$k(\vec{b}(n)) = 3 \text{ when } \vec{b}(n) \in G.$$

In region C, there is a probability 1 of having $\vec{b}(n+1) \in G$. Hence, we conclude that

$$k(\vec{b}(n)) = 4 \text{ when } \vec{b}(n) \in C.$$

For region B, the demonstration is a little more complex. First, we use that for $\vec{b}(n) \in B \setminus S$, we have

$$b_1(n) > b_{max} + M - m + 3$$

and

$$b_2(n) = b_3(n) = 0.$$

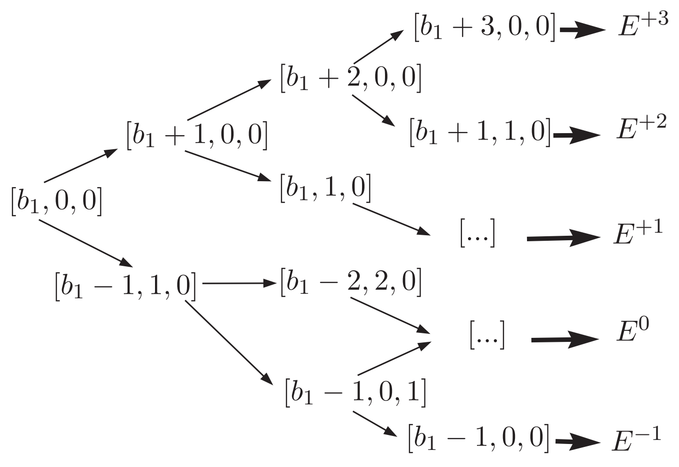


Figure 5.14: Tree representing all possible transitions at steps $n + 1$, $n + 2$ and $n + 3$ starting from $b(n) \in B \setminus S$. The five possible resulting events are are E^{+3}, E^{+2}, E^{+1}, E^{0}, E^{-1}; where $E^x = E^x(\vec{b}(n))$ is the event that $h(\vec{b}(n + 3)) - h(\vec{b}(n)) = x$.

Thus, it follows from (5.4) and (5.6) that

$$c\vec{w}(n) = [2^M, 2^m, 2^m, 2^m] \text{ for } \vec{b}(n) \in B \setminus S.$$

Next, we obtain $\epsilon_{3,\vec{b}}(n)$ by defining $E^x(\vec{b}(n))$ as the event that

$$h(\vec{b}(n + 3)) - h(\vec{b}(n)) = x.$$

Then, we compute the probabilities for the five possible events $E^{+3}(\vec{b}(n))$, $E^{+2}(\vec{b}(n))$, $E^{+1}(\vec{b}(n))$, $E^{0}(\vec{b}(n))$, and $E^{-1}(\vec{b}(n))$ (see Figure 5.14). We obtain that

$$\begin{array}{rcl}
\mathbb{P}(E^{+3}) & = & 1/(1 + 2^{M-m})^3 \\
\mathbb{P}(E^{+2}) & = & 1/(1 + 2^{M-m})^2 - 1/(1 + 2^{M-m})^3 \\
\mathbb{P}(E^{+1}) & = & 1/(1 + 2^{M-m}) - 1/(1 + 2^{M-m})^2 \\
\mathbb{P}(E^{-1}) & = & 2^{M-m}/(1 + 2^{M-m}) \cdot \\
 & & (1 - 2^{M-m}/(2 \cdot 2^{M-m} + 1)) \cdot \\
 & & 2^{M-m}/(1 + 2^{M-m}).
\end{array}$$

Then, we find that

$$\epsilon_{3,\vec{b}}(n) = 3 \cdot \mathbb{P}(E^{+3}(\vec{b}(n))) + 2 \cdot \mathbb{P}(E^{+2}(\vec{b}(n))) + \mathbb{P}(E^{+1}(\vec{b}(n))) - \mathbb{P}(E^{-1}(\vec{b}(n))),$$

and because $M - m > 1$, we have that

$$\epsilon_{3,\vec{b}}(n) < 0.$$

Thus

$$k(\vec{b}(n)) = 3 \text{ satisfies (A.2) for } \vec{b}(n) \in B \setminus S.$$

Finally, as Region $A \subseteq S$, the conditions of Foster's theorem are satisfied in all $\{\vec{b}(n), c\vec{w}(n)\}$ within the state space, and it proves that EZ-flow stabilizes the network.

□

5.6 Concluding Remarks

In this chapter, we proposed and designed EZ-flow, a new flow control mechanism for IEEE 802.11 WMNs. EZ-flow is fully backward-compatible with the IEEE 802.11 standard and works without any form of message passing. EZ-flow is implemented in a distributed fashion as a simple program running at each relay node. It takes advantage of the broadcast nature of the wireless medium to passively estimate the queue occupancy at a successor node. The minimum congestion window parameter is adapted at each relay node based on this estimation to ensure a smooth flow, specifically, each relay node adapts its contention window to avoid queue build-up at its successor node.

We demonstrated by experiments the attendant benefits of EZ-flow on a testbed composed of 9 standard wireless mesh routers deployed over 4 different buildings. Our measurement results show that EZ-flow simultaneously improves throughput and fairness performance. To our knowledge, this is among the first implementations of an algorithm addressing instability in a real multi-hop network.

We have also thoroughly evaluated the dynamic behavior of EZ-flow by using ns-2 simulation. The results show that EZ-flow quickly adapts to changing traffic loads and ensures end-to-end delays much lower than standard IEEE 802.11 WMNs.

Finally, we derived a Lyapunov function with which we analytically proved the stability of an IEEE 802.11-based linear 4-hop topology implementing EZ-flow.

Chapter 6

Joint Congestion-Control and Fairness

6.1 Background

6.1.1 Problem Statement

The root cause for congestion in WMNs is the Medium Access Control (MAC) protocol. Indeed, WMNs typically use distributed MAC protocols (e.g. CSMA/CA) that have been proved, in Chapter 3, to suffer from congestion when no countermeasure is applied. In wired networks, the queuing policy is the key factor for unfairness. A well-known solution is to use *fair queuing*: one queue per-flow is maintained, combined with a round-robin scheduler [DKS89, PG93]. However, in wireless networks, fair queuing is required but not sufficient by itself to ensure fairness among flows. Indeed, ensuring fairness depends on both the MAC and the queuing policy [GSK04]. In Figure 6.2, we depict experimental results that show how fair queuing fails to achieve fairness when a 1-hop flow competes with a 3-hop flow, even if it achieves fairness when a 1-hop flow competes with a 2-hop flow[1].

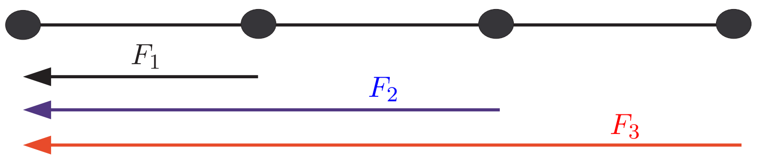


Figure 6.1: Linear wireless topology for which Fair Queuing achieves max-min fairness when a 1-hop flow F_1 and a 2-hop flow F_2 transmit concurrently, but it fails to do so when F_1 transmits with a 3-hop flow F_3 (see Figure 6.2).

[1]Demo available at: http://icawww1.epfl.ch/NetController/ (Video 1)

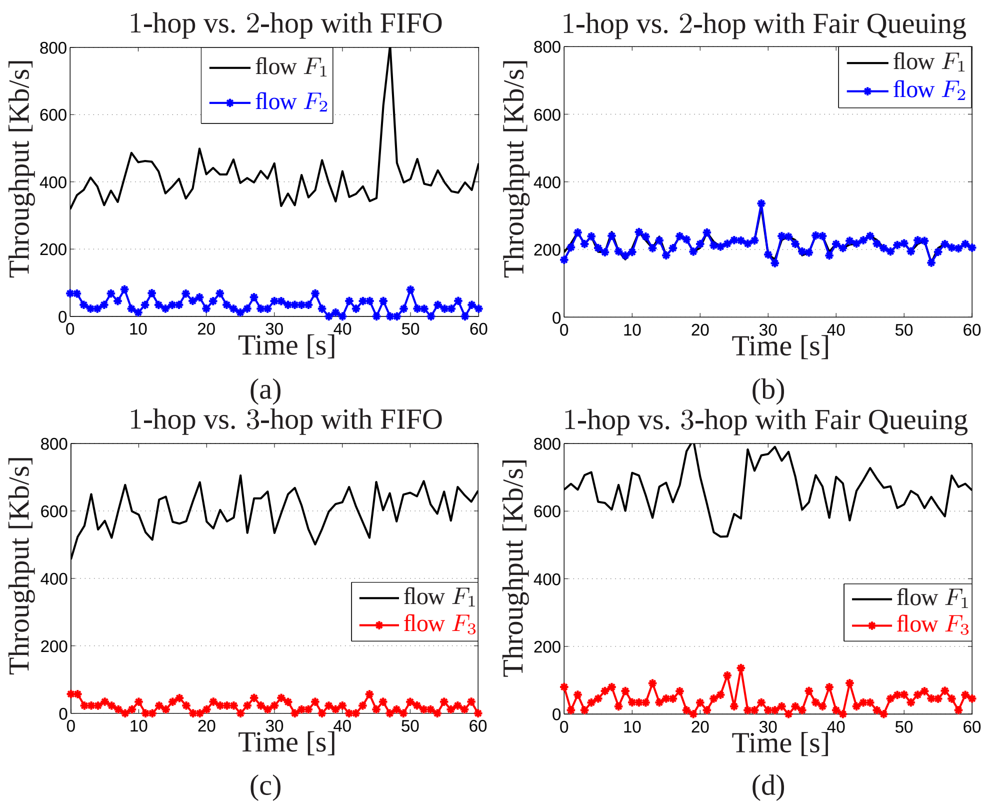


Figure 6.2: End-to-end throughput in a line topology (see Figure 6.1) with a single-hop and a multi-hop UDP flow (2- or 3-hop) with the standard FIFO policy (left) and with fair queuing (right). Fair queuing achieves max-min fairness for the 2-hop case (see video[1]), but fails to prevent starvation in the 3-hop case. Results in the top-right picture (b) look identical, but it is only an artifact of the scale of the picture. Smaller scale plots show differences.

Our goal in this chapter is to address both intra-flow congestion and inter-flow fairness for the backhaul of a WMN. In addition, we have several design objectives. We want our solution to be backward-compatible with existing hardware so that it can be readily deployed in existing networks. For instance, we want to avoid modifying any parameter of the MAC layer. We also want our solution to be distributed and to minimize message-passing.

Our approach is to first develop mechanisms that solve the intra-flow and the inter-flow problem separately. Hence, our first two sub-problems are as follows:

1. For a given flow, how do we efficiently perform intra-flow congestion control without message passing and without interacting with any parameter of the MAC layer in a dynamic network (i.e., with time-varying traffic demands and link capacities)?

2. How do we efficiently achieve inter-flow fairness when the traffic demands and link qualities are unknown and time-varying?

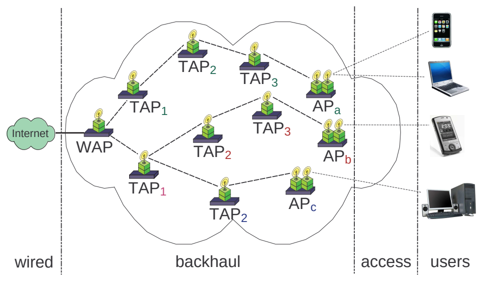


Figure 6.3: Illustration of a wireless mesh network, where a packer is encapsulated with an IP-in-IP header when entering and leaving the *backhaul* section.

Finally, we efficiently combine both the intra-flow congestion control and inter-flow fairness mechanisms.

6.1.2 System Model and Assumptions

We consider the single-channel multi-hop backhaul network of a WMN (see Figure 6.3). There is one Internet gateway in the WMN. Each node supports two wireless interfaces where one is configured as an Access Point (AP) for clients and the second interface belongs to the backhaul. We assume that the AP wireless interface (i.e., the access part of the mesh) runs on a channel orthogonal to the backhaul and therefore we will not consider the access part in our analysis hereafter. The gateway routes traffic to and from the Internet. Hence, all client traffic at the APs is forwarded to the Gateway. We assume that all nodes use the same MAC layer (i.e., IEEE 802.11).

There is a Limited Number of Flows in the WMN Backhaul

In this topology, we define a flow as the tuple $<$*src IP; dst IP* $>$ between a source AP and a destination AP. This is a pragmatic definition of a flow stemming from the observation that most traffic in the backhaul is going to/coming from the gateway. Hence the number of flows is $O(N)$, where N is the number of nodes within the backhaul of the mesh. Typical values of N are below 50 [ROO, BOW]. Practically, this is realized by performing an IP-in-IP encapsulation when a packet enters the backhaul and a decapsulation at the gateway. When client traffic enters the mesh at an AP, the outer IP header source address is the IP address of the AP and the outer IP header destination address is the IP address of the gateway. The inner IP header is the original header. Therefore, the number of flows that we consider does not explode, even though the final source/destination might be any address within the Internet.

Rate of a Flow and Capacity Region

Assuming F flows in the backhaul indexed from 1 to F, we denote the rate of a flow j, $j \in \{1, \ldots, F\}$ by x^j. Additionally, we denote the flow rate vector by

$$\vec{x} = [x^1 \; x^2 \; \ldots \; x^F].$$

Observe that, given our definition of flows in the previous paragraph, x^j is the rate achieved above the MAC layer.

Finally, we denote by Λ the network capacity region, which is the set of all achievable rate vectors $\vec{x}$.

Intra-Flow and Inter-Flow Performance Issues

Performance problems with decentralized CSMA/CA protocols, e.g. IEEE 802.11, in wireless multi-hop networks can be divided into two categories:

- *Intra-flow Congestion Problem*: Already with a single flow, network instability can occur. The queues of the relay nodes build up, reducing the throughput and increasing end-to-end delays. Indeed, we showed in Chapter 3 and 5 that the intra-flow congestion is due to the inefficiency of decentralized CSMA/CA protocols to form a smooth flow of packets when transporting the traffic hop-by-hop through the network.

- *Inter-flow Fairness Problem*: With two or more flows, interference between flows can lead to serious unfairness and starvation problems if no counter-policy is applied. This occurs with TCP in simple topologies where one or more 1-hop flows compete with one larger K-hop flow ($K \geq 2$) [GMSK09]. In Figure 6.2, our experiments show that unfairness and starvation are not limited to TCP and that they also occur with UDP. In these experiments, a 1-hop flow competes with a 2-hop (or 3-hop) flow. We observe that fair queuing is clearly needed to achieve fairness, but is not sufficient by itself.

Tradeoff between Fairness and Throughput

There is a tradeoff between (i) maximizing the total throughput and (ii) fairly sharing the capacity among competing flows. This tradeoff is intimately related to the maximization of a utility function $u(\cdot)$ of the flow rates [ES05].

For instance, Remember that x^j is the rate of flow j: maximizing

$$u_{thr}(\vec{x}) = \sum_{j=1}^{F} x^j \tag{6.1}$$

yields the maximum total throughput but completely ignores fairness. A utility function effectively balancing throughput and fairness is

$$u_{prop}(\vec{x}) = \prod_{j=1}^{F} x^j. \tag{6.2}$$

This utility function achieves *proportional fairness* [KMT98], because maximizing (6.2) is exactly the same as maximizing

$$u(\vec{x}) = \sum_{j=1}^{F} \log(x^j).$$

In the remainder of the chapter, we will consider the utility functions (6.1) and (6.2). The reader interested in further extensions of proportional fairness can consult [MW00].

Then, for a given utility function $u(\cdot)$ (i.e., either (6.1) or (6.2)), solving the optimization problem

$$\max_{\vec{x}\in\Lambda} u(\vec{x}) \tag{6.3}$$

finds a rate allocation vector $\vec{x}$ that maximizes the utility function and satisfies the particular fairness implicitly embedded by the definition of $u(\cdot)$. Note that directly solving the maximization problem (6.3) in one step would imply the knowledge of the capacity region Λ. However, the capacity region is, in practice, time-varying and challenging to measure [SSGG09] and therefore there is a real need for dynamic mechanisms such as the one that we propose in this chapter.

6.1.3 Related Work

As explained in Section 6.1, the serious unfairness and starvation problems that we observe in Figure 6.2 find their origins in both: (i) a queuing problem and (ii) a MAC problem. The queuing problem has been thoroughly studied for wired networks [DKS89, PG93]: flow-based scheduling with Weighted Fair Queuing (WFQ) efficiently provides fairness. However, WFQ alone is not sufficient for wireless multi-hop networks [GSK04] because the MAC layer plays a critical role.

In the previous chapters, we already discussed the recent analytical work on throughput optimal schemes [YST08, CKLS08, PYC08, YPC08, GLS07, SSR09]. Their goal is to achieve any rate in the capacity region by using variants of the MaxWeight scheduling algorithm [PYC08, SSR09]. It is important to point out that these approaches rely on three fundamental conditions: (i) the set of active flows in the network must be static and no flow can appear (or leave) [vdVBS09], (ii) all sources should know a priori the capacity region and (iii) no source can transmit at a rate above capacity. Our work fundamentally differs because we do not assume that the sources know the capacity region nor that they rate-limit themselves at the capacity of the network. Our mechanisms do not rely on any assumption about the source behavior and provide both congestion control and fairness in a distributed manner.

Interestingly, in [ES05] Eryilmaz et al. show for single-hop cellular networks that a combination of queue-length-based scheduling and congestion control leads to a fair resource allocation. Our work differs as we consider multi-hop scenarios that require distributed solutions because there exists no base stations that can act as a central scheduler.

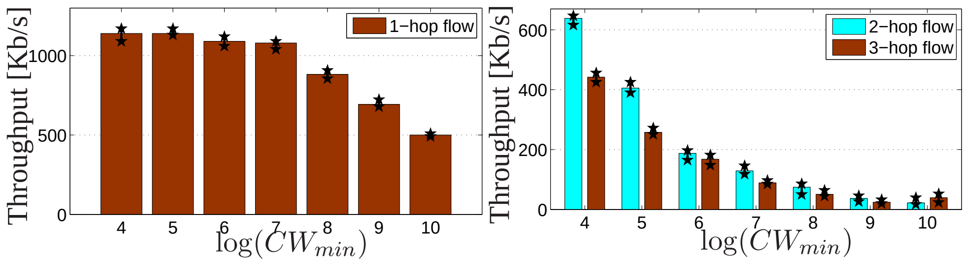


Figure 6.4: On the left, an experiment with UDP shows that increasing the contention window decreases the link capacity (1-hop). On the right, we see the end-to-end throughput of a single multi-hop UDP flow (either 2-hop or 3-hop). In either case, increasing the contention window on *the node directly connected to the gateway* (i.e., last hop) decreases the achieved UDP throughput.

Several practical solutions have also been proposed to address the unfairness and starvation problems. In [GSK04], Gambiroza et al. introduce the Inter-TAP Fairness Algorithm (IFA) that achieves a fair allocation. Despite encouraging performances, this solution requires a large amount of network-wide message passing, which is particularly problematic in dynamic scenarios. Indeed, all nodes compute their offered load and capacity, and transmit this information to all the other nodes in the network. The work in [GMSK09] by Gurewitz et al. is probably the most related to ours. The authors identify a starvation problem that occurs when a 2-hop TCP flow competes with one or more 1-hop TCP flows. Their proposed counter-starvation policy consists in "*increasing the contention window of all the nodes directly connected to the gateway*". This policy definitively provides fairness improvements but it is topology-dependent and suffers two serious pitfalls. First, in Figure 6.4, we show that increasing the congestion window (CW_{min}) seriously affects the link capacity. An increase to $2^7 - 2^8$ as proposed in [GMSK09] corresponds to a $10 - 20\%$ decrease in the link capacity. Now, in mesh networks, the nodes directly connected to the gateway are often the network bottlenecks as they collect all downstream traffic. Thus, this counter-starvation policy may reduce bottleneck link capacities and therefore affect the whole network. Second, the counter-starvation policy only works with TCP, as it provides congestion-control. Following this counter-starvation policy on 2-hop or 3-hop topologies with UDP will drastically reduce throughput: The source node of any multi-hop flow will aggressively access the channel and the last hop will not be able to transmit packets to the gateway due to its higher contention window (Figure 6.4, right). Our algorithm differs because it neither reduces the bottleneck link capacity, nor needs the help of any congestion-control mechanism from the upper-layers.

Finally, additional work implements algorithms to achieve fairness and congestion control [RJJP08, SSGG09, WJHR09]. In [RJJP08], Rangwala at al. propose a transport layer congestion control algorithm called WCP that explicitly reacts to congestion without suffering the unfairness problem of TCP. Similarly, [SSGG09] Salonidis et al. introduce a rate-control protocol at the network layer, which esti-

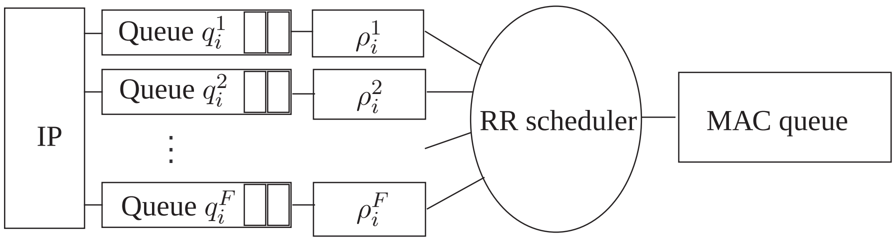


Figure 6.5: Congestion control mechanism running at node i, with each flow j having its dedicated queue and rate limiter ρ_i^j (limiting the link rate). A round-robin (RR) scheduler connects the rate limiters to the MAC (interface) queue.

mates the network capacity and adapts the transmission rate accordingly to avoid congestion. Nevertheless, both protocols build upon an end-to-end methodology, better suited for a static network than a dynamic one. Hop-by-hop approaches have the potential to provide better performance [YS07]. Both DiffQ [WJHR09] and EZ-flow (see Chapter 5) are hop-by-hop protocols that perform congestion control at the MAC layer by using different values for the contention window parameter (CW_{min}). Despite good performance, DiffQ and EZ-flow abuse the role of the CW_{min}, originally meant to deal with contention and not congestion. This abuse leads to antagonist goals between congestion (avoiding queue build-up) and contention (avoiding collisions) whenever a part of the network suffers from both. Indeed, the congestion-control mechanism tends to decrease CW_{min} in order to to flush the queue faster, whereas the contention-control mechanism tends to increase CW_{min} to avoid collisions. Congestion-control must be decoupled from contention-control at the MAC layer. Similarly to the methodology of EZ-flow, discussed in previous chapter, our network-layer hop-by-hop congestion-control algorithm works without message passing, but it significantly differs from the scheme of the previous chapter, because there are no interactions with the MAC layer. Moreover, here we also consider the notion of fairness through an inter-flow mechanism.

6.2 Intra-Flow Congestion Control

For a single multi-hop flow, the goal of intra-flow congestion control is to create a smooth packet flow with low end-to-end delays and high throughput. We use a hop-by-hop approach where the link rates of the source and its relay nodes are adapted to maintain a small (but non-zero) number of packets in the relay queues. Obviously, the last node just before the destination is not rate limited.

In this chapter, we design a novel layer 2.5 protocol performing per-flow queuing at each node i. As depicted in Figure 6.5, each queue is attached to a *rate*

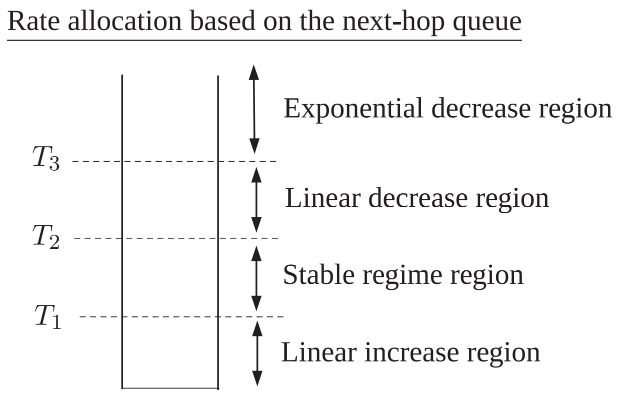


Figure 6.6: Mechanism used by the network-layer congestion-control scheme to adapt the rate ρ_i^j of the rate limiter based on the next-hop queue $\bar{q}_{s(i)}^j$.

limiter that limits the link rate of flow j to ρ_i^j. Finally, each rate limiter is scheduled in a Round-Robin (RR) manner to the unmodified MAC queue. In addition, q_i^j denotes the number of packets that are contained in the queue of flow j at node i and the index of the next-hop (or successor) of node i for flow j is $s^j(i)$. We will drop the index j if it is clear from the context. For instance, $q_{s(i)}^j$ is the size of the queue for flow j at the next-hop of node i.

For each flow j at node i, our algorithm sets the rate ρ_i^j according to the size $q_{s(i)}^j$ of the next-hop queue. Hence, our solution comprises two phases: (i) a passive estimation of $q_{s(i)}^j$, without message-passing, and (ii) the adaptation of ρ_i^j.

The first phase follows a methodology similar to the one described for the BOE of EZ-Flow in Section 5.2.2 (with the advantage of not requiring two wireless interfaces anymore). For each flow j, each node i maintains a packet-identifier list (e.g., UDP or TCP checksums) of the last P successfully transmitted packets (typically, $P = 100$). In addition, each node runs in the promiscuous mode and attempts to overhear packets forwarded by the *next-hop* node $s^j(i)$. Whenever a forwarded packet is overheard, node i can use the packet-identifier list to compute an estimate of the occupancy $q_{s(i)}^j$ of the next-hop queue. Node i simply counts how many packet identifiers have been added to the list since the identifier of the overheard packet was added. This method gives an exact value when the next-hop node uses the standard FIFO queuing policy.

For each flow j at node i, the second phase uses the estimates of $q_{s(i)}^j$ to adapt ρ_i^j. For a flow j, an update of the rate limiter parameter ρ_i^j is performed every R packets that are overheard from $s^j(i)$. Let $T_1 < T_2 < T_3$ denote queue thresholds and $\bar{q}_{s(i)}^j$ the per-flow time-averaged occupancy of $q_{s(i)}^j$ computed over the last R overheard packets. Whenever an update occurs *one* of this four cases takes place (see Figure 6.6):

1. $\bar{q}^j_{s(i)} \leq T_1$**:** The queue at the next-hop is under-utilized and should be increased (positive expected drift). Thus, if the node has packets in its own queue, ρ^j_i is linearly increased.

2. $T_1 < \bar{q}^j_{s(i)} < T_2$**:** The queue at the next-hop is neither empty nor overflowing. This is a desirable situation and ρ^j_i should remain unchanged (zero expected drift).

3. $T_2 \leq \bar{q}^j_{s(i)} \leq T_3$**:** The queue at the next-hop builds up and ρ^j_i should be decreased (negative expected drift). As $\bar{q}^j_{s(i)} \leq T_3$, a small decrease of ρ^j_i might be enough to maintain a reasonable number of packets at node i: ρ^j_i is linearly decreased.

4. $T_3 > \bar{q}^j_{s(i)}$**:** The next-hop queue is close to overflowing (e.g., due to a sudden environmental change) and ρ^j_i should be quickly decreased to avoid packet losses (large negative expected drift): ρ^j_i is multiplicatively decreased.

The role of T_1, T_2 and T_3 is to describe the number of packets that need to be maintained in the queues. They depend only on the buffer size of these queues. Typically, these are fixed and well-known.

Finally, in order to avoid the complete starvation of a flow, we do not allow ρ^j_i to go below the minimal value of 1 packet per second. This is necessary for the nodes to estimate the next-hop queue occupancies at any time. The parameter R represents a tradeoff between reactivity and stability: a large R fits a highly stable environment and smooths short-term variations. On the contrary, smaller R values are better for highly time-varying environments that require a quick reactivity of the protocol. In our experiments, we always set $R = 40$.

6.3 Inter-Flow Fairness

The intra-flow congestion control mechanism is required in order to avoid the queues of relay nodes from overflowing. Nevertheless, it is not enough, by itself, to ensure fairness between different competing flows. In fact, this problem is particularly serious when a 1-hop flow competes with other multi-hop flows. For instance, let a 1-hop and a 2-hop flow send traffic to a gateway. The 2-hop flow will be rate limited by the congestion control mechanism (in order to avoid that the queue of the relay builds-up), but the 1-hop flow will not be rate limited, because no information about the next-hop queue can be used for intra-flow congestion control.

We propose a solution for networks with a tree topology, the typical configuration for a mesh network where all traffic is directed to and from the gateway. The gateway has a global view of the throughput achieved by the different flows, and, in particular, of the fairness of their allocated rates. It can therefore use this information to set the rate of the rate limiters of its directly connected neighbors. A network with multiple gateways is a straightforward extension. Indeed, the multiple gateways can use their wired links to exchange information, thus making the problem similar to the case of a macro-gateway making the scheduling decisions for the whole network.

We believe that it is relevant to consider this fairness problem separately, in a single-hop scenario. Indeed, any rate adaptation on the last hop induced by a fairness enforcement policy will propagate along the mesh using the intra-flow congestion control mechanism, as discussed in Section 6.4. The experimental results in Section 6.5 confirm this behavior.

6.3.1 Model Description

We focus on a single-hop scenario, where F flows j ($j \in \{1, \dots, F\}$) send traffic to the gateway. We assume that the capacity region $\Lambda(n)$ is unknown, but for the purpose of the analysis, also time-constant ($\Lambda(n) = \Lambda$) and convex. In these settings, the fairness problem can be modeled as the utility maximization of a slotted-time system. A time slot corresponds to the fixed duration of a rate limiting (rate limiter) assignment.

We adapt the notation of Sections 6.1.2 and 6.2. We drop the node index i and extend the notation to consider the time-slotted behavior.

- **Service rate**: $\vec{x}(n) \in \mathbb{R}^F$, where $x^j(n)$ is the amount of traffic received at the gateway from flow j during time slot n.
- **Limiting rate**: $\vec{\rho}(n) \in \mathbb{R}^F$, where $\rho^j(n)$ is the maximum amount of traffic set by the gateway for flow j during time slot n.
- **Capacity region**: $\Lambda(n)$, which is the set of all the achievable $\vec{x}(n)$ by the system
- **Utility function**: We consider a concave, continuously differentiable and strictly increasing utility function $u : \mathbb{R}^F \to \mathbb{R}$. We denote $u(n) = u(\vec{x}(n))$ the utility achieved at slot n that can be computed by the gateway.

The achieved throughput vector $\vec{x}(n)$ obviously depends on the value of the rate limiters when $\vec{\rho}(n) \in \Lambda(n)$. Indeed, we have

$$\vec{x}(n) = \vec{\rho}(n) \tag{6.4}$$

when $\vec{\rho}(n) \in \Lambda(n)$. Otherwise, we can only write that

$$\vec{x}(n) \in \partial\Lambda(n),$$

where $\partial\Lambda(n)$ is the boundary of the capacity region $\Lambda(n)$. This constraint means that the gateway has full control of the service rate achieved by each flow when $\vec{\rho}(n) \in \Lambda(n)$. However, it loses this control, when going out of this region.

6.3.2 Algorithm Description

The optimization problem we focus on is equivalent to answering the following question:*"How should the gateway set the limiting rates $\vec{\rho}(n)$ in order to maximize the utility $u(n)$ without knowing the network capacity $\Lambda(n)$?"*.

To answer this question, we note that the utility $u(\vec{x})$ is a scalar field over the capacity region. A standard tool for performing optimization on $\vec{x}$ in a convex set is a gradient ascent. In practice, $\Lambda(n)$ is time-varying and may not always be convex, but Section 6.5 demonstrates experimentally that our algorithm also performs well in these scenarios. The key difference between our practical problem and the standard optimization problem is that the gateway does *not* directly control the achieved throughput $\vec{x}(n)$. Instead, the only variables it can control are the rate limiters $\vec{\rho}(n)$. The gradient ascent on $\vec{\rho}(n)$ can only be performed when the rates are set within the capacity region (i.i., $\vec{\rho}(n) \in \Lambda(n)$). Outside this region, $\vec{\rho}(n)$ does not determine $\vec{x}(n)$ anymore.

To tackle this problem, we introduce a new algorithm that we call E&E (Explore & Enhance). The E&E algorithm is detailed in Algorithm 1 and it combines two phases.

- The *enhance* phase is applied whenever the allocated rates $\vec{\rho}(n-1)$ at the previous time-step $n-1$ were feasible, i.e. whenever the system measured that $\vec{x}(n-1) = \vec{\rho}(n-1)$, indicating that the previous allocation was within the capacity region $\Lambda(n-1)$. The algorithm tries then to increase the utility by performing a gradient ascent. This is done at line 17 of the algorithm. If it is successful, it updates the rate vector $\vec{\rho}(n)$, and repeats a new enhance phase. If it fails, then it means that the new attempted rate allocation was outside the (unknown) capacity region. The algorithm then backtracks to the previous rate vector, which was within the capacity region, and moves to the explore phase described next.

- The *explore* phase is applied whenever the allocated rates at the previous time-step are not feasible. It first choses two flows at random among the F flows. Then it decreases the rate of the first one by a random amount, and increases the rate of the second one to reach the same value of the utility function. Note this is an easy operation that only requires to solve one equation with one unknown. This allows the algorithm to explore a new part of the capacity region, which provides the same level of utility $u(\cdot)$ but which may be a better point to successfully perform an enhance phase. The explore phases are crucial to avoid being locked in a local maximum, at the boundary of the capacity region.

The E&E algorithm starts from an initial condition that is the allocation obtained from running the underlying MAC (in our case IEEE 802.11). Then, we show in Lemma 6.1 that the algorithm improves the utility function during each successful enhance phase, and leaves it unchanged between two consecutive successful explore phases (during which the last stable assignment $\vec{r}$ remains equal to the measured achievable rates $\vec{x}$ at the end of the last successful phase). A convergence proof appears, however, beyond reach because the capacity region is not known, and even possibly time-varying.

Algorithm 2 E&E (Explore & Enhance) Algorithm

```
 1: **Init:** pick $\alpha > 0$
 2: **At time slot** $n = 1$**:**
 3: denote $\vec{x}(0)$, the service obtained during slot 0, and start the algorithm from
    the allocation $\vec{\rho}(1) = \vec{x}(0)$
 4: store the *last stable assignment* $\vec{r}(1) = \vec{\rho}(1)$
 5:
 6: **At each time slot** $n > 1$**:**
 7: denote $\vec{x}(n-1)$, the service obtained during slot $n-1$
 8: **if** $\vec{x}(n-1) = \vec{\rho}(n-1)$ **then**
 9:     goto **Enhance phase**
10: **else**
11:     goto **Explore phase**
12: **end if**
13: broadcast one packet that contains $\vec{\rho}(n)$
14:
15: **Enhance phase:**
16: set $\vec{r}(n) = \vec{\rho}(n-1)$
17: set $\vec{\rho}(n) = \vec{x}(n-1) + \alpha \cdot \frac{\overrightarrow{\bigtriangledown u(\vec{x}(n-1))}}{\|\overrightarrow{\bigtriangledown u(\vec{x}(n-1))}\|}$
18:
19: **Explore phase:**
20: pick randomly $i, j \in \{1, \ldots, F\}$, with $i \neq j$
21: pick randomly $\beta \in [0, \alpha]$
22: set $\rho^i(n) = r^i(n-1) - \beta$
23: find $\rho^j(n)$ that satisfies
        $$u(r^1(n-1), \ldots, \rho^i(n), \ldots, \rho^j(n), \ldots, r^F(n-1)) = u(\vec{r})$$
24: set
        $$\vec{\rho}(n) = [r^1(n-1), \ldots, \rho^i(n), \ldots, \rho^j(n), \ldots, r^F(n-1)]$$
```

We stress that Lemma 6.1 proves that the time-evolution of the utility achieved by the last stable assignment $\vec{r}(n)$ is non-decreasing, but this does not imply that the evolution of the network utility $u(\vec{x}(n))$ is necessarily non-decreasing. In fact, the network utility $u(\vec{x}(n))$ might decrease if an allocation outside the network capacity is chosen ($\vec{\rho}(n) \notin \Lambda(n)$). However, the non-decreasing property of $\vec{r}(n)$ is still interesting to have, because the algorithm can slightly be modified to take advantage of this. Indeed, an additional *exploitation phase* can be added during which the mechanisms exploits (i.e., uses) at time n the best achievable allocation it discovered at this point in time (i.e., it sets $\vec{\rho}(n) = \vec{r}(n)$). This exploitation phases can then be scheduled to happen periodically and from Lemma 6.1, we know that $u(\vec{x}(n))$ is guaranteed to be non-decreasing during these exploitation phases (because we have $\vec{x}(n) = \vec{\rho}(n) = \vec{r}(n)$ for a time-constant capacity Λ). As the addition of an *exploitation phase* does not modify the main methodology of the algorithm, we do not consider the addition of this third phase in the remainder of the chapter.

Lemma 6.1 *The utility function evaluated in the successive last stable assignment rates, $u(\vec{r}(n))$, is non-decreasing with n.*

Proof: We note that $\vec{r}(n)$ is the last stable assignment, and therefore $\vec{r}(n) = \vec{\rho}(n-k) = \vec{x}(n-k)$ for some $k \geq 0$. At the end of a successful enhance phase, one can take $k = 0$ in the previous relation, hence

$$\vec{r}(n+1) = \vec{r}(n) + \alpha \cdot \frac{\overrightarrow{\bigtriangledown u(\vec{r}(n))}}{\|\overrightarrow{\bigtriangledown u(\vec{r}(n))}\|},$$

and $u(\vec{r}(n+1)) \geq u(\vec{r}(n))$ because $u(\cdot)$ is concave (see Theorem 21.4 in [CZ08]).

At the end of an unsuccessful enhance phase the last stable assignment remains unchanged. In addition, its utility remains unchanged at the end of an explore phase. As a result, in these two cases we have that $u(\vec{r}(n+1)) = u(\vec{r}(n))$.
$\square$

We highlight that the E&E algorithm works for utility functions that are concave, continuously differentiable and strictly increasing. The standard max-min fair utility function does not satisfy these conditions, as it is not continuously differentiable. Nevertheless, a well-known solution exists to solve this optimization problem (i.e., water-filling). Therefore, the E&E algorithm can easily be extended to achieve max-min fairness by (i) starting from a feasible allocation where all flows achieve the same throughput and (ii) following the water-filling policy at each enhance phase (as the gradient may not be defined).

6.4 Joint Congestion Control and Fairness for WMNs

The complete framework that we propose results from the interaction between the intra- and inter-flow mechanisms. Indeed, we solve both the congestion-control and fairness problem by adapting the per-flow link throughput at each node. For flow j at node i, the parameter ρ_i^j is set by the inter-flow mechanism if the node is a one-hop neighbor of the gateway and by the intra-flow mechanism otherwise.

In the first case, all one-hop neighbors receive the value to set ρ^j by the gateway. To ensure fairness, the gateway runs the E&E algorithm to continuously update the rate allocation. The gateway sends a single broadcast message containing rate limiter parameter settings to its one-hop neighbors at regular time intervals (typically every three seconds).

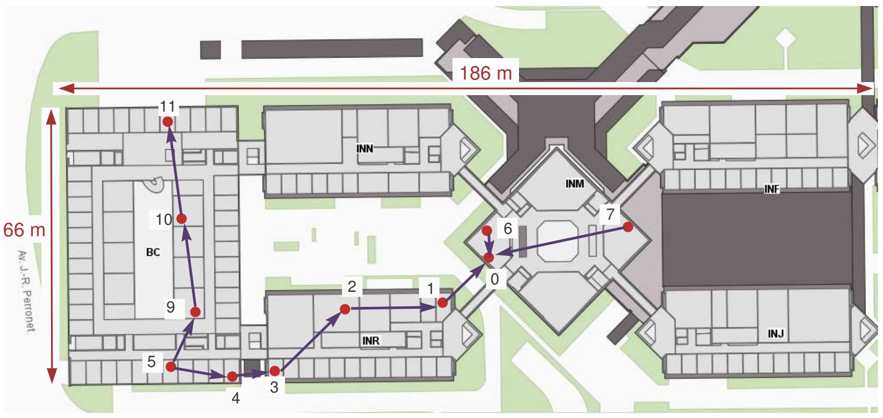


Figure 6.7: The 12-node testbed.

In the second case, the parameter ρ_i^j of rate limiter for flow j is controlled locally at node i by the intra-flow mechanism (without any message passing). Hence, the intra-flow congestion control mechanism *propagates* the fair allocation obtained at the one-hop neighbors of the gateway with the inter-flow mechanism deeper into the network.

In the next section, we demonstrate the effectiveness of this approach.

6.5 Experimental Evaluation

In this section, we extensively evaluate our solution on a real wireless mesh networks. We begin by studying the performances of both the intra- and inter-flow schemes in isolation, and then we evaluate their interaction in order to solve the initial starvation problem introduced in Section 6.1.

6.5.1 Hardware and Software Description

We use 12 IEEE 802.11 nodes of our testbed as depicted in Figure 6.7. Each node is an off-the-shelf Asus WL-500gP wireless router equipped with a single omni-directional antenna. Each router runs the version *8.09.2* of the openWRT firmware [OPE] with the *Click* modular router [KMC$^+$00] used in user mode. We implemented our mechanisms in C as five new *Click* elements that use the MultiFlowDispatcher [SL09] functionalities in order to create a new queue at run time only when the corresponding flow appears at a node. Additionally, we set the size of the MAC interface buffer to 10 packets and the size of the per-flow buffers to 100 packets. We set the parameters of the intra-flow mechanism accordingly to maintain a small amount of packets in the per-flow queues. We use $T_1 = 20$ (a little larger than the MAC queue size to maintain some packets in the per-flow queue), $T_2 = 40$ and $T_3 = 80$ (close to the buffer size limit to avoid overflows).

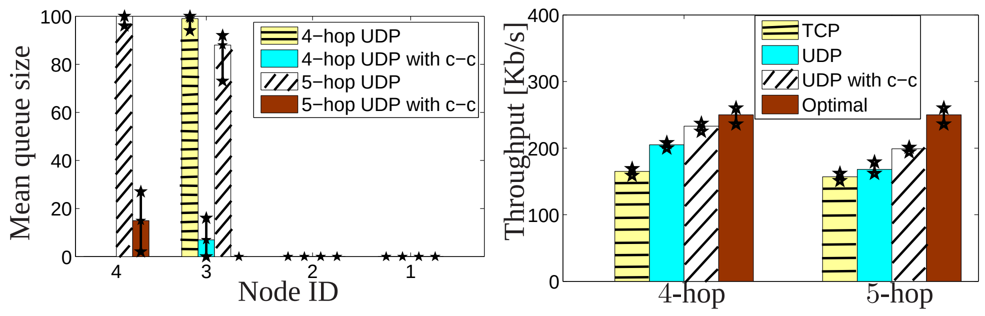


Figure 6.8: Experimental results for a 4-hop (F_4) and a 5-hop flow (F_5). We show the effect of our intra-flow congestion control described in Section 6.2 (with c-c) on the queue size (left) and the median end-to-end throughput (right) with the 25 and 75 percentile confidence intervals.

Finally, we make two practical extensions to the E&E algorithm: (i) we handle the time variability of the capacity region by testing the last stable assignment $\vec{r}$ after 10 unsuccessful explore phases in order to check whether it remains sustainable; and (ii) we limit the possible loss of the control messages sent by the gateway by using a pseudo-broadcast packet (addressed to the neighbor with the weakest link and overheard by the other nodes), instead of a pure broadcast one.

6.5.2 Evaluation of the Intra-Flow Congestion Control

We evaluate the efficiency of our intra-flow congestion control mechanism by considering one 4-hop flow (F_4) and one 5-hop flow (F_5). These scenarios are relevant because IEEE 802.11 is known to perform poorly and introduces much congestion in such configurations, as we showed in Chapter 3. F_4 and F_5 consist in the following paths through the network:

- F_4: $4 \to 3 \to 2 \to 1 \to 0$
- F_5: $5 \to 4 \to 3 \to 2 \to 1 \to 0$.

To assess the level of congestion, we look at the queue occupancies at the relay nodes (the source is fully-backlogged and its queue is full at all time). Additionally, we look at the end-to-end throughputs. We run each experiment for 90 minutes and each minute we measure the queue size and average throughput. Figure 6.8 shows the median value with its 25 and 75 percentile. We display the results for each of these two flows (F_4 and (F_5) both for the case when UDP is used alone (i.e., without a congestion control scheme) and for the case with our network layer congestion-control scheme.

We observe that, when UDP is used alone, the queue size of the first relay, for flow F_4, and the queue sizes of the first and second relays, for flow F_5, are very close to their size limit and often overflow. This creates packet losses, loss of efficiency, and high end-to-end delays.

In contrast, the intra-flow mechanism efficiently performs congestion control by keeping a small queue at all the relay nodes. This translates into a significant reduction of end-to-end delays, because both the queuing delay and the traveling delay (i.e., higher end-to-end throughput) are reduced.

Other solutions exist to achieve congestion-control, the most obvious one being TCP. Although TCP may behave relatively well when only one flow is present, the end-to-end throughput is reduced due to the explicit ACK messages. Our mechanism performs congestion-control without suffering from this drawback. It improves the end-to-end throughput and maintains small queue sizes. We stress that when TCP is used, the performance is exactly the same whether or not the intra-flow congestion control mechanism is used. Indeed, as TCP ensures that the relay queues are small, the rate set by the rate limiters ρ_i^j are such that all packets go through (i.e., the rate of the flow is limited by TCP and not the intra-flow mechanism).

In an attempt to assess how close to optimal the obtained throughputs are, we compute an *optimal* value as follows: we divide by 3 the capacity of the bottleneck link when transmitting in isolation (assuming a 2-hop interference model). This represents the best throughput that one could expect in such a scenario (it does not take into account the losses due to collisions).

We stress that our scheme is not intended to be a replacement to TCP, as we do not focus on reliable delivery. Nevertheless, our results give insight into decoupling the goals of congestion-control and reliable delivery. Indeed our intra-flow mechanism ensures that packet losses are, mostly, not due to buffer overflow (congestion) at a relay node anymore.

6.5.3 Evaluation of the Inter-Flow Fairness

We evaluate separately our inter-flow fairness mechanism by considering three 1-hop flows. We fix each flow at a different rate and measure its capacity with and without RTS/CTS enabled.

- F_1' : $1 \to 0$ (capacity: 1.6 Mb/s; 1.5 Mb/s with RTS)
- F_1'' : $6 \to 0$ (capacity: 852 kb/s; 806 kb/s with RTS)
- F_1''': $7 \to 0$ (capacity: 3.2 Mb/s; 2.6 Mb/s with RTS).

Using these three flows, we evaluate all the four possible scenarios:

- Inter 1: F_1' with F_1''
- Inter 2: F_1' with F_1'''
- Inter 3: F_1'' with F_1'''
- Inter 4: F_1' with F_1'' and F_1'''.

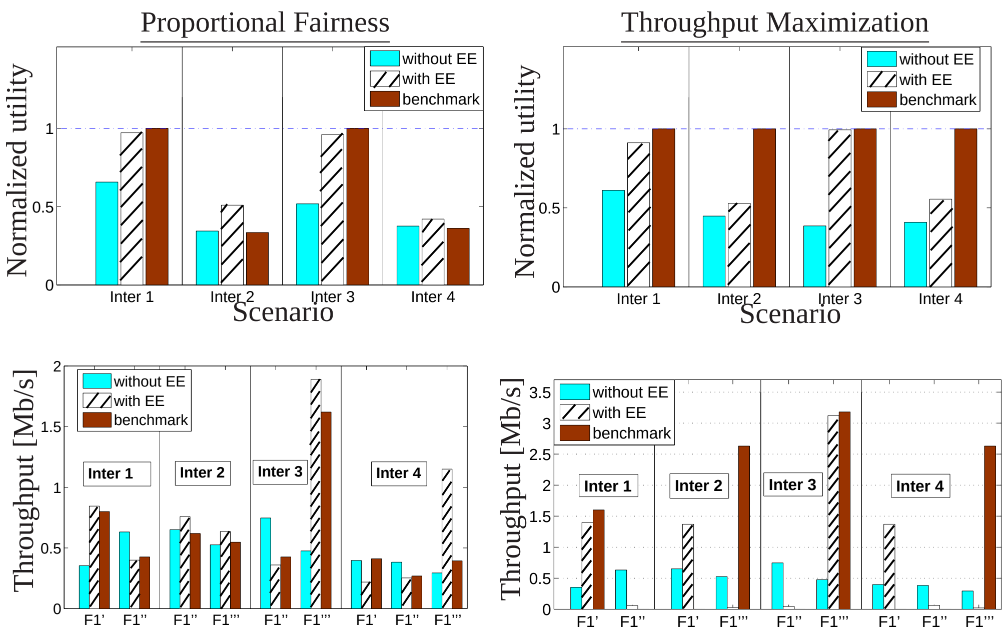


Figure 6.9: Illustration of the normalized utility and throughput achieved with and without our E&E mechanism for the four inter-flow scenarios. We show its performance for both utilities: proportional fairness (left) and throughput maximization (right).

As the sources from F_1'' and F_1''' are hidden from each other, we turn on the RTS/CTS mechanism in the scenarios Inter 2 and Inter 4, otherwise we turn it off. We test the efficiency of our algorithm for both the proportional fairness (Eq. 6.2) and max-throughput (Eq. 6.1) utility functions. We present our results in Figure 6.9 by showing both the achieved utility (normalized with respect to the theoretical upper-bound) and the per-flow throughput obtained for each scenario. We obtain the theoretical upper-bound by assuming an idealized MAC that prevents any collision from happening between flows. In such a case, the proportionally fair optimal allocation for each flow j is C_j/F, where C_j is the flow capacity when transmitting in isolation and F is the number of flows transmitting concurrently. Nevertheless, this upper-bound allocation is not necessarily achievable by IEEE 802.11 due to collisions (e.g., in the case of hidden nodes). Therefore, in order to allow for a fair comparison of our algorithm, we measure a *benchmark* allocation that is obtained by setting the source traffic rates at the rate given by the upper-bound allocation. We stress that our benchmark point is not necessarily the optimal point in the case of collisions (as the sources need to send at a lower rate due to collisions). However as the exact capacity region is unknown, the optimal point is also unknown. Thus our benchmark is a good comparison point in addition to the performances achieved by IEEE 802.11 without the E&E algorithm. Our results show that the inter-flow algorithm improves the performances in all scenarios (compared to both

IEEE 802.11 and the benchmark). Furthermore in the case of scenarios without collision, our algorithm reaches the theoretical optimal fair allocation.

For the case of throughput maximization, the optimal point is trivially obtained by giving all the throughput to the flow with the better link quality and by starving the other flows. Nevertheless, the policy of E&E is to never completely starve a flow so that the gateway can maintain a global view of all the flows existing in the network (i.e., it is impossible to detect whether a flow exists or not if it is completely starved by a rate limiter). Despite this difference, in the case without collision (Inter 1 and Inter 3), the E&E algorithm reaches a point that is very close to the optimal allocation, even though it does not starve the flow with the weakest link capacity.

In the case of hidden nodes (the scenarios Inter 2 and Inter 4), the E&E algorithm improves the performances of IEEE 802.11, but it reaches a local maximum and not the global maximum. This is due to the non-convexity of the capacity region in the case of collisions. A solution to this challenge is to allow the explore phase of the algorithm to try larger steps from time to time in order to discover the existence of a disjoint sub-set of rate allocations that achieve the same utility (and from which an enhance phase would be possible). We are currently studying different variations of the $E\&E$ algorithm to overcome these limitations.

6.5.4 Evaluation of the Complete Framework

We conclude our experimental evaluation by an evaluation of the complete solution that combines both the intra- and inter-flow mechanisms. We consider proportional fairness for this experiment. Toward this goal, we revisit the initial starvation problem between a 3-hop (F_3) and a 1-hop flow (F_1). To evaluate this scenario, we use exactly the same flows as in the experiments of Figure 6.2. In our testbed, these flows correspond to

- F_1: $7 \rightarrow 2$
- F_3: $14 \rightarrow 11 \rightarrow 7 \rightarrow 2$.

For benchmark purposes, we follow a similar methodology to the previous subsection and we derive the optimal allocation by allocating to each flow half of the throughput that it achieves when transmitting in isolation (note that there is no spatial re-use in this scenario). The experimental results are shown in Figure 6.10. We see that IEEE 802.11 starves the 3-hop flow and it achieves a utility that is less than 30% of the optimal allocation. On the contrary, our solution benefits from both our intra- and inter-flow mechanisms and it results in an improvement of the utility by a factor of 3.

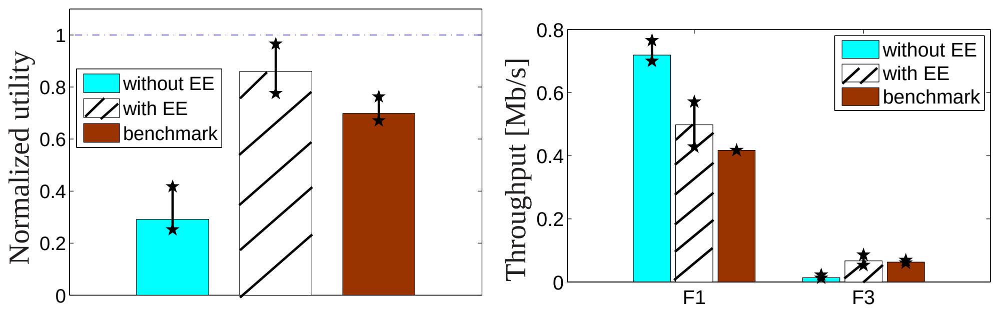


Figure 6.10: Illustration of the normalized utility and throughput achieved by our scheme for the introductory starvation problem of Figure 6.2 between a 3-hop and a 1-hop UDP flow. The E&E algorithm together with the intra-flow mechanism improve by a factor three the utility achieved by IEEE 802.11.

6.6 Concluding Remarks

We proposed and implemented a fully distributed scheme that allows us to control congestion in a wireless multi-hop mesh network with a tree topology, by decoupling the fair rate allocation scheme between flows and the congestion control within the different hops of each single flow.

The fair rate allocation is performed by the gateway, using an "Explore and Enhance" stochastic optimization that alternates gradient ascent on a given utility function (Enhance phase) with avoidance of local maxima (Exploration phase).

The intra-flow congestion control adapts the transmission rates of each node along a flow by estimating the queue sizes of the downstream nodes. It does not require message passing, as this estimation is made possible by the broadcast nature of the wireless channel. Furthermore, this algorithm operates at the network layer and does not interact with any parameter of the MAC. This makes our algorithm potentially compatible with multi-hop networks using another underlying MAC protocol than IEEE 802.11.

In our complete solution, message passing is limited to exchanges of messages between the gateway and its first hop neighbors: once the gateway has assigned rates on the first hop links, the intra-flow mechanism propagates them to the rest of the network. Some key advantages of the solution described in this chapter are that it does not require to know the capacity of the network (which is indeed difficult to obtain) and that it is transparent to the MAC and upper layers. Finally, the initial condition of the algorithm is given by the default rates obtained by an IEEE 802.11 network, which are progressively modified to increase the utility function. This avoids a long transient phase during which the network would not be operational. The experimental results show indeed a significant improvement in terms of fairness and throughput. Our next step is to analyze theoretically the convergence of the E & E algorithm when the capacity region is time-constant and convex.

Chapter 7

Multi-Hop Networks Beyond Capacity

7.1 Background

7.1.1 Problem Statement

The throughput of wireless multi-hop networks has received much attention in the case where the network operates *within* the capacity region, but not *above* it. In this chapter, we study the answers to this question for IEEE 802.11 networks on the simplest non-trivial topology, which is a linear network. We chose to focus on this topology for the following reasons: (i) it is found as a part of almost every wireless multi-hop network; (ii) its capacity region is easy to compute, but the network dynamics may be complex outside the capacity region, as indicated by the non-monotonic curve of Figure 7.1; (iii) despite the complexity of the dynamics, we can prove the non-monotonicity of the curve of Figure 7.1, first, mathematically on a Markov chain model and, next, experimentally through measurements in a real testbed. We compute the different phases that the throughput evolution goes through at different input rates, and the values at which the transitions occur.

In order to have a better idea of the counter-intuitive relation between the source rate λ and the end-to-end throughput μ, we present simulation results for a 4-hop network in Figure 7.1. In this scenario, only node 0 (i.e., the source) receives fresh packets at a rate λ, whereas node 1, 2 and 3 do not inject any new packets and solely act as relays for the packets to the destination node 4 (see Figure 7.2). We consider that the IEEE 802.11 protocol is used with its standard settings: (i) without RTS/CTS, (ii) with a transmission range of 1-hop, and (ii) with a sensing range of 2-hop. In other terms, this means that node i can successfully transmit only if the nodes $i \pm k$ are silent, with $k \in \{1, 2\}$. The results show that the relation between the input rate λ and the throughput μ can be divided into three distinct phases according to λ:

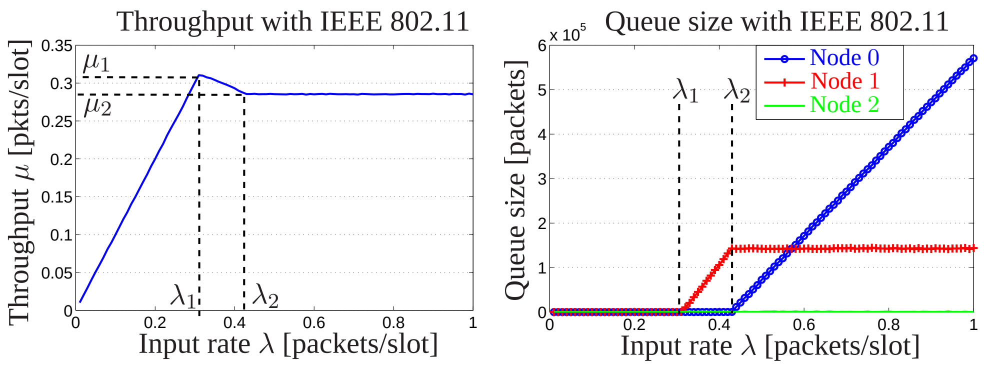


Figure 7.1: Simulation results for a 4-hop network show that after an optimal value λ_1 the end-to-end throughput decreases from μ_1 to μ_2 (when the input rate is λ_2) and remains constant thereafter.

- $\lambda < \lambda_1$: It is the case generally considered, where the source rate is within capacity. We note that in this case μ is an increasing function of λ and it reaches its maximum at μ_1 ($\mu_1 = \lambda_1$).
- $\lambda > \lambda_2$: It corresponds to the saturated regime, where the source always has packets to send. We note that in this situation, we have $\mu = \mu_2$ independently of the value of λ.
- $\lambda_1 < \lambda < \lambda_2$: It is the phase during which a form of congestion collapse occurs. Indeed, we highlight the counter-intuitive results showing that μ is a decreasing function of λ in this region.

Moreover, we note that these results are not an artifact of particular settings we would have adopted. Indeed, a similar phase-transition behavior is noticed in ns-2 simulations for a 7-hop network with RTS/CTS [LBDC$^+$01].

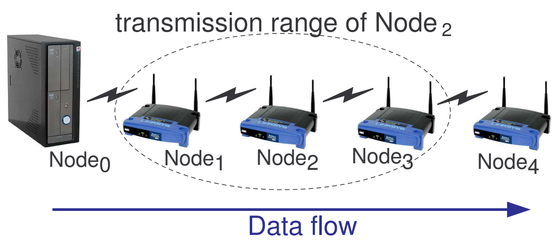


Figure 7.2: Simple 4-hop linear scenario leading to stability problem and throughput degradation with IEEE 802.11.

7.1.2 Related Work

The IEEE 802.11 protocol is known to perform poorly in a practical multi-hop environment, both in the case of TCP and in saturated UDP traffic. Indeed, in [GSK04] Gambiroza et al. show the inefficiency of the protocol in providing optimal performances, as far as delay, throughput and fairness are concerned. In a more recent work [JP09], Jindal et al. show with a simulation that, if the sources are rate-controlled, IEEE 802.11 achieves a max-min allocation that is at least 64% of the max-min allocation obtained by a perfect scheduler. If the sources are not rate-limited, we presented experimental evidence that the performance can be much worse (see Chapter 5 and 6). The performances of linear networks with a saturated source (i.e., node 0 always having packets to send) are studied both through a continuous-time model by Denteneer et al. [DBvdVH08]. By considering a 1-hop interference model, they derive the exact value of the throughput in a particular case and make conjectures for the general case. Our work differs from these papers, because we focus on: (i) understanding how the performance varies from an environment that is rate-controlled (small λ) to one that is not (large λ); and (ii) elucidating the reasons that cause these transitions.

In order to overcome the inefficiency of IEEE 802.11, enormous progress has been made since the seminal work on back-pressure by Tassiulas et al. [TE92]. Back-pressure is based on a centralized scheduler that selects for transmission the links with the greatest queue difference. Such a solution works well for a wired network, but it is not adapted to a multi-hop wireless network where decentralized schedulers are needed, due to the synchronization problem. Toward this goal, Modiano et al. introduce the first distributed scheduling framework that uses control messages to achieve throughput optimal performances [MSZ06]. Further extensions to distributed scheduling strategies are discussed in works such as [CKLS08], where Chapokar et al. propose a scheduler that attains a guaranteed ratio of the maximal throughput. Another effort to reduce the complexity of back-pressure is presented in [YST08], where Ying et al. propose to enhance scalability by reducing the number of queues that need to be maintained at each node. The tradeoff that exists in each scheduling strategy between complexity, utility and delay is discussed in depth in [YPC08] by Yi et al. Despite their multiple advantages, one of the drawbacks of these previous methods is that they require information about the queue from other nodes. The usual solution is to use message passing, which produces costly overhead even if it is limited to the direct neighbors.

More recently, researchers have proposed decentralized and throughput-optimal CSMA schemes that do not require queue information from other nodes. In [GLS07], Gupta et al. propose an algorithm that uses the maximal node degree in the network. Proutière et al. [PYC08] propose another algorithm, where each node makes the scheduling decision based solely on its own queue. Shin et al. [SSR09] propose an algorithm that achieves stability and where each node makes scheduling decisions based on a logarithmic function of its own buffer occupancy. The appealing property of these algorithms is their throughput optimality with perfect

CSMA, but they require very large queue sizes (i.e., in the order of thousands of packets), which can seriously degrade delay performances. A different approach is taken by Jiang et al. who introduce an adaptive CSMA algorithm that adjusts the transmission aggressiveness based on a differential between the arrival and service rate [JWa]. To summarize, significant theoretical progress has been recently made on algorithms that are based on variations of the back-pressure algorithm (i.e., MaxWeight). Nevertheless, we emphasize that MaxWeight is based on two assumptions: (i) the set of nodes and the traffic demands are fixed; and (ii) all the sources are rate limited to transmit only at a rate within the capacity region. In [vdVBS09], van de Ven et al. show that MaxWeight policies might fail to provide stability in the case that the first condition is violated, due to the variability in the system. Our work differs, because we focus rather on the second condition and we are interested in analyzing how the source rate affects the stability of a IEEE 802.11 multi-hop network. Indeed, the capacity is generally time varying and difficult to measure exactly in practice [SSGG09]. Hence, it is important to know what type of performance losses can be expected when the sources receives packets at a rate above the physical capacity of the network (which differs from the link capacity).

7.1.3 Network Model

We introduce a model that is based on the common assumption of a slotted discrete time axis [CKLS08, ES05, LE99, TE92, YST08, AST09], that is, each transmission takes one time slot and all the transmissions, occurring during a given slot, start and finish at the same time. Moreover, we consider that node 0 is the only source and that the packets arrive at the source following a random distribution of mean λ. Moreover, the number of arriving packets $\xi_0(n)$ is i.i.d (independent and identically distributed) in different time slots $n \in \mathbb{N}$. The destination of all packets is Node N, and nodes 1 to $N-1$ act as relays ($\xi_{i>0}(n) = 0$ for all n). Every node has an infinite buffer for storing messages, and uses FIFO scheduling policy, with transmissions from node i to its nearest right neighbor, node $i+1$. A packet leaves the system once it reaches the destination node N. We capture the restriction linked to the wireless medium by assuming a 2-hop sensing range (which eliminates collisions). A node i can therefore only transmit in a slot if the nodes $i \pm k$ are not scheduled for all $k \in \{1, 2\}$.

Under these assumptions, we describe the state of the system by the variables $\{b_i(n), n \in \mathbb{N}\}$, with $0 \leq i \leq N-1$, where $b_i(n)$ represents the number of packets at time instant n in the queue of node i. At the beginning of each slot the transmission pattern

$$\vec{z}(n) = [z_0(n)\ z_1(n)\ z_2(n)\ \ldots\ z_{N-1}(n)]$$

is selected, where $z_i(n) = 1$ if node i is scheduled for transmission at slot n and $z_i(n) = 0$ otherwise. In order to pick $\vec{z}(n)$, each node i with a non-empty queue competes for the access to the channel and picks a uniform backoff $\beta_i(n) \in [0; 1]$, and the nodes with an empty queue set $\beta_i(n) = \infty$. Next, if there is at least one node with a non-empty queue, the node i with the smallest backoff

$$\beta_i(n) = \min_{0 \le j < N} \beta_j(n)$$

is scheduled for transmission ($z_i(n) = 1$) and the nodes within its 2-hop sensing range are removed from the competition ($\beta_{i+k}(n) = \infty$ for $k \in \{0, 1, 2\}$). Then, the node with the next smallest backoff is selected. This process repeats itself until no more nodes compete for the channel (i.e. $\beta_i(n) = \infty$ for all i) and the final transmission pattern $\vec{z}(n)$ is obtained. We stress that this model is simpler than the real IEEE 802.11 protocol as we do not take into account the exponential increase of the contention window cw_i (i.e., the doubling of the contention window after an unsuccessful transmission). Nevertheless, we note that if IEEE 802.11 is used with a fixed contention window (i.e., $CW_{min} = CW_{max} = cw_i$) the protocol selects its backoff uniformly in $[0; cw_i - 1]$, which is equivalent to the selection of $\beta_i(n) \in [0; 1]$ in our model. We will validate that the results predicted by this model with experiments with the real IEEE 802.11 protocol in Section 7.4. Finally, the dynamics of our system is captured by the relation

$$b_i(n+1) = b_i(n) + z_{i-1}(n) - z_i(n) + \xi_i(n).$$

We note that the relation $b_i(n) - z_i(n)$ is always non-negative, because only the nodes with a non-empty queue (i.e., $b_i(n) > 0$) can be scheduled for transmission (i.e., have $z_i(n) = 1$).

7.2 Simulations

Before providing a quantitative analysis in the next section, we will first gain a qualitative understanding of the phase transition in throughput at λ_1 and λ_2 in Figure 7.1, by using time-slotted simulations of the model of Section 7.1.3. For all our scenarios, simulations are repeated to sweep all $\lambda \in [0, 1]$ with increments of 0.01. We stress that we assume, without loss of generality, that all the links have a capacity of 1 and thus we do not consider the cases $\lambda > 1$. The theoretical capacity of the network $\lambda = 1/3$ is reached by a perfect centralized scheduler.

7.2.1 4-Hop Networks

Simulating the 4-hop topology of Figure 7.2 for 10^6 slots, we find that the two transition points $\lambda_1 = 0.32$ and $\lambda_2 = 0.43$ correspond to two specific behavioral changes in the queue evolution of the nodes.

Indeed, a perfect centralized scheduler reaches the capacity $\lambda = 1/3$ by always scheduling nodes 0 and 3 concurrently. With IEEE 802.11, there is a non-zero probability of having these two nodes scheduled independently (e.g. if exactly one of these two nodes has an empty queue), which yields that $\lambda_1 < 1/3$. We see that the transition point λ_1 is the source rate where the queue of node 1 starts to build up in the network. Interestingly, it is at the first relay node 1 and not at the source node 0 where this happens. Because the congestion takes place after the first hop, any increase in the source rate beyond λ_1 (up to λ_2) maps to a proportional decrease in the end-to-end throughput μ. Indeed, as the queue of node 0 does not build up for $\lambda \in [\lambda_1, \lambda_2]$, we have that an increase in λ maps to a proportional increase in the number of packets transmitted by node 0. Moreover, the shared nature of the wireless medium implies that each time node 0 transmits a packet that will be queued up at node 1 (i.e., $\lambda - \lambda_1$ packets per time slot in average), it prevents both node 1 and node 2 to forward the packets waiting in their own queues. Resources are thus wasted on the queue build-up of node 1 and the end-to-end throughput decreases linearly with λ.

The point λ_2 corresponds to the threshold rate above which the queue of node 0 eventually begins to build up. After this threshold, the input rate λ has no resulting effect on either the throughput μ or the queue growth at node 1, because the (saturated) source always has packets to send anyway and thus an increase on λ does not make any difference in the network, except that the queue of the source grows faster.

7.2.2 5-Hop Networks

Having shown the relation between the phase transition in the end-to-end throughput μ and buffer build-ups in a 4-hop network, we confirm that the relation remains for larger K-hop topologies.

Toward this goal, we begin by simulating a 5-hop network in the same setting. Our results, depicted in Figure 7.3, show that there are in this case, not two, but three transition points at λ_1, λ_2 and λ_3. Again, each one of these points λ_i corresponds to a threshold rate above which a new queue begins to build up in the network.

We note that, in a 5-hop network, the first transition point $\lambda_1 = 0.3$ occurs again before the theoretical $1/3$ capacity of the network and this is due to the queue build-up at node 2. To explain why node 2 is the first to have its queue explode, we highlight that it is the only node that cannot transmit concurrently with another (due to the 2-hop sensing range). Hence, it has a smaller channel access probability than the other competing nodes and it is the first one to build-up when the source rate λ increases.

The second transition point $\lambda_2 = 0.35$ is the source rate where node 1 starts to build up. We observe that this build-up has two effects: (i) it reduces the slope of the throughput decrease relatively to the input rate λ; and (ii) it linearly reduces the amount of buffer build-up at node 2. The explanation for the second effect is that the build-up at node 1 implies additional transmissions from node 0, which proportionally reduce the number of transmissions from node 1, and as result reduce the backlog at node 2. This also explains the first effect, because the packets queued at node 2 consume more resources (2 transmission slots) than the packets

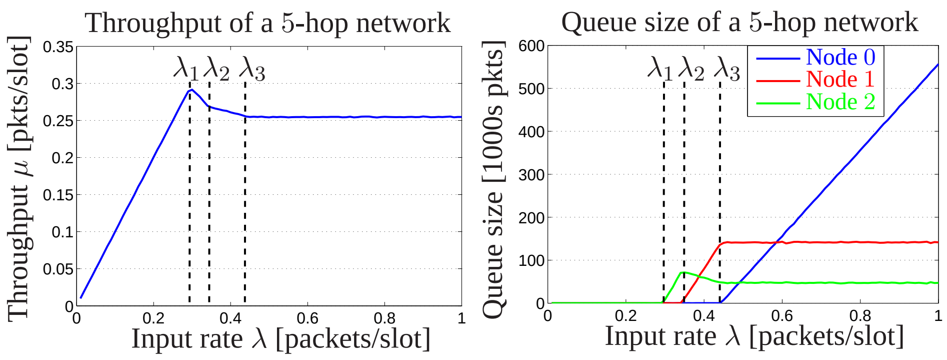


Figure 7.3: Simulation results for a 5-hop network. The transition points in throughput λ_i correspond to the threshold rate above which a new queue starts to build up in the network. The queue of nodes $k > 2$ are not displayed because they never build up (i.e., they follow a positive recurrent evolution).

queued at node 1 (1 slot). Hence, as in the region $[\lambda_2, \lambda_3]$ the additional packets are queued at node 1 instead of node 2, they consume less transmission resources and this reduces the slope of the throughput decrease relatively to λ.

Finally the last transition point $\lambda_3 = 0.45$ happens after the build-up of node 0 and the effect is similar to the one of λ_2 in the 4-hop case.

7.2.3 Larger K-Hop Networks

We end our simulation study by extending our investigations to larger K-hop topologies. We repeated our simulations for networks of up to 30 hops and we find that (i) the only nodes where the queues always build-up when the input rate is above a threshold λ_i are node 2 (at λ_0), node 1 (at λ_1) and node 0 (at λ_2); (ii) for networks larger than 7 hops, the queue of node 3 builds up when the input rate is in the range $0.28 \leq \lambda \leq 0.30$, whereas the queue of node 4 builds up only for the rate $\lambda = 0.28$; and (iii) the queue of node $i > 4$ always remains bounded for any input rate. Table 7.1 shows the values of the transition points λ_i that we obtained

Network size	λ_1	μ_1	λ_2	μ_2	λ_3	μ_3
4-hop	0.32	0.31	0.43	0.29	-	-
5-hop	0.30	0.29	0.35	0.27	0.45	0.26
6-hop	0.29	0.29	0.35	0.26	0.46	0.25
7-hop	0.29	0.28	0.35	0.26	0.46	0.25
10-hop	0.29	0.27	0.35	0.26	0.46	0.25
15-hop	0.29	0.27	0.35	0.26	0.46	0.25
30-hop	0.29	0.27	0.35	0.26	0.46	0.25

Table 7.1: Simulation results for the transition points $\{\lambda_i, \mu_i\}$ for different network sizes.

for different network sizes.

7.3 Mathematical Analysis

In Chapter 3, we showed that the 4-hop model is the simplest and smallest topology to suffer from congestion and throughput reduction in the saturated regime. Hence, we focus our mathematical study on this scenario.

7.3.1 4-Hop Network

A 4-hop network is completely defined by the state

$$\{b_0(n), b_1(n), b_2(n), b_3(n)\}$$

and, if $\{b_0(n), b_1(n), b_2(n), b_3(n)\} = \{b_0, b_1, b_2, b_3\}$, then its transmission probabilities can be expressed as

$$\begin{array}{rcl}
\mathbb{P}(z_0(n)=1) & = & \mathbb{1}_{b_0>0}(\mathbb{1}_{b_0>0}+\mathbb{1}_{b_3>0})/\sum_i \mathbb{1}_{b_i>0} \\
\mathbb{P}(z_1(n)=1) & = & \mathbb{1}_{b_1>0}/\sum_i \mathbb{1}_{b_i>0} \\
\mathbb{P}(z_2(n)=1) & = & \mathbb{1}_{b_2>0}/\sum_i \mathbb{1}_{b_i>0} \\
\mathbb{P}(z_3(n)=1) & = & \mathbb{1}_{b_3>0}(\mathbb{1}_{b_0>0}+\mathbb{1}_{b_3>0})/\sum_i \mathbb{1}_{b_i>0},
\end{array} \tag{7.1}$$

where $\mathbb{1}_A$ is the usual indicator of event A, which takes value 1 if event A occurs and 0 otherwise. Based on this model, we derive a series of lemmas and theorems.

Lemma 7.1 *If $\lambda < 1/4$, then the Markov Chain $\{\vec{b}(n)\}$ is positive recurrent.*

Proof: We will use the well-known Foster-Lyapunov technique for proving positive recurrence of the Markov Chain. Let us define the function

$$L(b_0, b_1, b_2, b_3) = 4b_0 + 3b_1 + 2b_2 + b_3$$

and prove that it is an appropriate Lyapunov function. Note that the function represents the workload of the system (i.e., the total number of time slots for all packets in the system to leave). Note also that every successful transmission by any node in the system reduces the function by exactly 1. Finally, note that if at a given time slot there is at least one packet in the system, then at this time slot there will be at least one successful transmission. Define also the set $V = \{(b_0, b_1, b_2, b_3) : b_0 + b_1 + b_2 + b_3 \le 1\}$. Then we have

$$\mathbb{E}\left(L(\vec{b}(n+1)) - L(\vec{b}(n))|\vec{b}(n) = \vec{b}\right) \le 4\lambda$$

for all $\vec{b}$ and

$$\mathbb{E}\left(L(\vec{b}(n+1)) - L(\vec{b}(n))|\vec{b}(n) = \vec{b}\right) \le 4\lambda - 1 < 0$$

for all $\vec{b} \notin V$, therefore the conditions for positive recurrence are satisfied. □

Lemma 7.2 *If $\lambda \geq 1/3$, then the Markov Chain $\{\vec{b}(n)\}$ is not positive recurrent.*

Proof: We again use Foster-Lyapunov techniques. Take the function

$$L(b_0, b_1, b_2, b_3) = 3b_0 + 2b_1 + b_2,$$

which represents the workload of nodes 0, 1 and 2. Note that the maximal number of transmissions accomplished by these nodes in one time slot is equal to 1, hence,

$$\mathbb{E}\left(L(\vec{b}(n+1)) - L(\vec{b}(n)|\vec{b}(n) = \vec{b}\right) \geq 3\lambda - 1 \geq 0$$

for any vector $\vec{b}$, and the lemma is proved. □

Lemma 7.1 and 7.2 suggest that there exists $1/4 \leq \lambda_1 \leq 1/3$ such that the system is stable for all $\lambda < \lambda_1$ and unstable for all $\lambda > \lambda_1$. We will show that in fact $\lambda_1 < 1/3$, which implies that the maximal end-to-end throughput is not achievable by an IEEE 802.11 network.

Theorem 7.1 *The value of λ_1 defined above is strictly smaller than $1/3$.*

Proof: We showed already that at $\lambda \geq 1/3$ the Markov chain representing the state of the system is not positive recurrent, which, in particular, means that the total number of packets tends to infinity. However, for our analysis we need more detailed information, specifically, we need to know at which node the queue build-up occurs.

For this, consider first node 0. From the equations about the probability of node 0 transmitting (i.e., $\mathbb{P}(z_0(n) = 1)$) we conclude that whenever node 0 is non-empty, its transmission probability is larger than or equal to $1/3$ and whenever node 3 is also non-empty, this probability becomes strictly larger than $1/3$. These two facts imply that the number of packets at node 0 does not tend to infinity with probability 1, or $\lim_{n\to\infty} \mathsf{P}(b_0(n) = 0) > 0$.

Similar arguments allow us to show that at $\lambda = 1/3$ the queues of nodes 1 and 2 grow infinitely. Indeed, the total number of packets in these queues grows infinitely due to Lemma 7.2 and the previously-established fact that the queue of node 0 stays bounded. Note from (7.1) that the probabilities of transmission of nodes 1 and 2 are equal in the case both these queues are non-empty and regardless of the states of queues 0 and 3. Indeed, both probabilities are equal to

$$\frac{1}{2 + \mathbb{1}_{b_0>0} + \mathbb{1}_{b_3>0}},$$

and hence, the expected change in the queue size of node 2 in this case is 0. If we assume now that $\lim_{n\to\infty} \mathsf{P}(b_1(n) = 0) > 0$, then whenever the queue of node 1 empties, the expected change in the queue size of node 2 is negative, which yields that the queue of node 2 stays bounded. This contradicts however Lemma 7.2 and hence, the queue of node 1 grows infinitely. Now we make use again of the fact that whenever both queues 1 and 2 are non-empty, their transmission probabilities are the same, hence, in all these states the expected change in the size of queue 2 is equal to zero, and hence the queue builds up.

We have now established that at $\lambda = 1/3$ the queue of node 0 stays bounded, whereas the queues of nodes 1 and 2 grow infinitely. Assume that $\lambda_1 = 1/3$. This would imply that the throughput of the system grows linearly with λ up to the point $\lambda_1 = 1/3$ and hence it is equal to $\mu = 1/3$ when $\lambda = 1/3$.

Consider node 3 at $\lambda = 1/3$. As both nodes 1 and 2 are non-empty, we have from (7.1) that the probability of node 3 transmitting in any time slot is equal to

$$p_3 = \frac{1 + \mathbb{1}_{b_0>0}}{3 + \mathbb{1}_{b_0>0}},$$

and the probability of node 2 transmitting in any time slot is equal to

$$p_2 = \frac{1}{3 + \mathbb{1}_{b_0>0}}.$$

As $\lambda > 0$, the probability for queue of node 0 to be non-empty is strictly positive, implying that the queue of node 3 stays bounded because $p_3 > p_2$. Note that this analysis is valid for any $\lambda \geq 1/3$, hence the queue of node 3 is always bounded. As the queues of nodes 1 and 2 become infinite at $\lambda = 1/3$, the state of the system may be described by the Markov Chain $\{b_0(n), b_3(n)\}$ which has steady-state probabilities

$$p(i, j) = \lim_{n\to\infty} \mathsf{P}(b_0(n) = i, b_3(n) = j).$$

The throughput of the system (or, equivalently, of node 3) can then be written as

$$\begin{aligned} \mu &= 1/3 \sum_{j=1}^{\infty} p(0, j) + 1/2 \sum_{i=1}^{\infty}\sum_{j=1}^{\infty} p(i, j) \\ &= 1/3P(0, \mathbb{1}) + 1/2P(\mathbb{1}, \mathbb{1}) \end{aligned} \tag{7.2}$$

with obvious notations $P(0, \mathbb{1})$ and $P(\mathbb{1}, \mathbb{1})$. The throughput of node 2 is, however, also equal to μ and to

$$\mu = 1/2P(0, 0) + 1/3P(0, \mathbb{1}) + 1/3P(\mathbb{1}, 0) + 1/4P(\mathbb{1}, \mathbb{1})$$

where $P(0, 0) = p(0, 0)$ and $P(\mathbb{1}, 0) = \sum_{i=1}^{\infty} p(i, 0)$. The two last equations imply that

$$P(\mathbb{1}, \mathbb{1}) = 4/3P(\mathbb{1}, 0) + 2P(0, 0).$$

The throughput of node 0 is equal to $1/3$ and to

$$1/3 = 1/3P(\mathbb{1}, 0) + 1/2P(\mathbb{1}, \mathbb{1}) = P(\mathbb{1}, 0) + P(0, 0),$$

due to the latter equation. Recall now that $P(0, 0)+P(0, \mathbb{1})+P(\mathbb{1}, 0)+P(\mathbb{1}, \mathbb{1}) = 1$ and conclude that $P(0, \mathbb{1}) + P(\mathbb{1}, \mathbb{1}) = 2/3$. From (7.2) it finally follows that

$$\mu = 1/3P(0, \mathbb{1}) + 1/2P(\mathbb{1}, \mathbb{1}) < 1/2(P(0, \mathbb{1}) + P(\mathbb{1}, \mathbb{1})) = 1/3,$$

and the contradiction proves that $\lambda_1 < 1/3$. □

Theorem 7.2 *If node 0 is assumed to be saturated (i.e. to have a packet to transmit at every time instant), then the throughput of the system is equal to $2/7$.*

Proof: We will use an idea from [SdV10]. From the proof of Theorem 7.1 it follows that nodes 1 and 2 are also non-empty with probability 1, and as the queue of node 3 stays bounded, we can introduce time instants τ_0, $\tau_0 + \tau_1, ..$, where

$$\tau_0 = \inf\{t > 0 : b_3(t) = 0\}$$

and

$$\begin{aligned}\tau_{i+1} \;\; = \;\; & \inf\{t > 0 : b_3(\tau_i + t) = 0 \text{ and there exists } \quad 0 < s < t \quad \text{such that} \\ & b_3(\tau_i + s) > 0\}\end{aligned}$$

for $i = 0, 1, ...$ In words, we mark the time instants when node 3 becomes empty. Due to the stability of node 3, the sequence $\tau_0, \tau_1, ..$ consists of a.s. finite integers that are independent. Moreover, the random variables $\tau_1, \tau_2, ..$ are identically distributed, and the behavior of the system in each cycle is statistically the same as in any other cycle.

Thus, we can conclude that the throughput of the system is equal to

$$\mu = \mathbb{E}T/\mathbb{E}\tau_1,$$

where T is the (random) number of transmissions accomplished by node 3 in a typical cycle. Let us now find the expected values of T and τ. We can write

$$\tau_1 = \nu_0 + \nu_1,$$

where ν_0 is the time until node 2 transmits a packet to node 3, and ν_1 is the time until node 3 becomes empty again. First note that ν_0 has a Geometric distribution with parameter $1/3$. Indeed, when node 3 is empty, the other 3 nodes compete for the channel and have equal probabilities ($1/3$ each) to get the access. Hence, it will take a $Geom(1/3) - 1$ time slots until node 2 gets the access and then 1 more slot for the transmission itself. Note also that we can write

$$\nu_1 = \sum_{i=1}^{T} \rho_i,$$

where ρ_i is the time taken by node 3 to successfully transmit one packet. Note that the random variables $\rho_1, \rho_2, ..$ are independent and each of them has a Geometric distribution with parameter $1/2$. Indeed, when all nodes are non-empty, the probability for node 3 to successfully transmit in any given time slot is equal to $1/2$, hence, it will take a $Geom(1/2) - 1$ time slots until node 3 gets access to the channel and then an additional slot for the transmission. Taking into account our representations for ν_0 and ν_1, we write

$$\mathbb{E}\tau_1 = 3 + 2\mathbb{E}T. \tag{7.3}$$

The expected value of the number of transmissions of node 3 over a typical cycle now remains to be found. For this we will use a method similar to the one used to obtain the value of the expected length of the busy cycle in an $M/G/1$ queue. It is clear that every packet that is transmitted by node 2 over $\rho_1 - 1$ slots generates its own cycle, and the distribution of the number of successful transmissions of node 3 in this cycle is exactly the same as the number of successful transmissions of node 3 in the cycle τ_1. Moreover, all these cycles are independent. Hence, we can write

$$T = 1 + \sum_{n=1}^{\rho_1 - 1} \mathbb{1}_{z_2(n)=1} T_n, \tag{7.4}$$

where $\mathbb{1}_{z_2(n)=1}$ is 1 if node 2 is transmitting at time slot n, and is 0 otherwise, and where each T_n is distributed as T. Note that in any time slot during the time slots when node 3 is not transmitting (implying also that node 0 is not transmitting), node 2 has a probability of $1/2$ of getting access to the channel (nodes 1 and 2 compete for the channel), which means that

$$\mathbb{E}\mathbb{1}_{z_2(n)=1} = 1/2.$$

This, combined with (7.4), implies that

$$\mathbb{E}T = 1 + 1/2(\mathbb{E}\rho_1 - 1)\mathbb{E}T = 1 + 1/2\mathbb{E}T$$

and, hence,

$$\mathbb{E}T = 2.$$

Plugging this into (7.3) leads to the claim of the theorem. $\square$

Lemma 7.3 *There exist a $\lambda_2 = 3/7$ such that the queue of node 0 does not build up if $\lambda < \lambda_2$ and grows infinitely if $\lambda > \lambda_2$.*

Proof: We shall concentrate on the case $\lambda > \lambda_1$, so that queues of nodes 1 and 2 are non-empty with probability 1. Assume that the queue of node 0 is non-empty. We shall use Foster-Lyapunov techniques. Consider

$$\mathbb{E}\left(b_0(\tau_1) - b_0(0) | b_0(0) = b_0\right),$$

where τ_1 was defined in the proof of Theorem 7.2. It is clear that during τ_1 time slots node 0 will on average receive $\lambda\mathbb{E}\tau_1 = 7\lambda$ new packets. Let us now count how many packets on average will leave node 0 over a typical cycle. Recall the representation

$$\tau_1 = \nu_0 + \nu_1$$

used in the proof of Theorem 7.2. Note that during the ν_1 time slots node 0 will transmit exactly as often as node 3, i.e., will transmit T (see again the proof of Theorem 7.2) packets. Among the ν_0 time slots there are $Geom(1/3) - 1$ time slots when nodes 0 or 1 transmit, and node 0 transmits on average in exactly half of them. Taking this into account, we can conclude that the average number of packets transmitted by node 0 over a typical cycle is equal to $1/2(\mathbb{E}\nu_0 - 1) + 2 = 3$.

Now we write

$$\mathbb{E}\left(b_0(\tau_1) - b_0(0) | b_0(0) = b_0\right) = 7\lambda - 3,$$

concluding the proof based on the standard arguments of the Foster-Lyapunov type. □

7.4 Experimental Validation

We validate the results from our simulations and our model on a testbed composed of five wireless routers.

7.4.1 Topology Description

The testbed is composed of 5 Asus WL-500gP routers in which we change the mini-PCI WiFi card to an NMP-8602 Atheros card in order to benefit from the open-source driver. Each router runs the OpenWRT firmware [OPE] with the Mad-Wifi driver [MAD] that we extended with new commands: (i) to modify the maximal buffer size (from 50 to 100 packets), and (ii) to monitor the queue size evolution. Each node has a single interface that operates in the the channel 13 of IEEE 802.11g with the RTS/CTS mechanism disabled (i.e., standard setting) and with the data rate set to 1 Mb/s. We chose to deactivate the auto-rate and set a fixed rate instead, in order to limit to the minimum the effect of factors external to our study. Moreover, we avoid the problem of interference by launching our measurements during the night.

We deploy the topology over one floor of a building as depicted in Figure 7.4. As opposed to the simulation and the mathematical analysis, the measurements in a real testbed show that different links do not generally have the same capacity. We

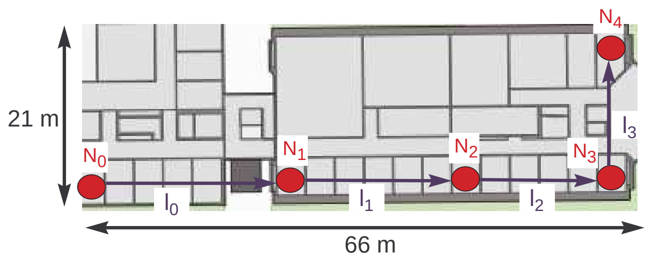


Figure 7.4: Deployment map of the testbed composed of five Asus WL-500gP wireless routers that form a 4-hop network.

present in Table 7.2 the link capacity that we measured each minute for 660 seconds (i.e., 11 measurement points). From these measurement points, we compute and show the median value together with the 50% confidence interval. We perform all our throughput measurements by using the `iperf` software. From our results, we see that: (i) the link l_2 is the bottleneck of the network and (ii) the 2-hop interference model can accurately capture some practical deployments as it is the case for the one of Figure 7.4.

7.4.2 Measurement Results

We consider the 4-hop flow going from node 0 to node 4 with the bottleneck in the 3^{rd} hop l_2. We perform our measurements by launching traffic at different rate λ for 660 s and measuring the queue evolution and the per-minute throughput μ. Then we compute the median throughput that we depict in Figure 7.5.

First, we note that our experimental results show a similar trend as the simulation results introduced in Figure 7.1 (except for the case of the two outliers at 1 Mb/s and 2 Mb/s that we discuss later). Even though we never have exactly $\lambda = \mu$ in practice due to the packet losses due to collisions or poor channel conditions, we note that the throughput μ increases as a function of the source rate until a first transition point $\lambda_1 = 125$ Kb/s ($\mu_1 = 113$ Kb/s). Moreover, the logs show that this transition point λ_1 corresponds to the threshold after which the queue of one relay node builds up (it is node 2 in our experiment due to the low capacity of l_2), before the queue of the source.

This queue build-up is synonym of wasted resources due to the packets that are lost because of buffer-overflow. It results in a reduction of throughput with $\mu_2 = 102$ Kb/s.

Then, we investigate the presence of two outlayers at $\lambda = 1$ Mb/s and $\lambda = 2$ Mb/s. It is interesting to note that, for both rates, the median throughput $\mu = 108$ (109) Kb/s is close to the optimum, but with a large confidence interval range. Moreover, our logs show that the queue of the source node 0 already builds up (i.e., saturated) at a rate of 500 Kb/s. This is even more counter-intuitive as we would expect that once the source rate is above the saturation threshold of the source, it would have no influence on the end-to-end throughput. After multiple repetitions of the experiment to validate this artifact, we conjecture that it is due

Link	Median Throughput	Confidence Interval
l_0	836 Kb/s	$[836 - 839]$
l_1	858 Kb/s	$[857 - 863]$
l_2	222 Kb/s	$[213 - 230]$
l_3	830 Kb/s	$[824 - 837]$

Table 7.2: Measurements of the link capacity l_i for each node i when transmitting alone. We show the median throughput with the 50% confidence interval.

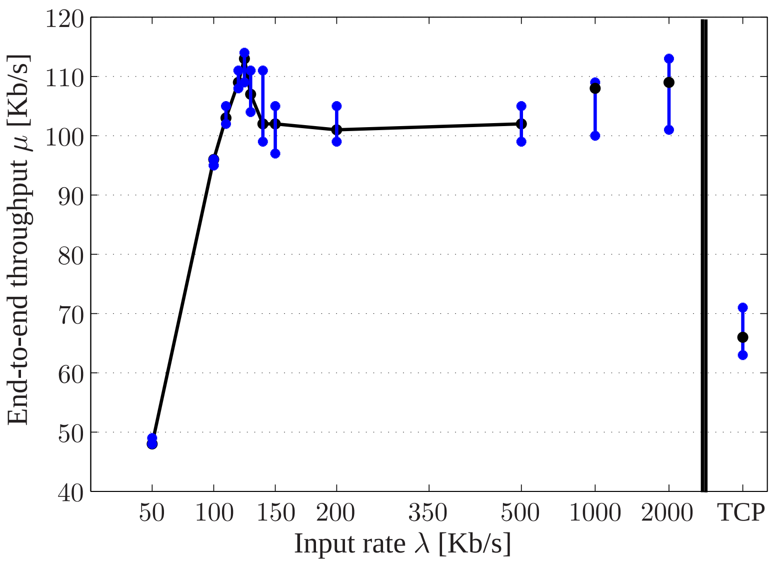


Figure 7.5: Measurement results showing the evolution of the end-to-end throughput depending on the source rate λ. We plot the median value together with the 50% confidence interval.

to the performance limitation of our wireless routers that run on a 266 MHz CPU. Indeed, we note that the packets generated at node 0 consume the device resources: (i) when they are generated at the application layer by `iperf`, and (ii) when they are passed to the MAC driver that has to check its queue to decide whether to accept the packets or not. This extra workload on the machine slows down node 0 that ends up transmitting less packets on the link l_0 and this results in an increase of both the median and the variability of the throughput.

Finally, we run an experiment with TCP traffic for comparison purposes and we find that it only reaches a throughput of 66 Kb/s, which corresponds to only 58% of the best performance achieved by UDP at μ_1. Obviously a direct comparison between TCP and UDP is not fair, because TCP delivers more than only congestion control (i.e., it guarantees reliable and in-order packet delivery). We find it interesting to note that the gap between UDP and TCP is large. Hence, there is still a significant room for improvement for new protocols that decouple the problem of congestion-control from the problems of reliable in-order delivery, and that deal with them with separated mechanisms at a different layers.

7.5 Hop-by-Hop Congestion Control

IEEE 802.11 is unable to reach the theoretical network capacity, even in simple linear topologies. Indeed, given an optimal capacity μ^* ($\mu^* = 1/3$ in linear networks with 2-hop interference model) we show, both in the simulations of Table 7.1 and analytically in Theorem 7.1, that there exist no source rate λ such that the network achieves a throughput $\mu = \mu^*$. A key implication of this finding is that it is *impossible for any end-to-end congestion control scheme that runs over IEEE 802.11 to be throughput-optimal*. This supports the idea that doing congestion control in a hop-by-hop manner, instead of end-to-end, allows the system to reach a higher throughput. In order to see how well hop-by-hop schemes perform, we analyze through simulations some state-of-the-art scheduling algorithms. We consider the four following scheduling schemes in addition to IEEE 802.11:

- **Policy 1 - IEEE 802.11:** Each node with a non-empty queue selects its backoff uniformly in the same interval (recall that we do not consider the exponential backoff mechanism)

$$\beta_i = U(0, 1).$$

- **Policy 2 - linear own queue policy:** This policy comes from an algorithm introduced by Shin et al. for a non-slotted time model [SSR09]. In this algorithm a node i wakes up at a clock tick and if the channel is idle it transmits with a probability $\frac{exp(W_i(t))}{1+exp(W_i(t))}$, where $W_i(t)$ is a function of $\log(b_i+1)$ with b_i being the queue size of node i. We simulated a similar policy for a slotted-time system by making each node i select its backoff β_i as a function of its queue

$$\beta_i = \frac{1}{b_i + 1} \cdot U(0, 1).$$

- **Policy 3 - logarithmic own queue policy:** Another approach studied in [SSR09] is to select the transmission probability depending on $W_i(t)$ that is a function of $\log(\log(b_i + e))$. We captured this policy in a slotted-time system by making each node i select its backoff β_i as a function of the $\log(\cdot)$ of its queue

$$\beta_i = \frac{1}{1 + \log(b_i + 1)} \cdot U(0, 1).$$

- **Policy 4 - next-hop queue policy:** This policy is inspired from the EZ-flow algorithm that we presented in Chapter 5. In this scheme each node i adapts its transmission probability depending on the queue at the next-hop node $i+1$. When the queue at node $i+1$ goes above a threshold (i.e., $b_{i+1} > b_{max}$) the transmission probability is increased (i.e. β_i divided by two), and when it goes below another threshold (i.e., $b_{i+1} < b_{min}$) the transmission probability is decreased (i.e. β_i is doubled). In our simulation, we simplify this policy by neglecting the dependence between $\beta_i(n)$ and $\beta_i(n-1)$. Instead, we consider that each node i selects its backoff β_i as only a function of the queue at its next hop

$$\beta_i = \left(1 - \frac{1}{b_{i+1} + 1 + \epsilon}\right) \cdot U(0, 1),$$

  for a small $\epsilon > 0$ that is used in order to avoid having exactly $\beta_i = 0$ for all the nodes having their next-hop node with an empty queue (i.e., $b_{i+1} = 0$).

- **Policy 5 - airtime-limiting policy:** This policy comes from an algorithm proposed by Jang et al. for TCP traffic in [JPG10]. In their scheme, the authors find the allocation time A_i of a link l_i by using the notion of: N_i (the link l_i and its interfering links), W_i (the number of flows traversing the link l_i), and NW_i (the number of flows traversing a link in N_i). Finally the allocation is obtained by trading some efficiency for stability (i.e., not taking advantage of spatial reuse) and computing

  $$A_i = \frac{W_i}{\max_{k \in N_i}(NW_k)}.$$

  Each link l_i has then its airtime transmission restricted to be not more than a fraction A_i of the time.

We study each one of these algorithms and we present the throughput evolution as a function of the input rate λ for multi-hop networks of various sizes in Figure 7.6. We obtain analytically the throughput of the airtime-limiting policy and we simulate the other policies for 10^7 time slots using the same settings as in Section 7.2.

In order to derive the throughput of the airtime-limiting policy (policy 5), we note that we have $W_i = 1$ for all i and thus we have

$$\overrightarrow{NW} = [3, 4, 4, 3]$$

for a 4-hop network and

$$\overrightarrow{NW} = [3, 4, 5, 4, 3]$$

for a 5-hop network. That gives a limitation for link l_i of $A_i = 1/4$ for all i in the case of a 4-hop network, and $A_i = 1/5$ for a 5-hop network. The advantage of this policy is that it never decreases with λ (i.e., robustness) as opposed to policy 1 and 2. However, this is achieved at the cost of a decrease in efficiency when UDP traffic is used. Indeed, when the sources send traffic at exactly the network capacity $\lambda = 1/3$ this policy only delivers a throughput of $1/4$ (75% of the optimum) or $1/5$ (60% of the optimum). One should stress that this scheme was designed for TCP, where robustness to different rates λ is critical – remember from Section 7.4.2 that TCP performs poorly over 802.11 linear networks.

As expected, our results show that, in the 4-hop case , both policies 2 and 3 (i.e., own queue policies) are throughput-optimal when the source rate remains within the network capacity $\lambda \leq 1/3$ (i.e., efficiency). For larger topologies, we find interesting results as we note that policy 2 is still throughput-optimal, whereas policy 4 appears to suffer a throughput degradation, even though it has been formally proven to be throughput-optimal [SSR09]. This simulation result illustrates the importance of having very large queues in order for policy 4 to be indeed throughput-optimal. Based on these results we can conclude that policy 2 is the policy delivering the best *efficiency*. However, this is performed at the cost of *robustness* as we note that both policies suffer from throughput degradation when the source receives packets at a rate $\lambda > 1/3$. Furthermore, we see that this degradation is a serious problem for the linear case (policy 2), but it is less severe with the $\log(\cdot)$ function (policy 4). This result supports the proposition of Shin et al. to use the $\log(\cdot)$ as it provides more robustness that the linear counter-part.

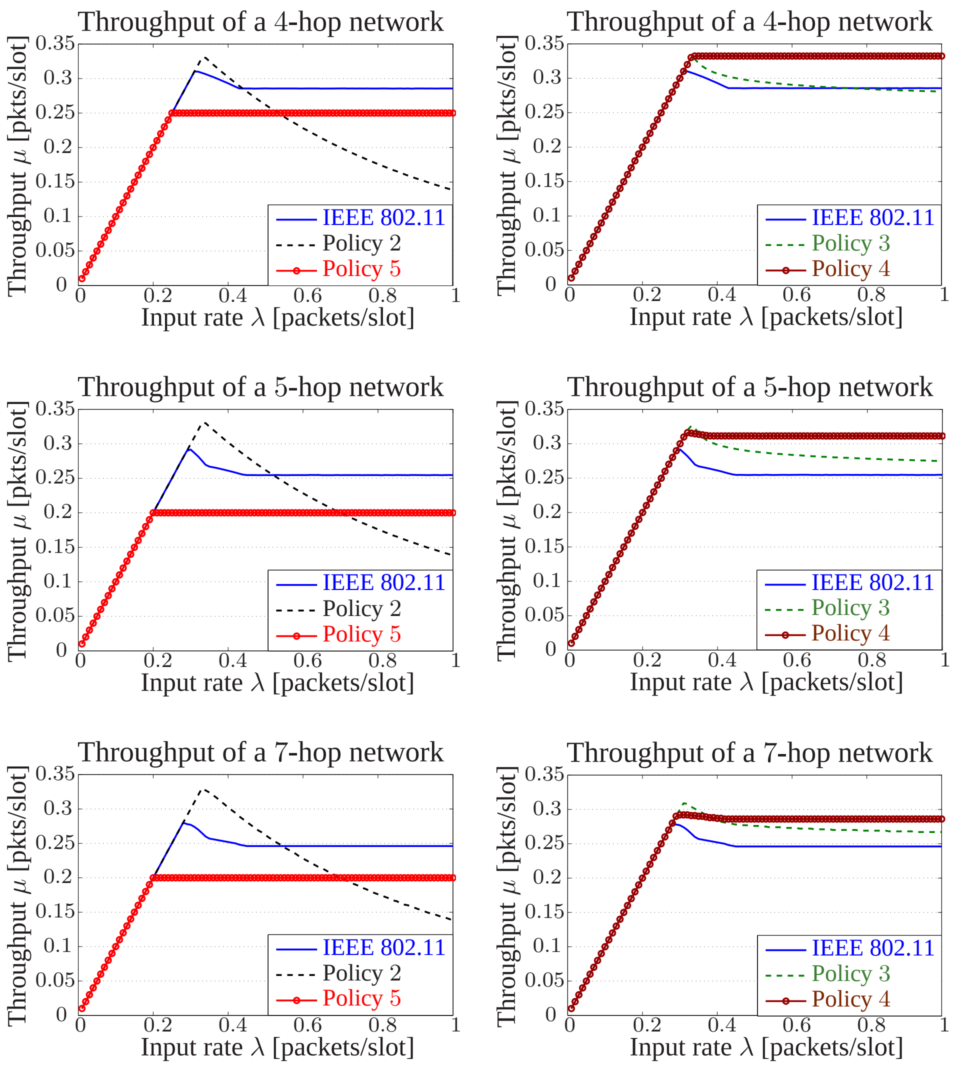


Figure 7.6: Simulations for a multi-hop network with different hop-by-hop scheduling schemes used for congestion control. Scheduling policies based on a node's own queue (policy 2 and 3) reach high throughput at capacity, whereas policies based on either the next-hop queue (policy 4) or airtime-limiting (policy 5) provide robustness to source rates λ that are above capacity.

Finally the next-hop queue policy (policy 4) results in a nice tradeoff between robustness vs. efficiency. Indeed, it also achieves the robustness property of policy 5, and it delivers a throughput that is closer to the optimum and is always better than IEEE 802.11 in all the topologies that we tested (we simulated networks up to 14 hops).

7.6 Concluding Remarks

The capacity region of a multi-hop network is usually assumed to be known in most of the recent congestion-control schemes and they, moreover, require this information in order to be throughput-optimal. Nevertheless, contrary to wired networks, wireless capacity is time-varying and usually unknown. This makes it almost impossible to know whether a network is operating within the capacity region or not.

Up to our knowledge, we are the first to study, both analytically and experimentally, how a multi-hop network behaves at different source rates either below or above the capacity. Our contributions are threefold: (i) we proved the existence of the different transition points $\{\lambda_i, \mu_i\}$ (see Figure 7.1) and explained the rationale behind them; (ii) we formally proved that it is impossible for an end-to-end congestion control scheme to be throughput-optimal if it runs over IEEE 802.11; (iii) we studied through simulations some state-of-the-art hop-by-hop congestion control mechanisms and we highlighted the tradeoff between *optimality* (throughput optimality) and *robustness* (no throughput collapse beyond capacity) that should be taken into consideration when designing new hop-by-hop scheduling algorithms.

Chapter 8

Conclusion

8.1 Discussion of the Results

Throughout this thesis we have focused on the scheduling problem that leads an IEEE 802.11 multi-hop network to be unstable, where we defined instability to happen when the queue of at least one node in the network builds up indefinitely (in practice this takes the form of buffer saturation and overflow). First, we validated experimentally the 3-*hop stability boundary* that means that in the source-saturated regime a 3-hop network is stable, whereas larger topologies are intrinsically unstable. The effects of this stability boundary have been reported experimentally by some researchers who stated that ”*with current commodity wireless technology it does not make sense to handle more than three hops*” [1]. To the best of our knowledge, we are the first to propose an analytical model to understand the root causes behind these phenomenon. In Chapter 3, we proposed a Markovian model and we introduced the notion of *stealing effect*. The stealing effect takes the form of collisions that are the result of the hidden node situation and non-zero transmission time. We showed that, interestingly, these collisions have the beneficial effect of stabilizing the 3-hop network. Indeed, a 3-hop network would be unstable if the stealing effect was not accounted for (i.e., if it has a zero probability of occurring). In practice, however, the stealing effect probability is always strictly positive due to the non-zero transmission times

In addition to explaining the instability problem, we proposed, implemented and evaluated some practical solutions to solve it. The first approach we investigated is to perform the congestion control at the MAC layer with *EZ-flow*. The key idea behind EZ-flow is to work in a hop-by-hop manner by using at each node i two separate modules (i) that passively compute the queue size at the next-hop q_{i+1}, and (ii) that automatically adapt the channel access probability accordingly by varying the contention window parameter cw_i. We showed experimentally, analytically and through simulations that EZ-flow successfully stabilizes a multi-hop

[1] Lunar project: http://cn.cs.unibas.ch/projects/lunar/

network without requiring any form of message passing.
A second approach that we considered is to perform the congestion control at the network layer instead of the MAC layer. Toward this goal, each node i uses one IP queue q_i^j per flow j (a flow is the tuple $< src\ IP; dst\ IP >$ within the mesh). Each IP queue is throttled by a rate limiter of rate ρ_i^j and all the IP queues are then scheduled to the MAC queue through a Round-Robin scheduler. Similarly to the MAC approach, the per-flow next-hop queue size q_i^j is obtained passively by taking advantage of the broadcast nature from the wireless medium. Nevertheless the channel access probability is adapted by varying the limiting rate ρ_i^j instead of the MAC contention window cw_i. This modification allows us to decouple the problem of congestion from contention, because it does not abuse the MAC contention mechanism when performing congestion control.

In order to complement the hop-by-hop congestion control scheme that operates within a flow, we proposed an additional mechanism that runs at the mesh gateway and that achieves a level of fairness between the different flows of the mesh. This algorithm called *Explore & Enhance* (E&E) optimizes a given utility function without having a prior knowledge of the network capacity region. In order to perform its task, it runs (i) exploration phases to discover the capacity region and (ii) enhancement phases to improve the utility by a gradient ascent. E&E does not require network-wide message passing to propagate the rate allocation. Instead, it only uses a broadcast message to inform its direct neighbors of the new allocation and this information is automatically propagated passively by the congestion-control scheme. We showed in a real deployment both the practicality and the efficiency of the E&E algorithm. The exact dynamics during the explore phases, however, are difficult to analyze mathematically and, even in the case of a fixed capacity region, it is hard to provide a formal proof of convergence to the optimum. In order to overcome this limitation, we are currently investigating slight variations of the E&E algorithm that are simpler to study analytically.

After having mostly focused on the source-saturated scenario, we analyzed the effect of the source rate λ on the end-to-end throughput μ of an IEEE 802.11 multi-hop networks. We reported, both through simulations and experimentally, some counter-intuitive results that showed that the relation between μ and λ is non-monotonic. We proposed an analytical model that allows us to derive some results concerning the transition points of the curve showing the relation between λ and μ for a 4-hop network. Using both our simulations and our mathematical analysis, we showed that if a multi-hop network (4 hops or larger) uses an unmodified IEEE 802.11 MAC layer, it cannot reach the network capacity μ^* for any possible source rate λ. This results implies that it is impossible for an end-to-end congestion control scheme to be throughput optimal when operating above an unmodified MAC layer. It supports, therefore, the idea of performing the congestion control in a hop-by-hop manner (at the MAC or network layer) rather than end-to-end (at the transport layer). As a result, we compare through simulations the performance of different state-of-the-art methodologies to perform hop-by-hop congestion control in a network. This study showed the important tradeoff that exists between *efficiency* (i.e., throughput-optimality) and *robustness* (i.e., no throughput collapse when the sources attempt to operate at a rate above the network capacity). Currently, most of the work in the literature focus solely on the efficiency part. The wireless capacity, however, is in practice both difficult to measure exactly and time varying. Hence, in reality it is almost impossible to guarantee that the sources always operate *within* the capacity region without being too conservative. As a result, the robustness criteria is at least as important as efficiency and we strongly believe that it should be evaluated when considering new scheduling schemes for wireless multi-hop networks

Finally, a practical outcome of this thesis is the design and deployment on the EPFL campus of the first large-scale multi-hop testbed based on IEEE 802.11. Initially we planed our network to span over the six buildings of the I&C Department, in order to support our experimental research on wireless mesh network. Furthermore, we designed our testbed to be as flexible as possible in order to be able to meet research needs in other fields. The fact that our testbed was used in various other research projects [Epi10, Bec11, JSPF$^+$10, EFLBA10] is an indicator that we reached our objective of building an efficient platform to support experimental research. For the future, we plan to continue using the testbed and we are currently looking at different possible upgrades and extensions.

8.2 Possible Extensions

In this thesis we used our Markovian model to study the stability or instability of the network using the Lyapunov-Foster criterion. Furthermore, additional studies based on this model provide more detailed information about the network behavior. An example of possible extensions has been recently performed by Guillemin et al. [GKvL10]; they further study our model in order to derive various asymptotic expressions for the stationary large buffer probabilities of a 3-hop network.
Another possible direction for the extension of our model is to adapt it to cover topologies more general than the linear ones that we focused on. Indeed, we studied more general topologies through simulations and experiments, but not analytically. The challenge there is that, using our methodology, a network containing N nodes would be modeled as a random walk in $\mathbb{N}^{N-2}$ and thus the number of states to consider in the analysis of the Markov chain would rapidly explode.

The experimental studies we performed on a real wireless multi-hop deployment allowed us to draw attention to the fact that the shared nature of the wireless medium brings both new features and new challenges.

On the one hand, our measurements showed that the multi-hop wireless capacity is highly variable through time, especially in the indoor setting. This implies that the traditional static scheduling schemes from wired networks cannot directly be applied to the wireless setting and more adaptive techniques need to be used instead. In this thesis, we followed this methodology both in EZ-flow and the Explore & Enhance algorithm, and we stress that a similar adaptive approach can be adapted to other protocols
On the other hand, the shared nature of the wireless medium also implies that additional information is available at a node (i.e., a node hears more than only the packet targeted to it). In the original IEEE 802.11 protocol, this information is wasted because at the MAC layer the nodes discard all the packets not targeted to them. However, the new trend in the network community is to take advantage of this *free* information delivered by broadcast wireless medium. Toward this goal, the nodes are set in the promiscuous mode and use all the sniffed packets in order to better perform this task. In this thesis, we showed how EZ-flow follows this strategy to derive the queue occupancy at the next-hop. Other examples following this strategy are found in topics such as in opportunistic routing. Nevertheless, we think that this topic is still in its infancy and there are still many directions to explore in order to take full advantage of the possibilities offered by the broadcast medium.

In Chapter 7 we showed that no end-to-end congestion control scheme can be throughput-optimal if it runs directly above an unmodified IEEE 802.11 layer. This results supports the idea that the congestion control in wireless multi-hop networks should be performed at the network or MAC layer instead than at the transport layer. Yet, in practice congestion control is currently performed mostly by TCP at the transport layer. The key advantages of TCP over UDP are that (i) it performs congestion control and (ii) it provides reliable in-order packet delivery.
Hence, in order to propose a credible alternative to TCP, it should match these two features. In this work, we showed that the first mission of congestion control can be performed more efficiently in a hop-by-hop manner at the network layer. The second mission of in-order packet delivery is still an open issue and an interesting extension of this work would be to develop a transport layer that guarantees in-order packet delivery without the congestion-control part. Such a new transport protocol, coupled with a hop-by-hop congestion-control mechanism (such as EZ-flow) would be a serious alternative to TCP in wireless multi-hop networks and wireless mesh networks.

Beside the scheduling problem, the experimental framework that we developed at EPFL is general enough to allow for the extension of our study on wireless multi-hop networks into many different directions. Below we describe five different directions that we studied or that are possible extensions.

Indoor Localization: The problem of localization has been solved outdoors by the Global Positioning System (GPS), but this technique is not applicable to the indoor scenarios. An alternative solution to the indoor localization problem is to use the IEEE 802.11 indoor Access Points (APs) to localize a mobile user [BP00, HLL06, KKCP10]. We consider an approach where the different nodes vary their transmission power and we evaluated its efficiency in our testbed[2] [Epi10]. Our results showed that an important factor for the precision of the estimation is density of the APs. Indeed, the denser the network, the more precise the estimation. We think that this tradeoff would be interesting to investigate analytically in more detail, in order to derive a mathematical expression of this relation.

Opportunistic Routing: Recently, opportunistic routing algorithms have been proposed to take advantage of the broadcast nature and the variability of the wireless medium [BM05, RSMQ09, LDFK09]. To evaluate the performance of these protocols, most authors only consider throughput. We agree that throughput is important, but we argue that delay and congestion metrics are at least equally important. Our large-scale multi-hop testbed is an ideal platform to evaluate opportunistic as it provides independent routes across different buildings. We are currently investigating these aspects through a deployment on our testbed [Bec11]. Once the congestion results are obtained, the next step would be to adapt the technique of congestion-control discussed in this thesis to the case of opportunistic routing. The challenge there would be to deal with the fact that the next-hop is not known beforehand and that the packets might be forwarded out-of-order.

Security Attacks: In the field of network security a large amount of problems and solutions have been studied both analytically or through simulations. We think that our testbed would be an adequate platform to evaluate experimentally different attacks and counter-measures. Indeed, we show in one of our projects that it is relatively easy to perform ARP spoofing with off-the-shelf hardware[3]. As of yet, we have not investigated more complex attacks, but this small example shows that our experimental infrastructure can be used as a tool to support experimental research in this field.

Multi-Channel: In this thesis, we mainly focused on the single-channel scenario, but an interesting extension of this work would be to study the similarities and differences between single-channel WMNs and the multi-antenna, multi-channel WMNs. Indeed, in the multi-antenna setting there are two approaches possible.
In the first approach, a form of graph coloring could be used to set each link to one specific channel so as to maximize the spatial reuse. Following such a strategy implies that a node i is very likely to use different channels when communicating with node $i-1$ or $i+1$. Therefore, either the queue estimation mechanism of EZ-flow should be adapted accordingly (when the number of channels is larger than the number of antennas, this may require an additional interface), or explicit messages need to be used for a node to inform its previous-hop of its queue size.
In the second approach, the number of channels used is the same as the number of

[2]Aziala testbed - IEEE 802.11 Indoor Localization: http://icawww1.epfl.ch/aziala (Video 3)
[3]Aziala testbed - ARP Spoofing Attack: http://icawww1.epfl.ch/aziala (Video 1 and 2)

antennas and all the nodes use all their interfaces independently (note that this is similar to having a single channel of larger capacity). If this methodology is used, the results and solution that we proposed for a single-channel mesh networks are then directly applicable to this setting.
Obviously the first approach could result in higher throughput if the number of channels is larger than the number of available interfaces. Unfortunately, the level of complexity is also higher due to the channel attribution step. It would be therefore interesting to study both approaches in order to evaluate their performance and to see how the single-channel results apply to this multi-channel setting.

Cooperative APs: As a last extension, we suggest the possibility to consider our entire testbed as an infrastructure WLAN and to study how cooperation between the APs could help significantly improve the performance [WKSK07].
In current deployments, such as the one from the EPFL IT, a user is able to hear many access points in his range. Nevertheless, he usually transmits his packets to a single destination AP. In case a packet is not correctly received by the destination AP (e.g., due to poor channel conditions), it is retransmitted by the user regardless if another AP was able to correctly receive the packet or not. Instead, a new alternative could be to have the different APs collaborate (as they are typically connected to a wired infrastructure), then they would send an acknowledgment if at least one AP receives the packet correctly.
Even though such a mechanism is a relatively drastic change to the current protocol, it is still an interesting alternative to evaluate. Indeed, there is an interesting gain that can be obtained from spatial diversity (i.e., different APs experience different collision and noise levels). Furthermore it is possible to perform the evaluation of such a mechanism in our testbed by implementing new elements in Click[4] that (i) send broadcast packets instead of unicast, and (ii) perform the new acknowledgment mechanism.

[4]Click Modular Router: http://read.cs.ucla.edu/click/

Appendix A

Useful Theorems

A.1 Foster's Theorem

Theorem A.1 (Foster [FMM95], p. 30) *Let the transition probability matrix P on the state space $\mathbb{Z}^2$ be irreducible and suppose that there exists a positive function $h : \mathbb{Z}^2 \to \mathbb{R}$ such that for some finite set S, some $\epsilon > 0$ and some positive integer-valued function $k : \mathbb{Z}^2 \to \mathbb{R}$ where $\sup_{\vec{b}\in\mathbb{Z}^2} k(\vec{b}(n)) < \infty$ the following conditions hold*

$$\mathbb{E}\left[h(\vec{b}(n+1)) \mid \vec{b}(n) = \vec{i}\right] = \sum_{\vec{k}\in\mathbb{Z}^2} p_{\vec{i}\vec{k}} h(\vec{k}) < \infty \tag{A.1}$$

for all $\vec{i} \in S$ and

$$\mathbb{E}\left[h(\vec{b}(n + k(\vec{b}(n)))|\vec{b}(n) = \vec{i}\right] \leq h(\vec{i}) - \epsilon k(\vec{b}(n)) \tag{A.2}$$

for all $\vec{i} \notin S$. Then the corresponding HMC is ergodic.

A.2 Transcience Theorem

Theorem A.2 (Transience [FMM95], p. 31) *For an irreducible Homogeneous Markov Chain (HMC) to be transient, it suffices that there exist a positive function $h(\vec{i}), \vec{i} \in \mathbb{Z}^3$, a bounded integer-valued positive function $k(\vec{i}), \vec{i} \in \mathbb{Z}^3$, and numbers $\epsilon, c, d > 0$, such that, setting $S_c = \{\vec{i} : h(\vec{i}) > c\} \neq 0$, the following conditions hold:*

1. $\sup_{\vec{i}\in\mathbb{Z}^3} k(\vec{i}) = k < \infty$;
2. $\mathbb{E}[h(\vec{b}_{n+k(\vec{i})})|h(\vec{b}_n) = h(\vec{i})] - h(\vec{i}) \geq \epsilon, \forall n$, *for all* $\vec{i} \in S_c$;
3. *for some* $d > 0$, *the inequality* $|h(\vec{i}) - h(\vec{j})| > d$ *implies* $p_{\vec{i}\vec{j}} = 0$.

A.3 Non-ergodicity Theorem

Theorem A.3 (Non-ergodicity [FMM95], p. 30) *For an irreducible Homogeneous Markov Chain (HMC) to be non-ergodic, it is sufficient that there exist a function $h(\vec{i}), \vec{i} \in \mathbb{Z}^2$, a constant d and c, such that the sets $\{\vec{i} \mid h(\vec{i}) > c\}$ and $\{\vec{i} \mid h(\vec{i}) \leq c\}$ are non-empty, and the following conditions hold for every $n \in \mathbb{N}$:*

$$\mathbb{E}\left[h(\vec{b}(n+1)) \mid h(\vec{b}(n)) = h(\vec{i})\right] - h(\vec{i}) \geq 0$$

for all $\vec{i} \in \{\vec{i} : h(\vec{i}) > c\}$ and

$$\left|\mathbb{E}\left[h(\vec{b}(n+1)) \mid h(\vec{b}(n)) = h(\vec{i})\right] - h(\vec{i})\right| \leq d$$

for all $\vec{i} \in \mathbb{Z}^2$.

Appendix B

Mesh How-To Guide

B.1 Building, Installing and Configuring OpenWRT

OpenWRT [OPE] is a great open-source firmware that allows you to install a Linux-based system in your router and to easily cross-compile your own programs for its architecture. For a first quick installation of OpenWRT, one can directly download from their website the binary source that corresponds to the router architecture. Nevertheless, we recommend more advanced users to install the OpenWRT build-tree on their computer in order to build their own image for the router and add any package they are interested in. In the remainder of this section, we describe all the step-to-step procedure to follow in order to install the release 8.09 of OpenWRT on your router (note that additional information is also available in the wiki of the openWRT website).

B.1.1 Getting the Source and Compiling on Your Computer

The first step consists in downloading the openWRT buildtree and you do so by deciding in which directory you want to install the buildtree (we call it <*your-path-to-openwrt*>) and typing the following commands:

```
# cd <your-path-to-openwrt>
# svn co svn://svn.openwrt.org/openwrt/branches/8.09
# svn co svn://svn.openwrt.org/openwrt/packages/
```

Following this procedure, you end up with two directories **8.09/** and **packages/**. Then to make all the packages of **packages/** accessible to openWRT, you need to type:

```
# cd <your-path-to-openwrt>/8.09/package
# for i in ../../packages/*/*; do ln -s $i; done
```

The next step is useful to directly include the desired configuration files directly into the built image. First we create the useful directories as follows:

```
# cd <your-path-to-openwrt>/8.09
# mkdir files
# cd files
# mkdir etc
# cd etc
# mkdir config
# mkdir dropbear
# mkdir init.d
```

Then we create the five following files inside those directories.

1st File: ”files/etc/config/system”

```
#### Set the hostname
config system
option hostname TAP1
```

2nd File: ”files/etc/config/network”

```
#### VLAN configuration
config switch eth0
option vlan0 ”1 2 3 5*”
option vlan1 ”0 5”
option vlan2 ”4 5”

#### Loopback configuration
config interface loopback
option ifname ”lo”
option proto static
option ipaddr 127.0.0.1
option netmask 255.0.0.0

#### LAN configuration
config interface lan
option ifname ”eth0.0”
option proto static
option ipaddr 192.168.1.1
option netmask 255.255.255.0

#### DMZ configuration
config interface dmz
option ifname ”eth0.2”
option proto static
option ipaddr 192.168.10.1
option netmask 255.255.255.0

    #### WAN configuration
config interface wan
option ifname "eth0.1"
option proto dhcp

    #### Wireless configuration
config interface wifi
option ifname "ath0"
option proto static
option ipaddr 10.10.10.1
option netmask 255.255.255.0
```

3rd **File: "files/etc/config/wireless"**

```
    #### WiFi settings
config wifi-device wifi0
option type atheros
option channel 13
option agmode 11b

    # REMOVE THIS LINE TO ENABLE WIFI:
# option disabled 1

config wifi-iface
option device wifi0
option network wifi
option mode adhoc
option ssid aziala
option bssid 02:ca:ff:ee:ba:be
option encryption none
```

4th **File: "files/etc/dropbear/authorized_keys"**

```
    # File containing the SSH public keys of all the authorized machines:
ssh-rsa AAA...VhZw== adel@adel-desktop
ssh-rsa AAA...ISqw== julien@icsil1-pc12
...
```

...

5th File: "files/etc/init.d/done"

```
#!/bin/sh /etc/rc.common
# Copyright (C) 2006 OpenWrt.org

    # REGULAR TASKS to be done when booting:
START=95
boot() {
[ -d /tmp/root ] && {
    lock /tmp/.switch2jffs
    firstboot switch2jffs
    lock -u /tmp/.switch2jffs
}
# process user commands
[ -f /etc/rc.local ] && {
    sh /etc/rc.local
}
# set leds to normal state
. /etc/diag.sh
set_state done

    # Then add your OWN TASKS to be done when booting:
# TASK 1: Launch WPA
ifconfig eth0.1 promisc
wpa_supplicant -c /etc/config/wpa_supplicant/wpa_supplicant.conf -i eth0.1 -D ro-
boswitch -B

# TASK 2: Set the channel rate to 1M
iwconfig ath0 rate 1M

#TASK 3: Mount USB drive
mkdir /root/mnt
mount /dev/sda1 /root/mnt
}
```

Once you are done editing the above configuration files, you are almost ready for cross-compiling in your computer an image that will be ready to be flashed on your wireless router. The only remaining step consists in selecting your router profile (to set for which architecture the compilation should be done) and the programs that you want to be included as packages in the image. To do so, you need to type:

# **make menuconfig**
and this makes you enter a pretty intuitive graphical configuration menu with the following categories.

- **Target System**: This fields allows you to specify the platform of your router. Our Asus WL-500gP routers are based on the Broadcom BCM94704 platform, thus we select ”Broadcom BCM947xx/953xx [2.6]”. The number in bracket ([2.4] or [2.6]) is the version of the Linux kernel that is selected. In the earlier version of our testbed, we had our routers running the [2.4] version without any problem. Recently, we flashed all our router with a [2.6] version for a reason of compatibility with the Click package [KMC$^+$00]. In case you are using other routers than the Asus WL-500gP and do not know the platform they are based on, you may find this information on [1].

- **Target Profile**: This field allows you to specify the model of your router.

- **Base system**: This category allows you to include additional library in the image of your router (this is useful if you start coding your own program that require a specific library).

- **Network**: This category provides you with a huge amount of package that you can include in your image. However, before adding too many packages, you must keep in mind that the more selected packages, the larger the size of your image after compilation and the allowed image size is restricted by the Flash memory of your router (4Mb for most routers and 8Mb for the Asus WL-500gP [1]). In our testbed, the package we were interested in adding to the standard configuration are: click, iperf, tcpdump and wpa-supplicant[2].

- **Kernel modules**: This category allows you to include useful modules and driver. In our case, we were particularly interested in (i) adding the open-source madWiFi driver to our image by selecting *kmod-madwifi* in the ”Wireless Driver” directory; (ii) adding USB support to benefit from 2Gb extra memory through USB sticks. To do so, we had to select the modules *kmod-usb-core*, *kmod-usb-ohci*, *kmod-usb-storage*, *kmod-usb-uhci* and *kmod-usb2* in the ”USB Support” directory. Additionally, we also had to include the modules *kmod-fs-ext2*, *kmod-fs-ext3*, *kmod-fs-hfs*, *kmod-fs-vfat*, *kmod-nls-cp437*, *kmod-nls-cp850*, *kmod-nls-iso8859-1* in the ”Filesystems” directory.

The above list describes the package that we found, given our needs, the more useful to be included in our image. However, it is clearly not an exhaustive description of the multiple possibilities of available configurations, and we would recommend the interested readers to go quickly through the different categories of the

[1]`http://wiki.openwrt.org/oldwiki/tableofhardware`

[2]As explained further in Section B.2.3, we are interested to have 802.1X (WPA) available on the routers. However, the default version available in openWRT does not work with our configuration. Section B.2.3 explains what to do in a scenario similar to ours.

configuration menu in order to have a clearer idea of what is available. Moreover, if you know the name of a program you would like to include without knowing its location on the configuration menu, a useful command is the search command that is obtained by typing ”/” in the menu.

Now that you are done with the configuration, you are ready to launch your first compilation of OpenWRT by typing:

```
# make V=99
```

where the ”V=99” parameters provides you with all the debugging messages (just type **make** if you do not want to see them). During this first compilation, your computer needs to be connected to the Internet as openWRT will download the needed source code and cross-compile it. All this process takes a pretty long amount of time for the first compilation, so you may want to launch it over night. Note that OpenWRT only compiles everything at the first compilation (or if you execute a **make clean**). In the future compilations, only the modified files will be compiled and thus, the whole process will be significantly faster.

If you followed all the previous steps correctly and went through the compilation process without any errors, you should now obtain, in the *bin/* directory, different images *openwrt-*.trx* and *openwrt-*.bin* (the *.trx* is the one used for the Asus router). The next step is then to flash your router with the obtained image and we describe two ways of going it either through: (i) TFTP or (ii) SSH.

Flashing the image through TFTP

In order to follow this procedure, you need to start your router in the TFTP mode. For the Asus WL-500gP, this is done by keeping pushed down the RESTORE button (in the back of the router), while plugging in the power cord. If the Asus successfully started in TFTP mode, you notice it by seeing the READY light blinking. Once in this mode, the router has a TFTP server running that accepts a *.trx* image (i.e., the firmware) to be flashed. To do so, you need to plug your computer on the LAN port of the Asus router and type the following command

```
# cd <your-path-to-openwrt>/8.09/bin
# tftp 192.168.1.1
# tftp> binary
# tftp> rexmt 1
# tftp> put openwrt-brcm47xx-squashfs.trx
```

Once the TFTP upload is completed, the router initializes itself with the new firmware and this is a critical phase. Indeed, **do NOT turn off** your router during the 2 minutes following the TFTP transfer or you might break it without being able to recover. After, this 2-minute delay, you can unplug the power cord and plug it again and then you can start connecting to your router and configuring it. For details on how to do this, see the Section B.1.1

Flashing the image through SSH

This alternative way of flashing the router can be relatively useful if it happens that you can access the router in the normal mode, but not in the TFTP mode (we do not know the cause of this problem, but it sometimes happened to us). To perform your flashing through SSH, you need to plug your computer on the LAN port of the Asus router and type the following command

```
# cd <your-path-to-openwrt>/8.09/bin
# scp openwrt-brcm47xx-squashfs.trx root@192.168.1.1:/tmp[3]
# ssh root@192.168.1.1
```

Then once logged in the router TAP1, you type

```
root@TAP1# mtd write openwrt-brcm-2.4-squashfs.trx linux && reboot
```

At the end of this process the router will automatically reboot by itself (you do not need to touch the power plug) and you will be able to connect to it and configure it by following the instruction of next section.

First connection to the router and its configuration

A router that has been just flashed cannot be directly accessed via *ssh*, because the connection is blocked. To activate this connection and allow the connection through the wireless and WAN port, you need to set a password and follow the following steps:

```
# telnet 192.168.1.1
root@TAP1# passwd
root@TAP1# /etc/init.d/firewall disable
root@TAP1# iptables -F
```

After this you should be able to connect to your router in *ssh* and use your router normally. Moreover, if for security reasons you want to disable the connection via password, you can easily do that by editing the file **/etc/config/dropbear**.

B.2 Extending OpenWRT with Some Useful Packages

In the previous section, we explained how to build your own image of openWRT by including some of the existing packages directly into the image through the **menuconfig** window. However, one might be interested in adding later on some extra packages to a working router without wanting to go through the process of flashing it once again. In this section, we will show how this can easily be done and how you can easily cross-compile add your own packages/programs into the **menuconfig** window.

[3] Instead of /tmp, you might want to adapt the directory in which you copy the image in the router. Indeed, you need to copy it onto a volume with enough free space.

B.2.1 Basics for Adding a Package on your Router

After having compiled openWRT, one can see that many packages (the ***.ipk** files) are created in the folder **bin/packages**. Moreover, in case you want to create a new ***.ipk** file (for example you want to add the vpnc package to an already flashed router), you just need to select this package in the **menuconfig** window and to run **make** once again. Once the compilation process ends, you have a new package created and you install it following the procedure below (note that this example is for the Asus router that is built on a mipsel architecture, router based on another architecture should follow a similar methodology):

```
# cd <your-path-to-openwrt>/8.09/bin/packages/mipsel
# scp vpnc_0.5.3-1_mipsel.ipk root@192.168.1.1:/root
# ssh root@192.168.1.1
```

Then once logged in the router TAP1, you simply type

```
root@TAP1# opkg install vpnc_0.5.3-1_mipsel.ipk
```

B.2.2 Adding the Click Modular Router

Before describing the procedure of how to cross-compile **click** [KMC+00] for openWRT, we note that this is a relatively big package. Therefore, it might be wise to directly incorporate it in the flashed image instead of installing it afterwards with the **opkg** command.

In our deployment, we chose to use the MultiFlowDispatcher library [SL09], which allows to dynamically span sub-element at run time (e.g., in per-flow queuing, we only create a queue once a new flow appears in the network). In order to compile Click with new library included (in this case MultiFlowDispatcher), we needed (i) to add the file **multiflowdispatcher.cc** in
<your-path-to-openwrt>/8.09/build_dir/mipsel/click-1/lib,
(ii) to add the file **multiflowdispatcher.hh** in
<your-path-to-openwrt>/8.09/build_dir/mipsel/click-1/include/click, and
(iii) to modify the Makefile in:
<your-path-to-openwrt>/8.09/build_dir/mipsel/click-1/userlevel/Makefile.in
in order to add **multiflowdispatcher.o** in the **GENERIC_OBJS** variable.

Now that you know how to cross-compile **click** for your router, you might be interested in adding your own elements to Click. To do so, you might want to read a bit on the basics of how to add an element for click on a computer (i.e., without cross-compilation) on the FAQ section of the Click website [4]. Then once you understand this process, the only difference to add your element in the openWRT buildtree, is that you should add your elements in the folder:

<your-path-to-openwrt>/8.09/build_dir/mipsel/click-1/elements/local/

[4] `http://read.cs.ucla.edu/click/learning`

Note that during the debugging process of your elements you might need to re-install many times the *click*.ipk* package on your router, as any modification of one of your elements is equivalent to a new version of click that contains the new element and that needs to be installed. Personally, we find this procedure rather cumbersome and we even encounter problem re-installing click through the standard **opkg** command. Instead, we found that a quicker way to re-install click on your router (that always worked for us) is to simply copy the binary file on your router. This simple technique is performed by typing:

# scp <*your-path-to-openwrt*>*/8.09/build_dir/mipsel/click-1/userlevel/click* root@192.168.1.1:/usr/bin/

B.2.3 Adding 802.1x Support for a Secure Wired Connection

Indeed, one of the main goal of wireless multi-hop networks is to avoid using wires. However, if the network is deployed for research purposes, one should consider connecting the nodes to a wired network in order to be able to control them, deploy programs and perform maintenance tasks. If the nodes are deployed within your research institution's building(s), you hopefully will be able to use an already existing wired infrastructure.

In Section 4.3.2 we present an useful way of remotely controlling the nodes. We describe here the procedure that we followed to connect the nodes using the wired network that was already present in the buildings.

Connecting the wireless nodes to a regular wired LAN should not represent a big issue, provided that ethernet interfaces are present on your hardware. However, some institutions require the users of the infrastructure to use secure protocols such as VPN[5] or 802.1X[6] (also known as WPA).

We had successful experiences connecting our nodes through VPN with open-WRT, using the `vpnc` [vpn] open-source VPN client. However, the VPN gateway is disconnecting the nodes once every couple of hours, changing their IP addresses. A much more convenient way of connecting the routers while using a legacy secure protocol available in our institution is to connect the nodes using 802.1X. This way, each node is part of the school secure network and has a public IP address that almost never changes.

Indeed, such a configuration requires that each node connected to the wired network can access an 802.1X plug. In our case, we had to ask the IT department of the university to install one such plug in every office where we were willing to install a node, as very few such plugs were active by default.

Once the plugs in place, we installed a WPA (802.1X) client on the nodes. We use the well-known `wpa-supplicant` program [wpa] for this purpose.

.

[5]`http://en.wikipedia.org/wiki/Virtual_private_network`

[6]`http://en.wikipedia.org/wiki/IEEE_802.1X`

Here are all the steps that we follow to enable 802.1X authentication with OpenWRT 8.09 :

- Get the `wpa-supplicant-0.6.9-2` package provided by Jouke Witteveen at `http://www.liacs.nl/~jwitteve/openwrt/8.09/brcm-2.4/packages/`, or on the Aziala website [AFBT].

- Install it on the nodes using the package manager :
  `opkg install wpa-supplicant_0.6.9-2_mipsel.ipk`

- In our case, the config file is `/etc/config/wpa_supplicant/wpa_supplicant.conf` and it looks like this:

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=root
ap_scan=0
network={
key_mgmt=IEEE8021X
eap=TTLS
identity="p-aziala"
password="xXxXxXxX"
phase2="auth=PAP"
ca_cert="/etc/config/wpa_supplicant/Thawte_Premium_Server_CA.pem"
priority=5
}
```

  Of course, this file is relevant to our own configuration. In particular, the `.pem` file is the certificate required by the authenticator. In order to avoid to copy it on every node, we include it in the OpenWRT build tree[7].

- The WAN interface is eth0.1 with our configuration. We need to set this interface in promiscuous mode : `ifconfig eth0.1 promisc`

- The command to launch wpa-supplicant is:
  `wpa_supplicant -c /etc/config/wpa_supplicant/wpa_supplicant.conf -i eth0.1 -D roboswitch -B`. The option `-B` is to run wpa-supplicant as a daemon in background.

  We added this line at the end of `/etc/init.d/done` so that the nodes automatically authenticate on the wired network at boot time.

[7] `files/etc/config/wpa_supplicant/Thawte_Premium_Server_CA.pem`

B.2.4 Adding your Own Program/Package

Now that you master the technique of including existing program, you might be interested in including your own program (for example *HelloWorld.c*) in the open-WRT tree in order to cross-compile it and produce a nice *HelloWorld.ipk* file that you will be able to install on your router and share with other people. To achieve this goal, you have to follow the procedure below:

1. **Create the OpenWRT Makefile:** The role of the Makefile is to make your package appear correctly in the **menuconfig** and to tell the cross-compiler where it should look for the source code (most likely locally for your own code, but it can also be online if you provide an URL). The Makefile for your program (here HelloWorld) is easily created by typing:

   ```
   # cd <your-path-to-openwrt>/8.09/package
   # mkdir HelloWorld
   # cd HelloWorld
   # cat > Makefile
   ```

   ```
   include $(TOPDIR)/rules.mk

   PKG_NAME:=HelloWorld
   PKG_VERSION:=1
   PKG_RELEASE:=1

   PKG_SOURCE:=$(PKG_NAME)-$(PKG_VERSION).tar.gz

   include $(INCLUDE_DIR)/package.mk

   define Package/HelloWorld
   SUBMENU:=C
   SECTION:=lang
   CATEGORY:=Languages
   TITLE:=Simple ”Hello World” Program
   # URL:=http://website/.../helloworld
   # DEPENDS:=+libpcap @!mips
   endef

   define Package/HelloWorld/description
   HelloWorld is a basic program that prints the message ”Hello World” on
   standard outputs and then exits.
   endef
   ```

```
define Build/Compile
$(MAKE) -C $(PKG_BUILD_DIR) \
CPPFLAGS="$(TARGET_CFLAGS) -I$(STAGING_DIR)/usr/include -I$(STAGING_DIR)/include
\
all
endef


define Package/HelloWorld/install
$(INSTALL_DIR) $(1)/usr/bin
$(INSTALL_BIN) $(PKG_BUILD_DIR)/HelloWorld $(1)/usr/bin/
$(INSTALL_DIR) $(1)/usr/share/HelloWorld
endef


$(eval $(call BuildPackage,HelloWorld))
```

2. **Put your source code in the buildtree:** To do so, you first need to create the **.tar.gz** file that contains your program with its Makefile (note that this is a second Makefile different from previous point). For our Asus based on the mipsel architecture, you create this **HelloWorld.tar.gz** file by doing:

```
#> mkdir HelloWorld-1
#> mv HelloWorld.c HelloWorld-1
#> cd HelloWorld-1
#> cat > Makefile


CC = mipsel-linux-uclibc-gcc


all:
$(CC) HelloWorld.c -o HelloWorld


#> cd ..
#> tar czf HelloWorld-1.tar.gz HelloWorld-1/
#> mv HelloWorld-1.tar.gz <your-path-to-openwrt>/8.09/dl
```

Once you follow this process, you should be able to see and select your package through the **menuconfig** window. After selecting it and launching a compilation with **make**, you should see your package HelloWorld.ipk created under the *bin/packages/mipsel* directory and you can then install this **.ipk* file using the standard procedure shown at the beginning of this section.

B.2.5 Cross-compiling your Program without Making a Package

Creating a package is a cleaner way to transfer your program into OpenWRT. Nevertheless, you might be interested in the first debugging phase to quickly compile your program without going through the work of dealing with the Makefiles.

If that is the case, you can also easily compile your program by using the cross-compiler that are available int the *staging_dir* directory from your Openwrt buildtree. For example, in the case you want to compile your program HelloWorld.c (or HelloWorld.cpp) for the Asus WL-500gP router (mipsel architecture), you only need to type:

```
# cd <your-path-to-openwrt>/8.09/staging_dir/toolchain-mipsel_gcc4.1.2/bin/
# ./mipsel-linux-uclibc-gcc HelloWorld.c -o HelloWorld
# HelloWorld root@192.168.1.1:/root/mnt
```

B.3 Hacking the MadWiFi Driver

B.3.1 Unlocking the Modification of MAC Parameters

The IEEE 802.11e uses four different classes of service: Best Effort (BE), Background (BK), Voice traffic (VO), and Video traffic (VI). Nevertheless, by default in IEEE 802.11, all the traffic is queued as BE traffic. Moreover the parameters for BE traffic are locked by default in the MadWifi driver and this prevent us from changing MAC parameters such as CW_{min}, CW_{max}, etc. In order to unlock these modifications, the MadWifi driver needs to be hacked by commenting the corresponding part of the code in the function **ieee80211_wme_updateparams_locked()** of **madwifi-trunk-r3314/net80211/ieee80211_proto.c**. Note that a patch containing all the needed modification is available at [8].

B.3.2 Enabling MadWifi to Announce the FULL_BUFFER Status to Upper Layers

By default MadWifi does not inform the upper layer when its queue is full and it directly discards the packet instead. This is pretty annoying if one plans to use an additional queue at the IP layer (for example with Click). Indeed this queue will always be empty, because it will always gives any new packet to the MAC queue regardless of whether the MAC queue has space to accept new packets or not.

In order to overcome this problem, we patch the driver by modifying some files both in MadWifi and Linux.
First, in MadWifi the files that need to modified are

- **madwifi-trunk-r3314/net80211/ieee80211_output.c**.
- **madwifi-trunk-r3314/net80211/ieee80211_proto.h**.

[8] http://icawww1.epfl.ch/aziala/P1_unlock_BEparam_modif.patch

In **ieee80211_output.c** the modifications take place in the methods **ieee80211_hardstart()** and **ieee80211_parent_queue_xmit()** (that has its return value changed from void to int). These changes consist in returning a value (i.e., an int) in order to tell the upper layer if the MAC has space available in its queue to accept new packets or not. Finally, the change in **ieee80211_proto.h** consists simply in updating the type of the function **ieee80211_parent_queue_xmit()** from void to int.
Then in Linux, the modifications consist in using this new feedback delivered by the MAC in order to only transmit a packet to the MAC if it has enough room to accept it. The modifications of the code take place in the files **linux-2.6.26.8/net/core/dev.c** and **linux-2.6.26.8/net/packet/af_packet.c**.
For the readability of this report we do not discuss all the changes here, but we provide the links for the patch we created with details modification both for MadWifi[9] and for Linux[10].

B.3.3 Adding New Commands to Access MAC Parameters

In our research, we were particularly interesting in knowing the queue occupancy of a node, but unfortunately this information is not given by default by the MAC driver. In order to satisfy our needs, we hacked the MAC driver in order to add some new commands that allow to:

- Enable/disable the use of the 4 MAC queues existing in IEEE 802.11e (by default this is disable).
- Get/Set the maximal MAC buffer size (by default this is 50 packets).
- Access the current MAC queue occupancy.

In order to achieve this goal, we extended the *wlanconfig* command with our own commands, and we note that a similar methodology than the one we used can be use to access/modify some other parameters of interest. More specifically, the files that we needed to modify to reach our objectives were 3 files in the folder **madwifi-trunk-r3314/ath/** (see the details in[11])

- **if_ath.c**
- **if_athioctl.h**
- **if_athvar.h**

2 files in the folder **madwifi-trunk-r3314/net80211/** (see the details in[12])

- **ieee80211_output.c**

[9]http://icawww1.epfl.ch/aziala/P2_2_enable_fullBuffer_MACsignal_madwifi.patch
[10]http://icawww1.epfl.ch/aziala/P2_1_enable_fullBuffer_MACsignal_linux.patch
[11]http://icawww1.epfl.ch/aziala/P3_1_enable_additionalMAC_commands.patch
[12]http://icawww1.epfl.ch/aziala/P3_2_enable_additionalMAC_commands.patch

- **ieee80211_var.h**

and 1 file in the folder **madwifi-trunk-r3314/tools/** (see the details in[13])

- **wlanconfig.c**

B.4 Installing and Using Net-Controller

B.4.1 Installation

You can download Net-Controller from Sourceforge[14]. The latest version of the code can be downloaded with SVN using the following command:

```
   svn co https://netcontroller.svn.sourceforge.net/svnroot/netcontroller
netcontroller
```

Since the user interface is written in Python, it does not require to be compiled. One simply needs the following installed on the machine:

- `Python (>= 2.5)`
- `PyQT`
- `Matplotlib`

Then, once in the source folder of the user interface (`/ui`), it suffices to launch:

```
python NetController.py
```

`cidaemon` needs to be compiled in the following way (using `gcc` in this example):

```
$YOUR_GCC -Wall -lm -pthread -o cidaemon cidaemon.c
```

Where `$YOUR_GCC` denotes your version of `gcc`.

B.4.2 Configuration

Net-Controller uses one main configuration file and needs at least one map file, that maps node indexes to IP addresses.

[13] http://icawww1.epfl.ch/aziala/P3_3_enable_additionalMAC_commands.patch

[14] `http://sourceforge.net/projects/netcontroller/`

`netcontroller.cfg`

The file `netcontroller.cfg` describes the main options of the program. We explain them here.

- `interval` is the period (in seconds) between two updates of the plots. It can be useful to increase the plot update interval when the control network has large delays. However, setting intervals smaller than a typical RTT should not hurt much (although in this case the curves may be drawn in gray if the values are not received fast enough from the network).
- `flow_interval` is the period between two updates of the list of flows. This typically does not need to be small.
- `log_dir` is the directory in which the logs of the commands launched on the nodes will be saved.
- `trace_dir` is the directory in which the traces containing all the values that have been plotted.
- `map` is the name of the file that contains the mapping between the node indexes and their IP addresses on the control network. Specifying a valid file name is mandatory here, as pretty much every operations done by Net-Controller requires to communicate with the nodes through the control network.
- `trafficMap` is the name of the file that contains the mapping between the node indexes and their IP addresses on the actual wireless network. This is used by the traffic manager of Net-Controller in order to start/stop traffic.
- `max_commands` is the maximum number of simultaneous running commands. Each command is launched in a separate thread. If the limit is reached, new commands will wait until running commands are done. A value around 100 is probably good for most situations.
- `default_nbPoints` is the default number of points of the plots. For example, if the update period is one second, a `default_nbPoints` value of 60 means that the plots will represent the values received during the last minute, by default. Indeed, this value can be changed by the user once the GUI is launched.
- `direct_time` represents the direction of the time in the plots. If ’True’ (direct time), the plot moves from right to left. If ’False’ (reverse time), the plot moves from left to right.
- `ip` is the IP address of the central host that runs Net-Controller on the control network.

- `port` is the UDP port on which the user interface on the central host is listening. This is used to receive the values to plot and the lists of flows.
- `client_port` is the UDP port on which the nodes of the network are listening (on the control network). This is used to transmit the requests for the values to plot, or the requests for the lists of flows.

B.4.3 Using the Graphical Interface

In Net-Controller, the nodes are represented by integer indexes. After having specified a set of indexes, one can choose actions to perform related to these nodes. The main actions that are available at the time of writing are related to plots, commands or traffic.

The nodes one which perform actions can be selected on the left of the GUI ('Nodes' section). Shortcuts for selecting the nodes with indexes 1 to 4, all the nodes at once, or the node with the highest index, are provided in the form of checkboxes. To select any arbitrary set of nodes, one should use the 'other' textbox. In this textbox, it is possible to specify simple regular expressions. Set of contiguous indexes are noted using "-" between the smallest and the highest index, and several such expressions can be combined separated by ",". For example, if one wants to select the nodes $\{3, 5, 6, 7, 8\}$, the expression to use is "`3,5-8`".

Plotting stuff

The section 'Live plots' allows to select one or several parameters to plot. A parameter is just a keyword that uniquely identifies a quantity to plot. The 'Plot!' button opens a new window representing a plot of all the indicated parameters as measured in each of the selected nodes. The GUI offers shortcuts in the form of checkboxes for three parameters. 'CW' denotes the value of the IEEE 802.11 CW_{min} parameter, read directly from the Madwifi driver. For example, in our previous example, if one checks this box and clicks the 'Plot!' button, a window will appear showing the temporal evolution of CW_{min} for each one of the nodes $\{3, 5, 6, 7, 8\}$. The parameter 'buffer' denotes the occupancy of the MAC layer sending queue (here to, as read from Madwifi).

The parameter 'throughput' is a bit particular. Indeed, the throughputs that the nodes measure are related to the notion of flow. A flow denotes a tuple $< IP_{source}, IP_{destination} >$. All the measured throughputs are *per flow*, that means that the throughput represents what the selected nodes have received *for this flow*. That also means that the parameter has to denote the flow in which one is interested in. All the nodes of the network report to Net-Controller the flows they are aware of, and Net-Controller assigns unique IDs to the flows and displays them in the 'Flows' section of the GUI. Now, if one wants to plot the throughput flow with ID i, the name of the parameter is '`thr%F`i'. The checkbox 'throughput' is simply a shortcut for '`thr%F0`'.

It is possible to plot several parameters for the set of selected nodes. However, the parameters have to denote the same thing (i.e., use the same unit, since the same y-axis is used). It is currently only possible to plot several throughputs values (i.e., for several different flows) on one plot window.

An other particularity of Net-Controller is that it allows to plot parameters that are read through a Click socket [KMC+00]. In this case, a parameter name of the form '`click->element.handler`' will plot the temporal evolution of the handler `handler` of the element `element` (see Click documentation for more details about elements and handlers [KMC+00]).

If nothing is specified in the 'number of points' textbox, the default value for the number of points will be used. If the number of points is 1, the plot will display bars instead of curves.

By default, all the values that are plotted are locally stored in trace files, in the directory specified by the `trace_dir` option. If nothing is specified in the 'file name' textbox, a default name (that takes into account the exact creation time and selected nodes/parameters) will be used.

Sending commands

Once a set of nodes is selected in the 'Nodes' section, one can send some commands to all of them. By default, the checkbox 'send it over ssh (as root)' is checked. That means that all the commands that are typed in the textbox will be sent with '`ssh root@%IP `' prepended to them, where '`%IP`' denotes the IP address of the node to which the command is sent. For this to work properly, your public key should be present in the '`authorized_keys`' file related to the SSH configuration present on the nodes (for more details on how to setup SSH, see Section B.1). If you don't want to prepend '`ssh root@%IP `' (for example if the user that you want to use on the nodes is not root), you can uncheck the box. In this case, you can launch commands over the whole set of selected nodes using the following pre-defined variables: '`%IP`' (the IP address of the node) and '`%INDEX`' (its index). For example, if you want to create a local directory for each selected node, you can enter something like '`mkdir dir_%INDEX`. If you want to send a set of SSH commands with a custom user, say `joe`, you can enter something like '`ssh joe@%IP your-command`'. If you want to systematically be able to send SSH commands as `joe`, you should modify the prefix string in the `sendCommand()` method of Net-Controller (in the file `NetController.py`).

In addition, Net-Controller allows to send a special kind of commands to interact with Click handlers [KMC+00]. A commands such as '`click<-element.handler=value`' will set the handler `handler` of the element `element` to the value `value` (see Click documentation for more details about elements and handlers [KMC+00]).

The status of the command for each of the selected nodes is displayed in the 'status' textbox. The status can be among the three following states:

- `running...`: the command did not return yet.
- `done`: the command returned with an exitcode equals to zero.
- `failed`: the command returned with a non-zero exitcode.

For each command that is launched, its `stdout` and `stderr` outputs are appended to a file that contains such outputs for all the commands launched on the same node. These files are located in the directory indicated by the `log_dir` option. If the checkbox 'Auto display output of command' is checked (which is the case by default), the part of this file that concerns the last command is read and displayed on the right textbox once the command returns. That allows the user to quickly see the outputs of the commands.

The traffic manager

The traffic manager can be send through the menu `File -> Launch traffic manager` or with the keyboard shortcut `Ctrl+T`.

The goal of the traffic manager is to easily start/stop traffic between the nodes of the wireless network. The `iperf` program[15] is used to generate the traffic. Therefore, it is needed at the nodes for the traffic manager to work. The way the traffic manager works is rather simple. One simply has to specify a few options related to the traffic that is about to be generated, click the 'Start' button. Clicking 'Start' displays a new line after the current one, that allows to generate a new flow. Once a flow is properly running, its 'status' turns to 'On' and the button on the right becomes 'Stop'. Clicking 'Stop' stops the flows. It actually kills each Iperf and SSH processes individually. Therefore, it is highly recommended to wait for the status on this line to be 'Off' before trying to start a new flow.

The different option for the flow generation are:

- 'source(s)': That represents a set of nodes, that can be written using the same regular expression than for inputing the set of nodes on the main GUI (see Section B.4.3). An `iperf` client process will be launched in each of these nodes.
- 'destination': The node that acts as the traffic sink. An `iperf` server process will be launched on this node.

[15] `https://sourceforge.net/projects/iperf/`

- ’port’: The port to use. For the moment, only UDP traffic is implemented. The input for setting TCP instead should be added anytime soon. Entering nothing here will select ports automatically (from 6000 and incrementing for each new flow).
- ’duration (s)’: The duration of the traffic in seconds. Nothing defaults to 100,000 seconds.
- ’ToS’: The type of service. Can be one of the four values recognized by the MAC layer (BK, BE, VI, VO). Default is BE (the usual Best Effort traffic class).
- ’throughput (Mbit/s)’: The desired throughput can be set using a float or integer value.

Publications

Conference papers

- A. Aziz, S. Shneer, and P. Thiran. **Wireless Multi-hop Networks Beyond Capacity.** *Submitted for publication*.
- A. Aziz, J. Herzen, R. Merz, S. Shneer, and P. Thiran. **Explore, Enhance and Propagate: Fair Congestion Control in Wireless Mesh Networks.** *Submitted for publication*.
- A. Aziz, M. Durvy, O. Dousse, and P. Thiran. **Models of 802.11 Multi-Hop Networks: Theoretical Insights and Experimental Validation.** In *Proc. of ACM COMSNETS*, Bangalore, India, January 2011.
- A. Aziz, D. Starobinski, P. Thiran, and A. El Fawal. **EZ-Flow: Removing Turbulence in IEEE 802.11 wireless Mesh Networks without Message Passing.** In *Proc. of ACM CoNEXT*, Rome, Italy, December 2009.
- Adel Aziz, David Starobinski and Patrick Thiran. **Elucidating the Instability of Random Access Wireless Mesh Networks.** In *Proc. of IEEE SECON*, Rome, Italy, June 2009.
- A. Aziz, A. El Fawal, J.-Y. Le Boudec, and P. Thiran. **Aziala-net: Deploying a Scalable Multi-hop Wireless Testbed Platform for Research Purposes.** In *Proc. of Mobihoc* $S\hat{3}$, New Orleans, LA, USA, May 2009.
- A. Aziz, R. Karrer, and P. Thiran. **Effect of 802.11 adaptive Exponential Backoffs on the Fluidity of Downlink Flows in Mesh Networks.** In *Proc. of WinMee*, Berlin, Germany, March 2008.
- A. Aziz, T. Huehn, R. Karrer, and P. Thiran. **Model validation through experimental testbed: the fluid flow behavior example** In *Proc. of Tridentcom*, Innsbruck, Austria, March 2008.
- M. Raya, A. Aziz, and J.-P. Hubaux. **Efficient secure aggregation in VANETs.** In *Proc. of VANET*, Los Angeles, CA, USA, September 2006.

Journal paper

- Adel Aziz, David Starobinski and Patrick Thiran. **Understanding and Tackling the Root Causes of Instability in Wireless Mesh Networks.** *Accepted for publication in IEEE/ACM Transactions on Networking*.

Demos, Posters and Reports

- A. Aziz, and J. Herzen. **A Practical Hand-on Guide for Deploying a Large-scale Wireless Multi-hop Network.** *Technical Report*, EPFL-REPORT-160176, November 2010.
- J. Herzen, A. Aziz, and P. Thiran. **Demo Abstract of Net-Controller: a Network Visualization and Management Tool.** In *Proc. of Infocom*, San Diego, CA, USA, March 2010.
- A. Aziz, R. Karrer, and P. Thiran. **Promiting Fluidity in the Flow of Packets of 802.11 Wireless Mesh Networks.** In *Proc. of CoNEXT*, New York, NY, USA, December 2007.

Patent

- Aleksandar Damnjanovic, Adel Aziz, and Tao Luo. **Semi-persistent scheduling for traffic spurts in Wireless Communication.** *WO/2008/024890* QUALCOMM Incorporated, 2008.

Curriculum Vitæ

Adel Aziz was born in 1983 in Lausanne, Switzerland. In October 2006, he earned his M.Sc. degree in Communication Systems from Ecole Polytechnique Fédérale de Lausanne (EPFL) with a specialization in Network and Mobility, and a minor in Management of Technology. With an interest in managerial topics in addition to scientific research, he used the online course materials to pass the exams and earn his M.Sc. in Management from HEC Lausanne, in 2009. During his undergraduate studies, in 2003/2004, he was selected for EPFL's exchange program; he spent one year at Carnegie Mellon University (CMU) in Pittsburgh, Pennsylvania.

In 2006, he successfully completed his master's thesis at the Corporate R&D department of Qualcomm Inc. in San Diego, California. During his six months at Qualcomm, he worked on scheduling algorithms for VoIP traffic in LTE.

In November 2006, he joined the Laboratory for Computer communication and Applications (LCA), where he worked on his thesis under the supervision of Professor Patrick Thiran. There, he collaborated closely with Deutsche Telekom Laboratories (T-Labs), who funded his research.

During his Ph.D. work, he was a teaching assistant for the classes of Stochastic Models for Communication and Oriented Programming. In addition, he supervised eight students for their master's theses and semester projects.

In 2010, he received an Infocom travel grant and his current research interests are in developing adaptive solutions for time-varying wireless multi-hop networks.

Bibliography

[AFBT] Adel Aziz, Alaeddine El Fawal, Jean-Yves Le Boudec, and Patrick Thiran. Aziala-net: Deploying a scalable multi-hop wireless testbed platform for research purposes. In *Mobihoc Ŝ 3 2009*, http://icawww1.epfl.ch/aziala/.

[AHKT08] A. Aziz, T. Huehn, R. Karrer, and P. Thiran. Model validation through experimental testbed: the fluid flow example. In *Proceedings of TridentCom*, Innsbruck, Austria, March 2008.

[AKT08] A. Aziz, R. Karrer, and P. Thiran. Effect of 802.11 adaptive exponential backoffs on the fluidity of downlink flows in mesh networks. In *Proceedings of WinMee*, Berlin, Germany, March 2008.

[AST09] Adel Aziz, David Starobinski, and Patrick Thiran. Elucidating the instability of random access wireless mesh networks. In *Proceedings of SECON*, Rome, Italy, June 2009.

[ASTEF09] Adel Aziz, David Starobinski, Patrick Thiran, and Alaeddine El Fawal. Ez-flow: Removing turbulence in ieee 802.11 wireless mesh networks without message passing. In *Proceedings of CoNEXT*, Rome, Italy, December 2009.

[Bec11] Alexandre Becholey. Improving delay performances in wireless opportunistic routing. Master's thesis, EPFL, January 2011.

[Bia00] G. Bianchi. Performance analysis of the IEEE 802.11 distributed coordination function. *IEEE Journal on Selected Areas in Communications*, 18(3):535–547, March 2000.

[BJL08] Sem Borst, Matthieu Jonckheere, and Lasse Leskelä. Stability of parallel queueing systems with coupled service rates. *Discrete Event Dynamic Systems*, 18(4):447–472, 2008.

[BM05] Sanjit Biswas and Robert Morris. Exor: opportunistic multi-hop routing for wireless networks. In *Proceedings of Sigcomm*, Philadelphia, Pennsylvania, USA, 2005.

[BOW] *BOWL: Berlin Open Wireless Lab*. http://bowl.net.t-labs.tu-berlin.de/.

[BP00] P. Bahl and V.N. Padmanabhan. Radar: an in-building rf-based user location and tracking system. In *Proceedings of Infocom*, Tel-Aviv, Israel, March 2000.

[CBB$^+$06] Y.-C. Cheng, J. Bellardo, P. Benko, A. Snoeren, and G. Voelker nd S. Savage. Jigsaw: Solving the puzzle of enterprise 802.11 analysis. In *Proceedings of Sigcomm*, Pisa, Italy, September 2006.

[CK] J. Camp and E. Knightly. The achievable rate region of 802.11-scheduled multi-hop networks. *to appear in IEEE/ACM Transactions on Networking*.

[CKLS08] Prasanna Chapokar, Houshik Kar, Xiand Luo, and Saswati Sarkar. Throughput and fairness guarantees through maximal scheduling in wireless networks. *IEEE Transactions on Information Theory*, 54(2):572–594, February 2008.

[CZ08] Edwin K. P. Chong and Stanislaw H. Zak. *An Introduction to Optimization (Wiley-Interscience Series in Discrete Mathematics and Optimization)*. Wiley-Interscience, 3 edition, February 2008.

[DBvdVH08] Dee Denteneer, Sem Borst, Peter van de Ven, and Guido Hiertz. Ieee 802.11s and the philosophers' problem. *Statistica Neerlandica*, 62(3):283–298, 2008.

[DKS89] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Proceedings of Sigcomm*, Austin, Texas, USA, 1989.

[Dou07] O. Dousse. Revising buffering in CSMA/CA wireless multihop networks. In *Proceedings of SECON*, San Diego, CA, June 2007.

[DT06] M. Durvy and P. Thiran. A packing approach to compare slotted and non-slotted medium access control. In *Proceedings of INFOCOM'06*, Barcelona, Spain, April 2006.

[EAR] *EarthLink*. http://www.earthlink.net/.

[EFLBA10] Alaeddine El Fawal, Jean-Yves Le Boudec, and Adel Aziz. Empirical performance evaluation of data dissemination mechanims for spot applications. Technical report, EPFL, 2010.

[Epi10] Sebastien Epiney. Indoor localization using ieee 802.11 wlan. Master's thesis, EPFL, June 2010.

[ES05] Atilla Eryilmaz and R. Srikant. Fair resource allocation in wireless networks using queue-length-based scheduling and congestion control. In *Proceedings of Infocom*, Miami, FL, March 2005.

[Fle08] Glenn Fleishman. Francisco gets free wi-fi, courtesy of meraki. *Wi-Fi Net News*, January 2008.

[FMM95] G. Fayolle, V. A. Malyshev, and M. V. Menshikov. *Topics in constructive theory of countable Markov chains*. Cambridge University Press, 1995.

[FRE] *Freifunk*. http://freifunk.net/.

[GK80] Mario Gerla and Leonard Kleinrock. Flow control: A comparative survey. *IEEE Transactions on Communications*, 28(4):553–574, April 1980.

[GKvL10] Fabrice M. Guillemin, Charles Knessl, and Johan van Leeuwaarden. Wireless multi-hop networks with stealing: large buffer asymptotics. In *Proceedings ITC 2010, Amsterdam*, September 2010.

[GLS07] A. Gupta, X. Lin, and R. Srikant. Low-complexity distributed scheduling algorithms for wireless networks. In *Proceedings of Infocom*, Anchorage, 2007.

[GMSK09] Omer Gurewitz, Vincenzo Mancuso, Jingpu Shi, and Edward W. Knightly. Measurement and modeling of the origins of starvation of congestion-controlled flows in wireless mesh networks. *IEEE/ACM Transactions on Networking*, 17(6):1832–1845, 2009.

[GSK04] V. Gambiroza, B. Sadeghi, and E. Knightly. End-to-end performance and fairness in multihop wireless backhaul networks. In *Proceedings of ACM MobiCom*, Philadelphia, PA, September 2004.

[GSK08] Michele Garetto, Theodoros Salonidis, and Edward W. Knightly. Modeling per-flow throughput and capturing starvation in csma multi-hop wireless networks. *IEEE/ACM Transactions on Networking*, 16(4):864–877, 2008.

[HAT] Julien Herzen, Adel Aziz, and Patrick Thiran. Demo abstract of net-controller: a network visualization and management tools. In *Infocom 2010*, http://icawww1.epfl.ch/NetController/.

[HLL06] J. C. Hou H. Lim, L.-C. Kung and H. Luo. Zero-configuration, robust indoor localization: Theory and experimentation. In *Proceedings of Infocom*, Barcelona, Spain, April 2006.

[HRGD05] Martin Heusse, Franck Rousseau, Romaric Guillier, and Andrzej Duda. Idle sense: An optimal access method for high throughput and fairness in rate diverse wireless lans. In *Proceedings of SIGCOMM*, Philadelphia, PA, August 2005.

[IEE99] *IEEE 802.11, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, August 1999.

[Jia07] Wenyu Jiang. Accurate queue length estimation in wireless networks. In *Proceedings of PAM*, Louvain-la-Neuve, Belgium, 2007.

[JP09] Apoorva Jindal and Konstantinos Psounis. The achievable rate region of 802.11-scheduled multihop networks. *IEEE/ACM Transactions on Networking*, 17(4):1118–1131, 2009.

[JPG10] Ki-Young Jang, Konstantinos Psounis, and Ramesh Govindan. Simple yet efficient, transparent airtime allocation for tcp in wireless mesh networks. In *Proceedings of CoNEXT*, Philadelphia, USA, December 2010.

[JSPF$^+$10] Mahdi Jafari Siavoshani, Uday Pulleti, Christina Fragouli, Katerina Argyraki, and Suhas Diggavi. Erased secrets: Practical information-theoretic group secrecy. Under submission, 2010.

[JWa] L. Jiang and J. Walrand. A distributed csma algorithm for throughput and utility maximization in wireless networks. *to appear in IEEE/ACM Transactions on Networking*.

[J.Wb] J.Widmer. *NO Ad-Hoc Routing Agent (NOAH)*. http://icapeople.epfl.ch/widmer/uwb/ns-2/noah/.

[KKCP10] Anand Padmanabha Iyer Krishna Kant Chintalapudi and Venkat Padmanabhan. Indoor localization without the pain. In *Proceedings of Infocom*, Septemberl 2010.

[KMC$^+$00] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, Aug 2000.

[KMT98] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *The Journal of the Operational Research Society*, 49(3):237–252, 1998.

[KRW$^+$08] S. Katti, H. Rahul, Hu Wenjun, D.Katabi, M. Medard, and J. Crowcroft. Xors in the air: Practical wireless network coding. *IEEE/ACM Transactions on Networking*, 16(3):497–510, 2008.

[KZP06] Roger P. Karrer, Petros Zerfos, and Nischal M. Piratla. Magnets - a next generation access network. In *Poster at IEEE Infocom*, Barcelona, Spain, April 2006.

[LBDC$^+$01] Jinyang Li, Charles Blake, Douglas S.J. De Couto, Hu Imm Lee, and Robert Morris. Capacity of ad hoc wireless networks. In *Proceedings of MobiCom*, Rome, Italy, 2001.

[LDFK09] R. Laufer, H. Dubois-Ferriere, and L. Kleinrock. Multirate anypath routing in wireless mesh networks. In *Proceedings of Infocom*, Rio de Janeiro, Brazil, 2009.

[LE99] Wei Luo and Anthony Ephremides. Stability of n interacting queues in random-access systems. *IEEE Transactions on Information Theory*, 45(5):1579–1587, July 1999.

[MAD] *Madwifi/AtherosWireless Linux Driver Users Guide*. http://madwifi-project.org/.

[MF] S. McCanne and S. Floyd. *ns Network Simulator*. http://www.isi.edu/nsnam/ns/.

[MSZ06] Eytan Modiano, Devavrat Shah, and Gil Zussman. Maximizing throughput in wireless networks via gossiping. In *Proceedings of SIGMETRICS*, Saint-Malo, France, 2006.

[MW00] Jeonghoon Mo and Jean Walrand. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking*, 8(5):556–567, 2000.

[NAN] *Nantes Wireless*. http://www.nantes-wireless.org/.

[NL07] Ping Chung Ng and Soung Chang Liew. Throughput analysis of IEEE 802.11 multi-hop ad hoc networks. *IEEE/ACM Transactions on Networking*, 15(2):309–322, April 2007.

[NNCA06] N.S. Nandiraju, D.S. Nandiraju, D. Cavalcanti, and D.P. Agrawal. A novel queue management mechanism for improving performance of multihop flows in ieee 802.11s based mesh networks. *Proceedings of IPCCC*, 2006.

[OLP] *One Laptop per Child*. http://laptop.org/.

[OPE] *OpenWRT firmware*. http://openwrt.org/.

[PG93] A.K. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *Networking, IEEE/ACM Transactions on*, 1(3):344–357, Jun 1993.

[Pie94] Victor Pierobon. Patent wo/1996/008884: Massive array cellular system, 1994.

[PYC06] K. Papagiannaki, M. Yarvis, and W. S. Conner. Experimental characterization of home wireless networks and design implications. In *Proceedings of Infocom*, Barcelona, Spain, April 2006.

[PYC08] A. Proutière, Yung Yi, and Mung Chiang. Throughput of random access without message passing. In *Proceedings of CISS*, Princeton, NJ, March 2008.

[RJJP08] S. Rangwala, A. Jindal, Ki-Young Jang, and K. Psounis. Understanding congestion control in multi-hop wireless mesh networks. In *Proceedings of MobiCom*, San Francisco, CA, May 2008.

[ROO] *MIT Roofnet*. http://pdos.csail.mit.edu/roofnet/.

[RSMQ09] Eric Rozner, Jayesh Seshadri, Yogita Ashok Mehta, and Lili Qiu. Soar: Simple opportunistic adaptive routing protocol for wireless mesh networks. *IEEE Transactions on Mobile Computing*, 8:1622–1635, 2009.

[SdV10] S. Shneer and P. Van de Ven. Slotted and non-slotted csma: throughput and fairness. submitted, 2010.

[SGM$^+$08] J. Shi, O. Gurewitz, V. Mancuso, J. Camp, and E. Knightly. Measurement and modeling of the origins of starvation in congestion controlled mesh networks. In *Proceedings of Infocom*, Phoenix, AZ, April 2008.

[SL09] Harald Schiöberg and Daniel Levin. Multiflowdispatcher and TCP-Speaker. SyClick!, Nov. 2009.

[SSGG09] Theodoros Salonidis, Georgios Sotiropoulos, Roch Guerin, and Ramesh Govindan. Online optimization of 802.11 mesh networks. In *Proceedings of CoNEXT*, Rome, Italy, December 2009.

[SSR09] Jinwoo Shin, Devavrat Shah, and Shreevatsa Rajagopalan. Network adiabatic theorem: An efficient randomized protocol for contention resolution. In *Proceedings of SIGMETRICS*, Seattle, WA, June 2009.

[STL$^+$07] Björn Scheuermann, Matthias Transier, Christian Lochert, Martin Mauve, and Wolfgang Effelsberg. Backpressure multicast congestion control in mobile ad-hoc networks. In *Proceedings of CoNEXT*, New York, NY, USA, December 2007.

[TE92] Leandros Tassiulas and Anthony Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 37(12):1936–1948, December 1992.

[THE] *The Cloud*. http://www.thecloud.net/.

[TOF11] KEVIN C. TOFEL. Egypt as example: A case for mesh networks on phones. *The New York Times*, February 2011.

[vdVBS09] P. van de Ven, S. Borst, and S. Shneer. Instability of maxweight scheduling algorithms. In *Proceedings of Infocom*, Rio de Janeiro, Brazil, 2009.

[vpn] *VPNC client*. http://www.unix-ag.uni-kl.de/ massar/vpnc/.

[WJHR09] A. Warrier, S. Janakiraman, S. Ha, and I. Rhee. Diffq: Practical differential backlog congestion control for wireless networks. In *Proceedings of Infocom*, Rio de Janeiro, Brazil, 2009.

[WKSK07] Grace R. Woo, Pouya Kheradpour, Dawei Shen, and Dina Katabi. Beyond the bits: cooperative packet recovery using physical layer information. In *Proceedings of MobiCom*, Montréal, Canada, September 2007.

[wpa] *WPA-Supplicant*. http://hostap.epitest.fi/wpa_supplicant/.

[YPC08] Y. Yi, A. Proutière, and M. Chiang. Complexity in wireless scheduling: Impact and tradeoffs. In *Proceedings of MobiHoc*, Hong Kong, China, 2008.

[YS07] Yung Yi and Sanjay Shakkottai. Hop-by-hop congestion control over a wireless multi-hop network. *IEEE/ACM Transactions on Networking*, 15(1):133–144, 2007.

[YST08] Lei Ying, R. Srikant, and Don Towsley. Cluster-based back-pressure routing algorithm. In *Proceedings of Infocom*, Phoenix, AZ, April 2008.