Addressing the Challenges for TCP over Multihop Wireless Networks

Inauguraldissertation

der Philosophisch-naturwissenschaftlichen Fakultät

der Universität Bern

vorgelegt von

Ruy de Oliveira

von Brasilien

Leiter der Arbeit:

Prof. Dr. T. Braun

Institut für Informatik und angewandte Mathematik

Addressing the Challenges for TCP over Multihop Wireless Networks

Inauguraldissertation

der Philosophisch-naturwissenschaftlichen Fakultät

der Universität Bern

vorgelegt von

Ruy de Oliveira

von Brasilien

Leiter der Arbeit:

Prof. Dr. T. Braun

Institut für Informatik und angewandte Mathematik

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Der Dekan:

Bern, den 16. Juni 2005 Prof. Dr. P. Messerli

Acknowledgments

This thesis is the outcome of about four years (since April 2001) of research work at the Institute of Computer Science and Applied Mathematics (IAM) of the University of Berne. I am really grateful to many people that directly or indirectly contributed to this work. I do apologize if I do not explicitly name all of them here, but I indeed thank everyone who helped.

First I want to express my gratitude to my advisor, Prof. Dr. Torsten Braun. His trust in my work even when things seemed not to be on track was fundamental for my success. His clever advices always helped me to shape my research work toward something more meaningful. Professor Braun gave me the opportunity to work in the NCCR/MICS project in which I developed the concepts presented in this thesis. He always encouraged me to publish my research results in good conferences and so provided me great opportunities to meet other researchers from whom I got invaluable feedbacks.

I also want to thank Prof. Dr. Christian Tschudin for having accepted to conduct the "koreferat" of this work, and Prof. Dr. Oscar Nierstrasz for having accepted to be the co-examinator of the thesis.

My experience at the University of Berne was very positive. Even though the time was very short for me as a whole, I had some insightful discussions with most of my fellow PhD students. I want to thank the new PhD candidates Thomas Bernoulli, Marc Brogle, Dragan Milic, Thomas Staub, and Markus Wälchli for their kind assistance whenever I needed it. I thank Marc Danzeisen for the nice talks we had from which I learned a lot about telecommunications carriers. I thank Marc Heissenbüttel for the fruitful discussions we had in the scope of the NCCR/MICS project in which we both developed our thesis concepts. I thank Matthias Scheidegger for his insightful clues on subtle Unix/linux related issues. I want to thank Marc Alain Steinemann for the various social activities he organized very well. Special thanks go to Attila Weyland with whom I frequently discussed many general issues. I really appreciated his friendship in many instances. I also want to thank Dr. Florian Baumgartner, who has been working as a postdoc research assistant and lecturer at IAM. I really learned a lot from his work experience.

during my first year of studies but continued at the university as a postdoc research assistant and lecturer. I really learned a lot from his work experience.

I thank my former colleagues at the University of Berne. In particular, I thank Roland Balmer, Silvia Stattenberber, and Dr. Günther Stattenberger for the time we worked together at the third floor. The former colleague Marcin Michalak also deserves many thanks for his friendship and the many good technical references he frequently provided me with.

Special thank goes to Ruth Bestgen, the secretary of Professor Braun's research group. She was not only a staff member but also a friend providing me support inside and outside the University. I am forever indebt to her.

During Spring 2004, I visited Purdue University for nearly two weeks. I want to thank Prof. Dr. Bharat Bhargava for his kind invitation and for connecting me with both his researcher colleagues and his brilliant students. In particular, I want to thank Mohamed M. Hefeeda, Yi Lu, and Weichao Wang for their valuable suggestions that certainly influenced my research work. Many thanks go to Dr. Leszek Lilien for his general support during my stay there.

The work in this thesis was developed within the *National Centers of Competence in Research/Mobile Information and Communication Systems* (NCCR/MICS) project. This is a long term research project employing currently about seventy PhD students from various universities of Switzerland. The project is placed under the authority of the Swiss National Science Foundation (SNSF) to promote long term research projects in areas of vital strategic importance for the evolution of science in Switzerland.

During my stay in Berne, I was on leave from my educational institution in Brazil, the *Centro Federal de Educação Tecnológica de Mato Grosso* (CEFET-MT). I thank all my colleagues in Brazil who helped to obtain this leave of absence.

I wish to thank my family for having supported me throughout my studies. My parents Antonio and Adolfina always encouraged me to achieve high education. My siblings were also always supportive of my decisions and helped whenever they could.

I wish to reiterate my debt of gratitude to the most encouraging person for this doctoral degree, my wife Cristiane. She showed strength when I was not sure if I should pursue a PhD abroad. She was always motivating me and immensely patient throughout this work, which gave me balance. She really supported me a lot. Finally, the youngest lady I want to thank here is my daughter Rebeca. She was certainly a source of joy for the moments I just needed it. Although she is now only five-year-old, she could understand my usual lack of time. Only Cristiane and Rebeca know exactly how much effort I put into this work, and so I dedicate this thesis to them.

Contents

Chapter 1

Introduction

Wireless networks are becoming increasingly popular among corporate and home users worldwide. Users are looking forward to new technologies that allow them to communicate anytime, anywhere, and using any communication device. Toward this end, wireless communications are foreseen to play a key role in future communication systems. The primary advantages of wireless networks in comparison with their wired counterparts include flexible mobility management, faster and cheaper deployment, and ultimately easier maintenance and upgrade procedures.

The phenomenal growth of wireless communications today is largely driven by the popularity of the so-called Wi-Fi (Wireless Fidelity) networks. These are wireless networks based on the IEEE 802.11 standard. The name Wi-Fi is assigned to the standard in its various forms (i.e., including all different versions) and is used to allow interoperability among different manufacturers. Wi-Fi networks are gaining momentum toward the dominant data communication technology at home and corporate buildings worldwide. Their commercial use is already expressive in hot spots such as Internet cafes, airports, hotels, convention centers, etc. At home, more and more users are adopting Wi-Fi networks as a simple, flexible, low cost, highly convenient solution for interconnecting their various network devices. These applications generally communicate through a single wireless hop since the distance between communicating nodes or between a node and an access point (medium access coordinator) are relatively short. As a result, the 802.11 infrastructure mode is typically used in such communications. This requires a central entity (base station) coordinating the medium access requests.

In addition to the infrastructure mode, users are also starting to enjoy the ad hoc mode of 802.11 in which multiple wireless hops are used to connect two distant nodes. In ad hoc mode, nodes can communicate directly to each other (without a central coordinator) and should relay data to each other in a self-organizing fashion. This configuration is commonly referred to as multihop wireless ad hoc networks or simply *multihop wireless networks*. Thus, 802.11 is also capable of providing communication for connections spanning several wireless hops. This is a remarkable property of 802.11 that can enable effective communication among a community of nodes vulnerable to topology changes as well as fading channels.

Multihop wireless networks are emerging as a natural extension of the global Internet for scenarios where wired connections are unfeasible, impossible, or undesired. In these networks, nodes cooperate among themselves by relaying data to each other and generally can move at random. The topology of these networks can change rapidly and unpredictably as the mobile nodes change position or the wireless channel condition fluctuates. Such features require robust, adaptive communication protocols that can handle the unique challenges of these multihop networks smoothly. This chapter details the key challenges for the widely used Transport Control Protocol (TCP) in multihop networks and outlines the proposed approaches to solve the involved problems.

1.1 Motivation

TCP/IP is the natural choice for multihop wireless networks because most of today's applications such as HTTP, FTP, SMTP, and Telnet are developed to this protocol suite. Besides, the use of TCP/IP facilitates interoperation with the Internet. However, the unique features of 802.11, addressed in detail below, call for adjustments in the upper layer protocols used in the Internet today. In particular, the reliable data delivery provided by the predominant Internet transport protocol TCP is severed compromised in such networks. The larger the network the higher the degradation. To adjust TCP to these networks is therefore vital, as bandwidth is generally a very scarce resource in wireless networks.

TCP degradation in multihop networks is mostly caused by the mismatch between TCP and the MAC protocol. Even though the IEEE 802.11 standard has capability to work on ad hoc mode allowing the setup of a completely infrastructureless network, it is not optimized for scenarios with large number of hops. In fact, the standard specifies short RTS/CTS control frames to ensure that scenarios relaying on at most three hops are not impacted by the well-known hidden node problem. For more than three hops, contention collisions may arise degrading the channel quality. In general, the overhead of RTS/CTS combined with the lossy nature of the wireless channel as well as mobility can lead a TCP connection to experience very low performance. The reason is that TCP was originally designed for wired networks where such constraints do not exist. We summarize next the key challenges for TCP over multihop wireless networks.

High Channel Impairments: Unlike wired environments where a dropped packet is always associated to congestion, wireless networks face loss due to the lossy nature of its medium and may also experience loss caused by link interruption when nodes move. This may cause problems to conventional TCP because it always reduces its transmission rate when a drop is perceived irrespective of the loss nature. What is needed here is a mechanism at the sender that can discriminate the actual cause of a packet drop so the sender is able to react properly to each of the factors inducing losses. Past work addressing this problem have serious limitations such as high processing overhead and complete dependence on network explicit signaling, which justifies further investigations on this issue. Our proposal for packet loss discrimination using a lightweight fuzzy logic engine for these networks is synopsized in section 1.2.

High MAC Contention and Collisions: In order to ensure reliability, TCP relies on acknowledgment packets from receiver to sender establishing a bidirectional flow of data and ACKs. This is a very costly strategy in multihop wireless networks. First because of the significant MAC overhead associated to an ACK transmission despite the much smaller ACK size relative to a data packet. This happens because of both the RTS/CTS control frames exchanged before any packet transmission and the random backoff procedure that follows any unsuccessful transmission attempt. Yet, data and ACK flowing in opposite directions are highly susceptible to collide inside the network. Hence, TCP should avoid sending redundant ACKs under favorable conditions toward optimal bandwidth utilization. Traditional approaches addressing this problem have proposed to reduce the number of ACKs injected into the network in a static fashion. This is not feasible because the network condition changes and redundant ACKs may be crucial to the end-to-end performance under certain conditions. We summarize in section 1.2 our smart acknowledgment strategy at the receiver to optimize bandwidth utilization in a completely dynamic and adaptive manner.

Low Energy Resources: Multihop wireless networks are composed of mobile nodes that are presumably battery powered. Thus, it is important that the involved protocols find a well balanced compromise between performance and energy consumption. The main source of energy wastage in a TCP implementation over multihop networks is the self-induced retransmissions due to the poor interaction between TCP and 802.11. While various energy saving techniques for link and network layers are emerging, not much have been investigated on the transport layer. This thesis does not design techniques to exclusively reduce energy consumption, but the concepts involved in the contributions of this thesis certainly do not consume more energy than traditional approaches. In particular, the proposed smart acknowledgment strategy summarized below is very much energy efficient by reducing contentions and collisions in the wireless medium.

1.2 Contributions of the Thesis

The proposed solutions in this thesis address the challenges above pursuing better interaction between TCP and the 802.11 protocol to effectively enhance end-to-end performance. From the observations above, it is intuitive that many distinct solutions are possible, and in general no single mechanism can solve all the problems but a set of mechanisms. Specifically, this thesis proposes two mechanisms that can be implemented complementarily, as explained below.

Principles

It is important to understand a few design principles followed by this thesis. The first principle concerns the feasibility of the proposals. The 802.11 protocol is a reality today, so the concepts introduced in this thesis attempt to get the most out of it rather than propose a new MAC protocol. This implies that currently only short-range multihop networks are feasible, since 802.11 cannot sustain acceptable performance for long networks regarding the number of hops end-to-end.

The second principle refers to the deployment complexity. To change every node in the network is not always a good practice, so end-to-end solutions are appealing for concentrating the changes at the end nodes. Another important principle is related to the possibility of incremental deployment. An enhanced protocol should be able to interoperate with the regular protocols already in place. The fourth principle has to do with energy efficiency. That is, solutions to improve TCP in multihop networks should not be costly in terms of energy consumption, but should be as energy efficient as possible because the nodes in place are presumably battery powered. Therefore, the contributions of this thesis are built up on the following observations:

- Only short-range multihop wireless networks are feasible today.
- End-to-end solutions minimize implementation complexity.
- Incremental deployment is a clear advantage.
- Energy efficiency is a key issue.

Strategies

Having the concepts above in mind, two different strategies were investigated in this thesis. The first one addresses the problem of discriminating the nature of dropped packets to enhance TCP sender reaction to packet loss. The second strategy improves TCP performance by mitigating the problems created by the bidirectional flow established in a TCP connection. These strategies work as follows.

Improved Error Detection: This framework relies on fuzzy logic to identify the internal network state in order to enhance TCP performance when losses are caused by reasons other than congestion. It is tailored to multihop wireless networks based on the IEEE 802.11 standard and requires changes at the end nodes only. Specifically, this is an end-to-end scheme that monitors Round-Trip Time (RTT) measurements to infer the network condition without requiring any explicit feedback from the intermediate nodes. This approach exploits the fact in these type of communication channel the delay experienced by the transmitted packets follow patterns that can be associated to specific constraints in the channel. These patterns are, however, not straightforward to be accurately recognized because of the imprecision and uncertainties typical in such delay measurements. This motivated us to pursue an intelligent algorithm that could perform pattern recognition on these measurements in an effective manner. Fuzzy logic was then chosen as an viable solution for this problem. Using fuzzy logic, the continuous and imprecise behavior of the processed information can be handled without the necessity of arbitrary rigid boundaries. Hence, our proposal relies on a fuzzy engine that matches RTT mean and variance values to the network conditions in order to distinguish losses induced by the medium from losses due to congestion. This approach is especially efficient in steady state conditions where single packet drops can degrade performance substantially. Key advantages of this scheme include simplicity, ease of deployment, low processing power requirements, and no traffic overhead at all. Evaluations show that our approach can provide good results if the input data are properly sampled.

Dynamic Adaptive Acknowledgment Strategy: This approach adds functionalities to the TCP receiver so it can adjust its ACK transmission rate on the basis of the network condition. The key idea here is to mimic the congestion control strategy at the sender in that the algorithm should be adaptive to the wireless channel fluctuations toward better performance. Using this strategy, the receiver reduces the number of ACKs transmitted when the channel is in good condition and increases it otherwise. This reduces the overall medium collisions abruptly. This approach takes advantage of the cumulative acknowledgment strategy used by TCP to reduce redundant ACKs in the shared medium. This TCP strategy permits that the receiver does not transmit an ACK for each data packet it receives, since later ACKs confirm the receipt of early ACKs. However, to reduce the amount of ACKs has to managed carefully because redundant ACKs can play a fundamental role in the protocol performance when network conditions deteriorate. Our technique addresses this issue by continually monitoring the channel state at the receiver in order to fit the network needs. In particular, it keeps track of the data packet inter-arrival intervals, so that it can take action promptly when the channel deteriorates to prevent the sender from unnecessarily retransmitting. Another important aspect considered in this approach has to do with the optimization of the spatial reuse property in these networks. The TCP sender limits its congestion window to a proper small size to avoid overloading the network as that has counterproductive effects on the end-to-end performance. The designed technique not only improves bandwidth utilization but also reduces energy consumption by (re)transmitting much less than a traditional TCP does. Additional features of this approach include proactive behavior by reducing collisions instead of only reacting to their effects, easy of deployment by being also end-to-end, no signaling traffic, and possibility of incremental deployment. Simulation evaluations show that our technique outperforms traditional TCP and existing approaches in a variety of scenarios.

In summary, the contributions of this thesis include a qualitative and quantitative analysis of the main problems faced by TCP over multihop wireless networks, an intelligent mechanism to assist TCP in discriminating the actual reason of packet losses, and a dynamic adaptive strategy for optimizing bandwidth utilization proactively by reducing unnecessary traffic.

1.3 Thesis outline

This thesis is structured into the following chapters:

Chapter 2: Although TCP has been around for over two decades, there are some of its basic mechanisms that are not straightforward to understand. Additionally, new mechanisms have been added to the protocol since its first version. This chapter briefly explains

TCP role in the Internet protocol suite, details TCP main features including its evolution over the years, and introduces the main TCP extensions relevant to this work.

Chapter 3: Multihop wireless networks are introduced in this chapter. The fundamentals of these networks are explained emphasizing the key differences between wired and wireless networks. Important features of MAC and routing protocols in ad hoc networks are discussed, and a very brief introduction to sensor networks is given.

Chapter 4: This is a background chapter on Fuzzy Logic theory for supporting the discussions in chapter 6 where our proposed fuzzy logic based approach is presented and evaluated.

Chapter 5: In this chapter, the main concerns about TCP over multihop wireless networks are addressed. To substantiate some discussions, basic simulations results performed in the framework of this thesis are included. This chapter also discusses the main existing work on TCP over multihop wireless networks. The drawbacks of each approach are identified and when possible its features are compared with our own proposals.

Chapter 6 : This chapter introduces the proposed technique for packet loss discrimination using fuzzy logic. RTT patterns under congestion and medium induced error are evaluated and the parameter settings for the fuzzy engine are then identified. The chapter also introduces the general architecture for an improved error detection mechanism using the designed fuzzy engine, and presents some relevant evaluations.

Chapter 7: The second main contribution of the thesis is introduced in this chapter. This is the dynamic adaptive strategy for minimizing the number of ACKs in transit and mitigating spurious retransmissions. The design decisions are explained and extensive simulation evaluations are presented. The basic mechanism for moderate loss rates, as well as its enhanced version for more noisy conditions are addressed.

Chapter 8: This chapter concludes the thesis outlining the main lessons learned and pointing out potential future work.

Chapter 2

Transmission Control Protocol

2.1 Introduction

Congestion control algorithms are fundamental for distributed systems like the global Internet. These algorithms prevent such systems from collapsing by excessive traffic and may save resources by avoiding useless processing. In particular, congestion control mechanisms in the end nodes are appealing as they control the traffic source directly rather than its effects inside the network . The Transport Control Protocol (TCP) was designed to adaptively fit the network capacity on an end-to-end basis and has become a universal transport protocol. This chapter introduces TCP and discusses the main concepts and mechanisms associated with this widely used protocol. The chapter starts addressing TCP's role in the global Internet by describing the key network elements needed for an end-to-end communication over the Internet. The complexity behind TCP congestion control is explained as clearly as possible, establishing necessary background for the remainder of this thesis. As TCP is continuously evolving, this chapter also addresses the major existing TCP versions, showing how significantly the protocol has evolved over the years.

2.2 TCP/IP Protocol Suite

To better understand the TCP's role in the Internet, we first review briefly the Internet structure by describing the Internet Protocol Suite. In fact, TCP/IP is synonymous with Internet, and so, it is equivalent to say Internet protocol suite or simply TCP/IP protocol suite. The reason is that the two most important and defined protocols for the Internet specified in RFC 791 [Pos81] were exactly TCP and IP, the Transport Control Protocol and the Internet Protocol, respectively. The TCP/IP protocol suite represents the set of protocols that together define the protocol stack over which the Internet runs.

The Internet protocol suite provides full interoperability among the Internet users. This means that no matter the kind of computer or operating system the millions of users may be using, they are still able to communicate with each other. Moreover, TCP/IP is an entirely *open system* in that many of the implementations are publicly available.

All these features suggest that TCP/IP protocol will certainly remain the dominant set of networking protocol in the years to come.

2.2.1 Protocol Stack

The TCP/IP protocol stack is composed of four layers of protocols, in which each layer is responsible for a specific purpose. This modularization facilitates not only interoperability among manufacturing companies but also maintenance as proposed by the generic OSI reference model that comprises seven layers.

Fig. 2.1 depicts the structure of the Internet protocol stack. The application layer is responsible for originating and removing the user data that go through the layer stack and is transferred across the network. In this layer run the most common programs such as HTTP (web), FTP (file transfer), and SMTP (email) among others. The transport layer copes with the end-to-end transmission of the data created by the application layer. The most widely used transport protocols are the User Datagram Protocol (UDP) and the Transport Control Protocol (TCP). UDP provides a non-reliable data delivery over IP while TCP guarantees data delivery for the packets transported by the network layer. Generally, every application is associated with a particular port number in the transport protocols. This association permits multiple applications to share the same transport protocol between two hosts, which is known as $multiplexing$.

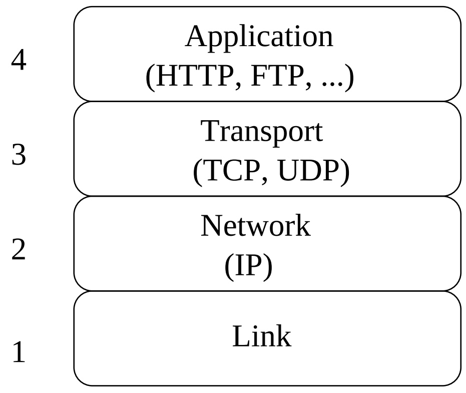


Figure 2.1: TCP/IP stack

The network layer is in charge of routing the sent packets across the network toward the receiving node, on a hop-by-hop basis. For this purpose, every node in the global Internet is assumed to have a unique IP address. The link layer (also called data-link layer or network interface layer) specifies how the packets of the network layer are transported over the physical medium connecting two nodes. This layer deals with all physical transmission details such as frame size, synchronization, frequency, etc.

The link layer makes the actual data delivery transparent to the upper layers. In this way, different physical media as well as alternative communication protocols may be deployed without relevant problems. This allows distinct communication technologies to coexist smoothly. The most popular link layer protocol for Local Area Networks (LANs) is the well-known “Ethernet” that is standardized by the IEEE as the 802.3 standard. Currently, the IEEE 802.11 standard [IEE99] is becoming more and more deployed in Wireless Local Area Networks (WLANs) as the link layer protocol.

2.2.2 Packet Encapsulation

In order to accommodate the control fields (headers) associated to each layer into a single packet to be sent through the network, the content of each layer is encapsulated in cascade as shown in Fig. 2.2. The size of the control fields in each layer is fixed, i.e., TCP and IP header size is 20 bytes each and the link header and trailer are 14 and 4, respectively. For a detailed description about the content of every header field, refer to [Ste94]. Section 2.2.4 describes the TCP header structure.

The data field size coming from the application layer may vary, but the total size should not exceed 1500 bytes to avoid packet fragmentation in the network layer. The final encapsulation results in a link layer’s frame containing header and frame for synchronization purposes, as shown in Fig. 2.2. The frame size illustrated represents the Ethernet protocol, and may certainly change for other link layer protocols.

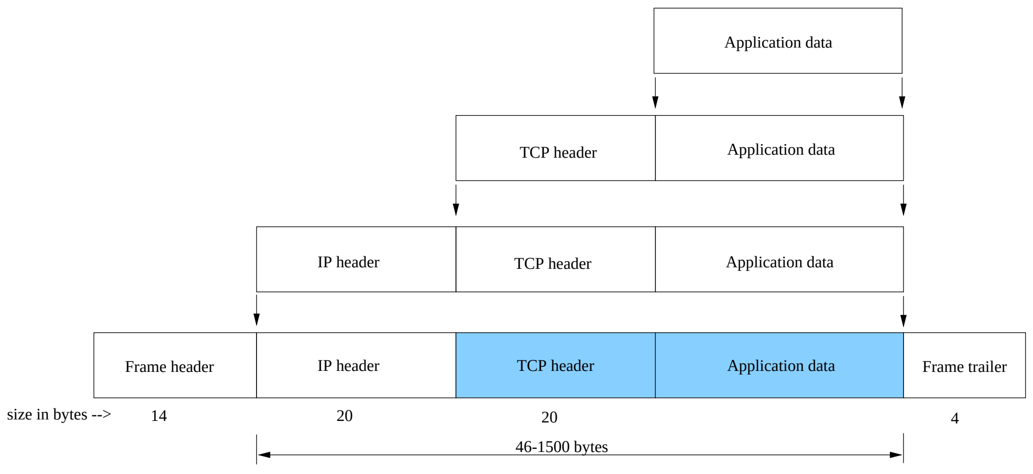


Figure 2.2: Protocols encapsulation resulting in an Ethernet frame

2.2.3 End-to-end Network Elements

Fig. 2.3 illustrates a typical end-to-end communication setup. The end nodes are identical except for the MAC layer given the fact that both use distinct channel media. The communication goes through an intermediate node that is an ordinary router. Since the purpose of the router is simply to forward the incoming data correctly, it does not comprise neither transport nor application layers, but only link and network layers. Hence, the transport layer is the first layer from bottom to top in the protocol stack that exists on an end-to-end basis. As previously mentioned, the network layer works on a hop by-hop basis. Yet, various routers can be in place, and each one of them forward the incoming data to the next router until the data reach the destination.

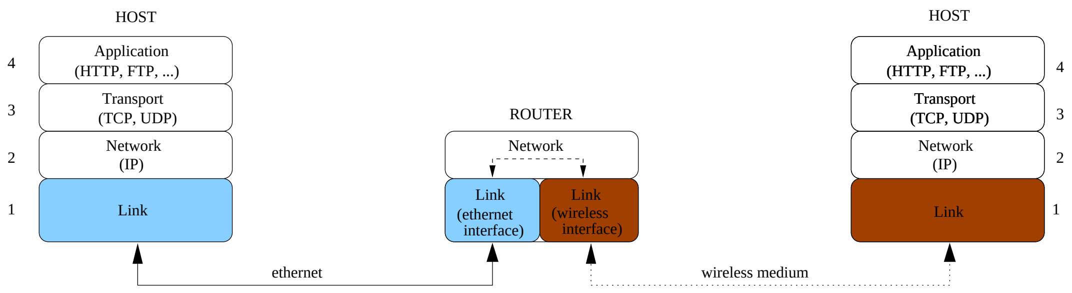


Figure 2.3: TCP/IP end-to-end communication

2.2.4 Structure of TCP Header

Before addressing the key mechanisms comprising TCP, we describe here the format of TCP header for easing understanding of the next sections. The smallest TCP header is composed of 20 bytes, but if options are used then its size may be as large as 60 bytes. TCP options are used to allow a TCP connection to carry different control fields without changing the structure of the basic header. These options are defined at the beginning of the connection between sender and receiver. Fig. 2.4 depicts the TCP header format. The fields inside TCP header are essential for managing the connection, and their purposes are as follows.

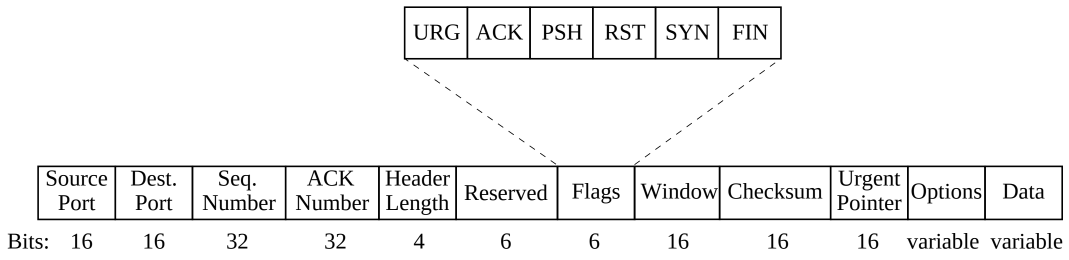


Figure 2.4: TCP header

- **Source Port** and **Source Port:** These fields identify the sending and receiving applications. This allows different TCP applications such as FTP, HTTP, and DNS to establish parallel connections between two particular hosts. These two port values combined with the source and destination fields in the IP header, uniquely identify each connection.

- **Seq. Number:** Each sent segment includes a sequence number which is increases monotonically as a function of the number of bytes transmitted. This permits sequential data delivery, which is needed for managing retransmissions.

- **ACK Number:** Contains the expected sequence number the sending host is asking for from the receiver host.

- **Header Length:** Indicates the length of the header. This is necessary because the length of the Options field is variable. By having 4 bits, this field limits the header size to 60 bytes. A TCP without any option, has a header size of 20 bytes.

- **Reserved:** Reserved and always set to zero.

- **Flags:** There are six flags defining the nature of the header, as follows:

  URG: This specifies that the Urgent Pointer in this header is valid.

  ACK: The acknowledgment number in this header is valid.

  PSH: Requires receiver to pass this data to the application as soon as possible.

  RST: Resets the connection.

  SYN: Synchronizes data and ACK sequence numbers to initiate a connection.

  FIN: Indicates that the sender is finished transmitting data.

- **Window:** This field contains the *receiver window*, which defines the number of bytes the TCP receiver is willing to accept from the sender. This provides a connection flow control governed by the receiver side, as explained in section 2.3.2.

- **Checksum:** It is calculated by the sender considering not only the header but also the data field. The receiver may check the data integrity by checking this field.

- **Urgent Pointer:** It is valid only if the URG flag is set. This field specifies a part of the data that must be sent quickly to the receiver.

- **Options:** This field carries possible options such as the *maximum segment size* (MSS), *timestamps*, *Window Scale Option*, etc.

2.3 TCP Mechanisms

In this section, we describe the main mechanisms used by the basic TCP algorithm. The explanations that follow are not intended to be exhaustive but only informative enough for better understanding of the next discussions. The unit of information passed by TCP to IP is called $segment$, but since a $segment$ ends up generating an IP packet, for each segment there is a corresponding packet. As a result, authors in general use both terms $segment$ and packet interchangeably for referring to a TCP information unit. The term packet is adopted in this thesis.

2.3.1 Connection Setup

TCP is a connection-oriented protocol, requiring connection establishment between sender and receiver for providing a reliable data transfer. Unlike a circuit-switched network that establishes circuits in the whole path between the end nodes, a TCP connection is entirely managed in the end nodes. In other words, only sender and receiver keep the connection state through the respective variables in each side. Hence, TCP defines a *virtual connection* between both hosts using the handshake process described below.

To establish the connection, either end nodes (hosts) may start the procedure by sending a request packet to the opposed side. The full procedure is commonly referred to as "three-way handshake" because it involves the exchange of three packets in total. The end node starting the connection establishment is called client host and the other side is the server host. The three-way handshake makes use of the SYN flag (1 bit) in the TCP header to mark the packet used exclusively for connection setup reasons. Fig. 2.5 illustrates the three-way handshake's exchanges [Dar81].

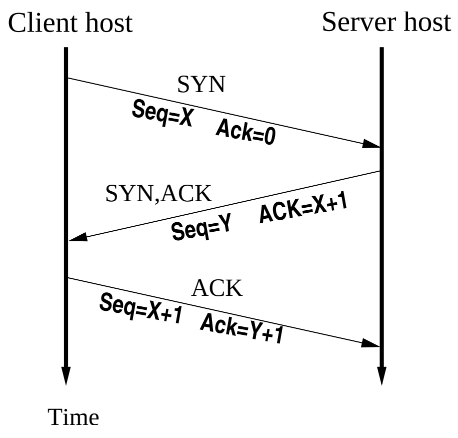


Figure 2.5: TCP three-way handshake

The client host first sends a special packet to the server host requesting a connection setup. This packet is generally named a SYN packet and contains no data but only the header with the flag SYN set to one and the desired initial sequence number (ISN) **X**. Provided that the server host is able to accept the connection, it allocates the TCP buffers and variables to the connection and sends back an acknowledgment to the client host. This acknowledgment also does not contain any data.

Similarly to the received packet by the server, this replied packet has the SYN flag set to one and the server host's desired ISN **Y**. Additionally, this packet has the acknowledgment field in the TCP header set to sequence number received plus one, i.e., its sequence number is **X+1**. This informs the client host that the request has been received and accepted, and that the receiver expects to receive the next data packet with sequence number **X+1**. This packet is generally called SYN,ACK.

Upon receipt of the acknowledgment of the server host, the client host also allocates buffers and variables to the connection, and transmits another acknowledgment to the server host. This last packet has its SYN flag set to zero and may contain data. Its sequence number is the requested one, **X**, and its acknowledgment field is also incremented by one relative to the received sequence number, i.e., **Y+1**. After these packet exchanges, the SYN flag is permanently set to zero and the regular data transmission begins.

The connection termination takes place in an analogous manner, in which any of the two end nodes may initiate the procedure. Another specific flag in the TCP header is used for closing a connection, the FYN flag (1 bit). After the connection termination, both end nodes have their resources freed.

2.3.2 Flow and Congestion Control

TCP is a window-based flow and congestion control protocol that uses a sliding window mechanism to manage its data transmission. The purpose of this scheme is to guarantee that the sender adjusts its transmission rate to meet both sender and receiver needs. Thus, the TCP sender contains a variable denoted $window$ determining the amount of packets it can send into the network before receiving an ACK. This variable changes dynamically over time to properly limit the connection's sending rate.

The sending rate of a TCP connection is regulated by two distinct mechanisms, the **flow control** and the **congestion control**. Although these mechanisms are similar, in the sense that both attempt to prevent the connection from sending at an excessive rate, they have specific purposes. This is sometimes confusing as many authors use both terms interchangeably.

Flow control is implemented to avoid that a TCP sender overflows the receiver's buffer. Thus, the receiver advertises in every ACK transmitted a window limit to the sender. This window is named receiver advertised window ($rwin$) and changes over time depending on both the traffic conditions and the application speed in reading the receiver's buffer. Therefore, the sender may not increase its $window$ at any time beyond the value specified in $rwin$.

In contrast to flow control, congestion control is concerned with the traffic inside the network. Its purpose is to prevent collapse inside the network when the traffic source (sender) is faster than the network in forwarding data. To this end, the TCP sender also uses a limiting window called congestion window ($cwnd$). Assuming that the receiver is not limiting the sender, $cwnd$ defines the amount of data the sender may send into the network before an ACK is received.

Considering both flow control and congestion control, the sender faces two limiting factors for its $window$ size, namely the $rwin$ and the $cwnd$. To conform with both control schemes, the TCP sender adjusts its $window$ to the minimum between $rwin$ and $cwnd$. In general, however, $cwnd$ is considered the limiting factor of a TCP sender because the receiver's buffer is mostly large enough not to constrain the sender's transmission rate.

TCP congestion control has been evolving over the years to detect congestion inside the network and promptly react to that by properly slowing down. Section 2.4 describes the congestion control mechanisms in more detail.

2.3.3 Retransmissions

To ensure reliability, TCP actively conducts retransmission of lost packets. Two mechanisms are used for retransmission, namely a retransmission timer and a sequence of generally three duplicate acknowledgments. A retransmission triggered by the retransmission timer is typically referred to as **retransmit timeout**, and **fast retransmit** is the name given to the mechanism that triggers retransmissions by the duplicate ACKs. As TCP is an ACK-clocked protocol in the sense that it only sends new data when an ACK is received, it needs a manner to detect complete absence of ACK from the receiver. The retransmission timer is the solution for that. During the connection's lifetime, there is always a timer running when packets are in transit. The timer is started when a given packet is sent and turned off when the ACK for that packet is received, and then turned on again when the next packet is sent. In this way, whenever a packet is transmitted but no ACK is received back, the timer expires and the packet is retransmitted.

A Retransmission by timeout is considered as the last resort for the TCP sender since it may lead the connection to unnecessary idle intervals. The fast retransmit mechanism was designed to accelerate the error detection procedure. This mechanism permits the TCP sender to detect a lost packet when three duplicate ACKs are received in sequence.

The duplicate ACKs are generated by the TCP receiver whenever it receives an out-of-order packet. By receiving such packets, the receiver infers that the expected (in sequence) packet has been dropped and then repeats the sequence number of the last in-order packet received. In short, the fast retransmit mechanism saves the time the sender would waste by waiting for the retransmission timer expiration. A retransmission by both mechanisms, retransmit timeout and fast retransmit, cause the congestion window to be reduced, lowering the transmission rate to an appropriate level. Section 2.4 explains the actions the protocol takes after a retransmission occurs.

The procedure of detecting a dropped packet by either the retransmit timeout mechanism or the fast retransmit mechanism is generally called **error detection**. Yet, the packet retransmission along with the *cwnd* reduction is termed **error recovery** [TM02].

2.3.4 Timeout Interval Computation

TCP uses the retransmission timer to ensure data delivery when no feedback from the receiver reaches the sender. The duration of this timer is referred to as RTO (retransmission timeout). RFC 2988 [PA00] is the most up-to-date specification for computing RTO. This RFC is a refinement of the algorithm proposed by Jacobson in [Jac88]. The algorithm specified in RFC 2988 is describe below.

A TCP sender maintains two state variables for computing RTO, the smoothed round-trip time ($SRTT$) and the round-trip time variation ($RTTVAR$). Additionally, a clock granularity of G seconds is assumed in the computation. As described in RFC 2988, the rules governing the computation of $SRTT$, $RTTVAR$ and RTO are as follows.

1. Until an RTT measurement has been made for a packet sent between sender and receiver, the sender should set RTO to three seconds.

2. When the first RTT measurement R is made, the sender must set:

   $SRTT = R$

   $RTTVAR = R/2$

   $RTO = SRTT + \max(G, K \cdot RTTVAR)$, where $K = 4$.

3. When a subsequent RTT measurement R' is made, the sender must update the variables as follows:

   $RTTVAR = (1 - \beta) \cdot RTTVAR + \beta \cdot |SRTT - R'|$

   $SRTT = (1 - \alpha) \cdot SRTT + \alpha \cdot R'$

   α and β are normally set to 1/8 and 1/4, respectively

   After the computation, the RTO must be updated:

   $RTO = SRTT + max(G, K \cdot RTTVAR)$

4. The minimum value of RTO should be one second, and the maximum one may be any value above sixty seconds.

When not using timestamps option [JBB92], RTT samples must not be taken for packets that were retransmitted, as specified in the Karn's algorithm [KP87]. Additionally, the RTT measurements are usually taken once per RTT. The recommendations of RFC 2988 for managing the retransmission timer are:

1. Every time a packet containing data is sent (including retransmission), if the timer is not running, start it running so it will expire after RTO seconds.

2. When all outstanding data have been acknowledged, turn off the retransmission timer.

3. When an ACK is received acknowledging new data, restart the retransmission timer so that it will expire after RTO seconds.

   When the retransmission timer expires, do the following:

4. Retransmit the earliest packet that has not been acknowledged by the TCP receiver.

5. The sender must double the RTO up to the limit discussed above. This procedure backs off the timer.

6. Start the retransmission timer, such that it expires after the current RTO.

2.3.5 Exponential Backoff Mechanism

A retransmission by timeout represents very likely a heavily impaired channel. Because of that, the retransmission mechanism used by TCP handles timeout occurrences in a persistent but careful way. The mechanism needs to be persistent given the high probability that the retransmitted packet is also going to be dropped. If this occurs, the retransmissions are performed insistently at every expiration of the retransmission timer. This process continues until either the packet is successfully retransmitted or the limit of attempts is reached. Assuming that the channel is already facing congestion, the retransmission attempts should not be too aggressive since this may induce more losses inside the network. To meet this requirement, the **Exponential Backoff** mechanism is implemented.

By the exponential backoff mechanism, at every unsuccessful retransmission attempt the RTO is doubled. This means that the retransmission scheme becomes more and more tolerant as the attempts follows. As mentioned above, the RTO limit may be any value above one minute, and it is generally set to 64 seconds. However, If the sender backs off this far without success, the next step is to abort the connection.

2.4 Congestion Control Mechanisms

2.4.1 Slow Start and Congestion Avoidance

In the basic TCP congestion control algorithm, whenever a dropped packet is detected by either the fast retransmit mechanism or timeout, the sender resets the *cwnd* to one. This leads the protocol to slow down, and afterward its *cwnd* increase is first governed by **slow start** and then **congestion avoidance**. Slow start causes the congestion window to be increased faster than in congestion avoidance. In slow start, for each ACK received the sender increases its $cwnd$ by one and so transmits two new data packets. When the ACKs corresponding to the two sent data arrive, the $cwnd$ is increased twice and four new data are transmitted. This is an exponential enlargement of $cwnd$. The process continues until the congestion avoidance is invoked or a dropped packet is detected. The idea of slow start is to make the connection rate to start slowly and then rapidly rises toward the communication channel capacity.

After reaching a certain rate, the $cwnd$ increasing rate should no longer be too aggressive, since that may adversely induce losses. Hence, the slow start threshold ($ssthresh$) is used to switch the $cwnd$ growth control from slow start to congestion avoidance. In contrast to slow start, congestion avoidance imposes a linear increase to $cwnd$. At the beginning of the connection and whenever the retransmission timer expires, slow start is invoked and depending on the $ssthresh$, the switching to congestion avoidance is started sooner or later.

Using packets instead of bytes to denote the congestion window size, the growth experienced by this window during slow start and congestion avoidance is generally performed as follows.

$$cwnd = \begin{cases} cwnd + 1, & \text{if} \quad cwnd < ssthresh \quad \text{(slow start)} \\ cwnd + \frac{1}{cwnd}, & \text{if} \quad cwnd \geq ssthresh \quad \text{(congestion avoidance)} \end{cases} \tag{2.1}$$

Fig. 2.6 illustrates how the congestion window varies over time. The initial value of $cwnd$ is two packets, and two drops are observed in the neighborhood of 0.2 and 0.8

seconds. Fig. 2.6 shows that both the $cwnd$ limit and the initial $ssthresh$ are higher than 16. For the sake of clarity, throughout this thesis, it is assumed that the receiver advertised window is sufficiently large so it is not a constraint for the sender.

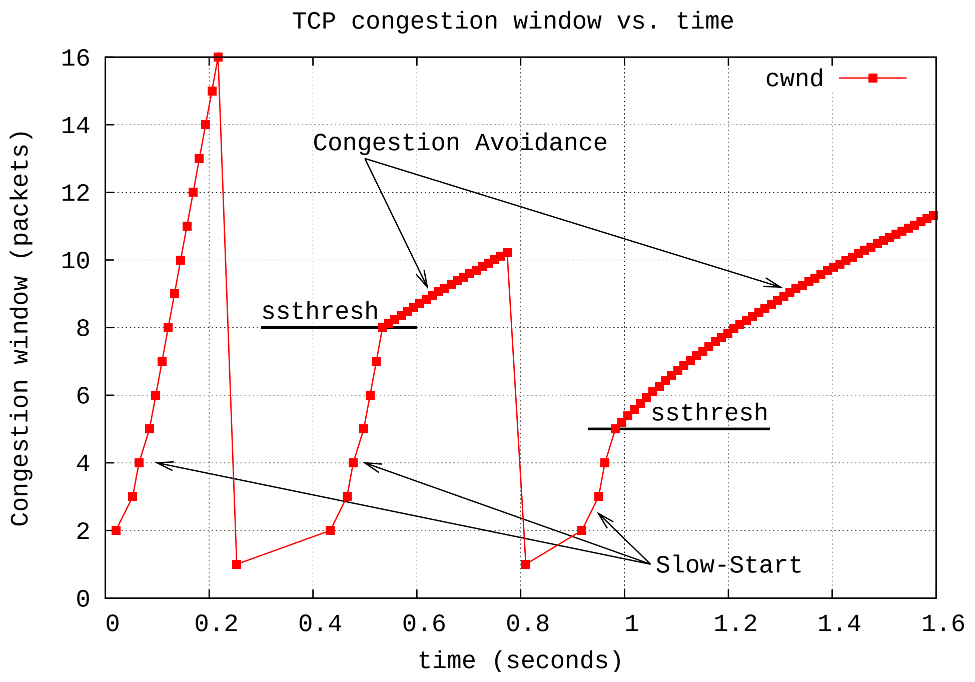


Figure 2.6: TCP congestion window evolution

The $ssthresh$ is typically initiated to its maximum value (65535). This high value ensures that the $cwnd$ increase begins with the slow start and, if no drop is experienced, the $cwnd$ is led to its limit using slow start only. This procedure provides better performance because the $cwnd$ increase in slow start is faster than in congestion avoidance, as shown in (2.1). The rationale here is that the TCP sender should probe the network resources quickly, and slow down in case it perceives lost packets.

Hence, the $cwnd$ growth in Fig. 2.6 begins in the slow start by increasing at the rate of one packet per ACK received until a dropped packet by the fast retransmit mechanism is detected. This happens when the $cwnd$ size is 16. At this point, $ssthresh$ is set to one half the current value of $cwnd$ ($ssthresh = 16/2 = 8$) and the $cwnd$ itself is reset to one. Note that a retransmission by timeout would cause the same changes in both $cwnd$ and $ssthresh$. The delay in the sender reaction would be higher, though.

After its first reduction, $cwnd$ resumes its enlargement in slow start until its value reaches the $ssthresh$ that is set to 8. Then the congestion avoidance begins and the $cwnd$ increasing rate is lowered. For the next drop, the actions are repeated with a smaller $ssthresh$ of 5, due to the smaller ongoing $cwnd$ of 10 at the instant the second drop takes place.

These mechanisms provide a very conservative behavior by abruptly slowing down the sender's transmission rate in the event of dropped packets. Further enhancements have been developed to address this issue, as it will be explained in section 2.5.

2.4.2 AIMD Congestion Control

The TCP algorithm used in current implementations is commonly referred to as **Additive Increase Multiplicative Decrease** (AIMD) congestion control. This algorithm was first proposed in [CJ89] as a general congestion control model to ensure network efficiency and fairness in a stable manner. Jacobson [Jac88] adapted the general model in [CJ89] to the basic TCP congestion control algorithm above by introducing the **fast recovery** mechanism. This modification originated the TCP version called TCP Reno, as described in section 2.5.2.

Fast recovery works in conjunction with the fast retransmit mechanism by specifying that under packet loss detection by fast retransmit, the *cwnd* should be reduced in half instead of set to one. Moreover, the algorithm should go directly to congestion avoidance rather than slow start. In short, AIMD is the same congestion control mechanism describe above except for the fast recovery mechanism, and was first implemented in TCP Reno version described below.

The name AIMD comes from the behavior of the mechanism when increasing and decreasing the congestion window. When expanding its $cwnd$ in congestion avoidance, the TCP sender additively and cumulatively increments it by $\frac{1}{cwnd}$, as shown by (2.1) in section 2.3.2. This continues until either a dropped packet is perceived or the $cwnd$ limit is reached. Using this incremental rate renders the $cwnd$ to be increased by one packet per window of data acknowledged.

When detecting a lost packet by the fast retransmit mechanism $cwnd$ is halved, which means a multiplicative decrease by two. Hence, assuming no retransmission by timeout, $cwnd$ increase/decrease in congestion avoidance occurs as illustrated in (2.2).

$$cwnd = \begin{cases} \uparrow cwnd + \frac{1}{cwnd}, & \text{(Additive Increase)} \\ \downarrow \frac{cwnd}{2}, & \text{(Multiplicative Decrease)} \end{cases} \tag{2.2}$$

AIMD has been fundamental in the Internet so far by providing the required stability for which it was developed. Nonetheless, it is too conservative for applications such as streaming multimedia or IP telephony. These kinds of applications demand stringent delay guarantees as well as low throughput variation on an end-to-end basis, which is hard to satisfy using the $cwnd$ reduction by two. So alternative mechanisms such as equation-based congestion control have been investigated in the research community, as shown below.

2.4.3 Equation-based Congestion Control

Equation-based congestion control has been proposed in the literature for two purposes. The first reason regards TCP optimization by proper modeling its algorithm. Having an accurate model for TCP is important for understanding TCP interaction with the network in terms of throughput optimization. This may render TCP viable for demanding applications such as streaming multimedia.

The second reason is concerned with fairness among TCP and non-TCP flows competing in a communication channel. In such cases, the non-TCP flows should be able to satisfy requirements of the demanding application, but should not be unfair to the competing TCP flows. So, such non-TCP flows should use a control equation to govern their sending rate friendly from the TCP flows perspective.

One equation that has been largely used to model TCP throughput is (2.3) [PFTK98, HFPW03]. Where r is the transmit rate in bytes/second; s is the packet size in bytes; R is the round-trip time in seconds; p is the loss event rate, with $0 \leq p \leq 1$, of the loss events as a fraction of the number of packets transmitted; t_{RTO} is the TCP retransmission timeout value in seconds; and b is the number of packets acknowledged by a single TCP acknowledgment.

$$r = \frac{s}{R * \sqrt{2 * b * \frac{p}{3}} + (t_{RTO} * (3\sqrt{3 * b * \frac{p}{8}}) * p * (1 + 32 * p^2))} \tag{2.3}$$

Unlike AIMD that slows down in response to a single dropped packet, equation-based congestion control uses an equation defining the maximum transmission rate for the connection. The equation relies on the loss event rate (p) that is generally computed at the receiver and sent to the sender. The accuracy of the equation is fundamental for this alternative congestion control concept. However, given the complexity of such a modeling, a complete model appears still to be too far way from being conceived.

Actually, it is not yet clear whether TCP should be extended to handle applications such as multimedia streaming or a new protocol should be developed for that. This subject represents indeed a very wide open research area [MSM+97, MSK02, HFPW03, WKST04]. For instance, [MSK02] conducts a modeling-based investigation on TCP for multimedia streaming and found that it is possible to adjust TCP toward that end as long as a few seconds of startup delay is tolerable. On the other hand, [HFPW03] specifies a completely new protocol, TFRC, to replace TCP for the stringent applications mentioned above. Only future investigations will be able to clarify which way is the best or at least more feasible.

2.5 TCP Variants

In this section, we present the main TCP variants (also termed: TCP versions or flavors) that have been investigated in the literature. Each variant has its own features tailored to a specific problem faced by TCP congestion control, and in most cases each new variant represents an evolution of the previous one.

We limit our discussion to the implementations that have been incorporated into the ns2 simulator, namely the ns version 2.1b9a that complies with the descriptions in [FF96]. Slight refinements to these implementation have been described in the RFCs 2581, 2582, and 3782 [APS99, FH99, FHAG04], but the general concepts remain unchanged.

2.5.1 TCP Tahoe

Tahoe represents the basic TCP version that was specified by Jacobson in [Jac88]. It was the first TCP designed to solve the congestion collapse then affecting the Internet. Modern TCP implementations still use most of the mechanisms developed for Tahoe, as it will be shown below.

In addition to the retransmit timeout mechanism introduced in section 2.3.3, which was already implemented in early TCP-like transport protocols, TCP Tahoe counts on the three key mechanisms already explained: *fast retransmit*, *slow start*, and *congestion avoidance*. Thus, Tahoe works exactly as explained in section 2.4.1 and illustrated in Fig. 2.6. Although Tahoe solved the congestion collapse problem mentioned above, it rapidly proved to be too conservative by always reseting its $cwnd$ to one upon a lost packet.

2.5.2 TCP Reno

TCP Reno [Jac90, FF96, Ste97] conserved the three essential mechanisms of the basic TCP Tahoe [Jac88], namely slow start, congestion avoidance and fast retransmit. As explained in section 2.4.2, the novelty introduced into TCP Reno is the fast recovery mechanism. This new mechanism allows for better recover strategy after a lost packet is retransmitted by the fast retransmit mechanism. Specifically, the fast recovery mechanism prevents the communication channel from going empty during the interval the sender is waiting for the ACK of the retransmitted packet. This procedure makes it possible for a single packet loss to be recovered without invoking the slow start mechanism, thereby avoiding unnecessarily abrupt slowdown in the ongoing transmission rate.

Fast recovery is generally invoked when a TCP sender receives three duplicate ACKs, just after the fast retransmit mechanism. By receiving three duplicate ACKs, the sender retransmits the packet that seems to have been dropped and reduces its congestion window ($cwnd$) by one half. Unlike TCP Tahoe, TCP Reno does not invoke slow start, but uses the additional incoming duplicate ACKs to clock out subsequent data packets.

During fast recovery, the $usable$ window of TCP Reno is defined as min($rwin$, $cwnd + ndup$), where $rwin$ refers to the receiver's advertised window and $ndup$ tracks the number of duplicate ACKs. By using the $ndup$ variable, the sender may estimate the amount of packets in flight. After receiving about half a window of duplicate ACKs, the sender may transmit new data packets since the received duplicate ACKs indicate that the receiver has received and acknowledged the involved data packets, and so the channel is somewhat good. Upon receipt of an ACK for a new data packet, which is called a "recovery ACK", the sender exits fast recovery setting $ndup$ to zero.

TCP Reno provides efficient loss recovery in conditions in which a single packet is dropped from a window of data. In such cases, the TCP sender can retransmit at most one dropped packet per Round-trip Time (RTT). TCP Reno is more efficient than its predecessor (Tahoe) but does not work so well when more than one packet is dropped from a window of data. The problem is that TCP Reno may reduce the $cwnd$ multiple times for recovering the lost packets, leading the connection to experience poor performance.

Fig. 2.7 illustrates an example showing how the TCP Reno algorithm works. In this scenario, the packets with sequence number 25 and 28 are intentionally dropped. The mechanism works as follows:

1. The first 24 packets are transmitted and acknowledged properly.
2. Packets 25 and 28 are dropped.
3. The duplicate acknowledgments generated by packets 26, 27 and 29 for packet 24 trigger the fast retransmit/fast recover mechanisms at the sender. These duplicate ACKs cause $ndup$, initially set to zero, to be increased by three.
4. The sender sets the $ssthresh$ and $cwnd$ to one half the current $cwnd$ (fast recover), in this case ten packets, and retransmits packet 25 (fast retransmit).
5. The $usable$ window is set to $cwnd + ndup = 5 + 3 = 8$, i.e., the $usable$ window is “inflated” by three.
6. At this point, the sender is not allowed to send any new packet as its $usable$ window is less than the amount of outstanding packets, which corresponds to twelve packets (packets 25-36).
7. By receiving the next four duplicate ACKs for packet 24, generated by packets 29-32, $ndup$ is incremented by four and so the $usable$ window reaches the size of twelve. The next three duplicate ACKs for packet 24, generated by packets 33-35, make the $usable$ window greater than twelve, allowing the three new packets to be sent (packets 37-39).
8. A new ACK for packet 27, generated by packet 36, is received, taking the sender out of the fast recovery. Note that the reception of packet 25 triggered the transmission of the ACK for packet 27 because packets 26 and 27 were already in the receiver’s buffer.
9. The usable window is “deflated” by having $ndup$ reset to zero, and the $cwnd$ reassumes the control of the sender’s effective window. So the sender cannot send any new packet.
10. Upon receipt of the next four ACKs for packet 27, generate by packets 36-39, the sender is in congestion avoidance phase. Its $cwnd$ is slightly increased but it is not large enough for allowing any new transmission.
11. When the third duplicate ACK for packet 27 arrives, the fast retransmit/fast recovery mechanism are again invoked, as above.
12. Packet 25 is retransmitted and the $cwnd$ is halved.
13. The receiver acknowledges all packets that were outstanding by sending an ACK for packet 39. This resets $ndup$ to zero and the fast recovery is finished.

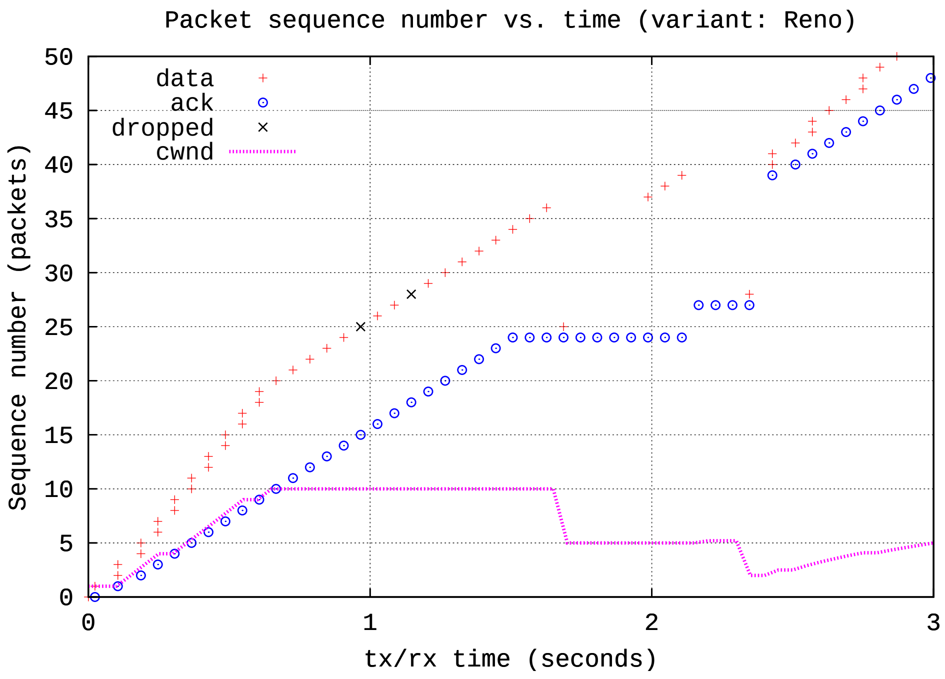


Figure 2.7: TCP Reno reaction to two dropped packets

14. After then, the $cwnd$ will grow up to its specified limit (10 packets) and the equilibrium will be reached.

Fig. 2.7 illustrates how TCP Reno works efficiently for conditions in which a single packet is dropped from a window of data. It avoids abrupt slowdown in the $cwnd$ by implementing the fast retransmit/fast recovery mechanism. This improves performance over a Tahoe implementation which would invoke slow start in such cases. However, if multiple packets are dropped from a window of data, then Reno may suffer performance degradation by reducing its $cwnd$ in sequence. In Fig. 2.7, the $cwnd$ was reduced twice causing the connection to experience performance degradation.

2.5.3 TCP NewReno

NewReno [FF96, APS99, FHAG04] improves the Reno implementation with regard to the fast recovery mechanism. The objective of TCP NewReno is to prevent a TCP sender from reducing its congestion window multiple times in case several packets are dropped from a single window of data. NewReno can also avoid retransmission by timeout in scenarios where the involved congestion window is small preventing enough ACK packets from reaching the sender.

In TCP Reno, when the sender receives a $partial$ ACK packet it exits fast recovery. The term $partial$ ACKs refers to ACK packets that acknowledges some but not all of the data packets that were outstanding when the fast recovery was started. Upon receipt of a

$partial$ ACK, the Reno sender brings the $usable$ window back to the congestion window size, and so exits fast recovery. If there are sufficient outstanding packets, the sender may receive enough duplicate ACKs to retransmit the next lost packet (or packets) until all dropped packets are retransmitted by the fast recovery mechanism. At every invocation of the fast recovery, $cwnd$ is halved. If there are not enough packets outstanding due to a low window size, then the sender needs to wait for the expiration of the retransmission timer. In this case the $cwnd$ is reset to one, inducing bandwidth wastage.

Differently from Reno that exits fast recovery by receiving $partial$ ACKs, the NewReno algorithm remains in fast recovery until all of the data outstanding by the time the fast recovery was initiated have been acknowledged. NewReno can retransmit one lost packet per RTT until all the lost packets from a particular window of data have been retransmitted. In this way, TCP NewReno avoids multiple reductions in the $cwnd$ or unnecessary retransmit timeout with slow start invocation, thereby improving the connection's end-to-end performance.

Fig. 2.8 illustrates how the algorithm of TCP NewReno works. In this scenario, the packets with sequence number 25 and 28 are also intentionally dropped. The mechanism works as follows:

1. NewReno works exactly like Reno until the first ACK for packet 27 arrives at the receiver. Thus, the steps 1-7 occur as described above for TCP Reno.

8. The first ACK for packet 27 is a partial ACK since it does not acknowledge all packet outstanding. Hence, packet 28 is retransmitted immediately and the fast recovery is not ended.

9. The $ndup$ is reset to zero and later increased by the number of duplicate ACKs corresponding to the partial ACKs, and the $cwnd$ is kept unchanged.

10. When receiving the next three duplicate ACKs for packet 27, the sender may not send any new packet because its $usable$ window is not large enough. These ACKs bring the $usable$ window to eight ($cwnd$+$ndup$=5+3=8), but there are twelve outstanding packets (i.e., packets 28-39).

11. Upon receipt of the retransmitted packet 28, the sender acknowledges packet 39, because packets 29-39 are already in its buffer.

12. ACK 39 acknowledges all outstanding packets, and so the sender exits fast recovery with a $cwnd$ of five and continues in congestion avoidance. In addition, $ndup$ is reset to zero.

13. After then, the $cwnd$ will grow up to its specified limit (10 packets) and the equilibrium will be reached.

NewReno prevents the $cwnd$ from being dropped multiple times when more than one packet is dropped from a window of data, as shown above. Nevertheless, like Reno, it is able to recover only one packet per round-trip time.

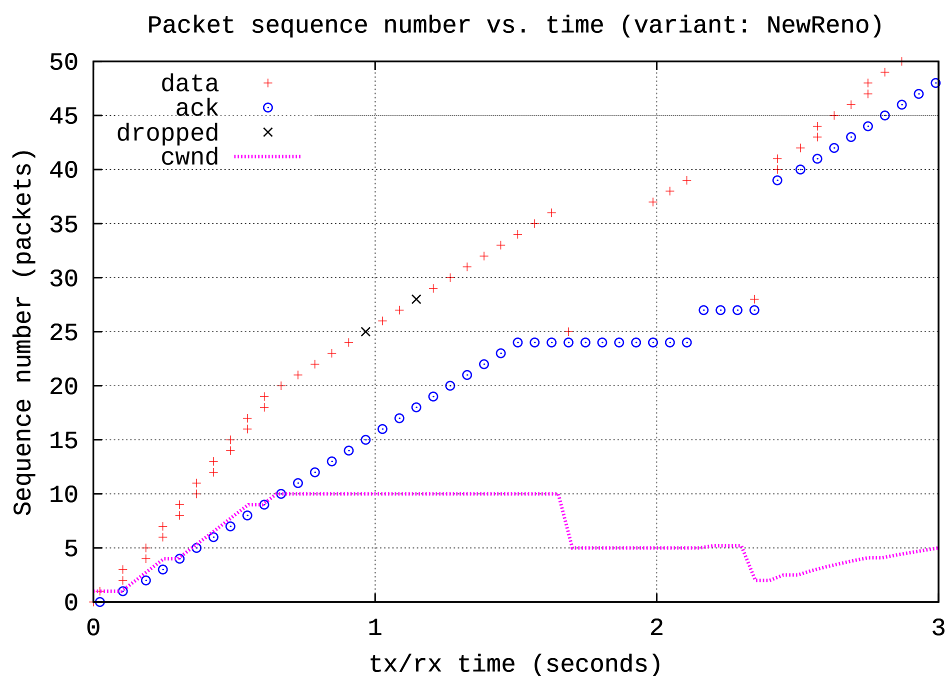


Figure 2.8: TCP NewReno reaction to two dropped packets

2.5.4 TCP Sack

TCP Sack (Selective Acknowledgment) [FF96, MMR96] preserves the basic principles of tcp Reno, namely the robustness in dealing with out-of-order packets and the retransmit timeout as the last resort of lost recovery. In fact, Sack uses of the same algorithms of Reno for increasing and decreasing its congestion window.

The novelty in TCP Sack lies in its behavior when multiple packets are dropped from one window of data [FF96], similarly to TCP NewReno. In Sack, the receiver uses the option fields of TCP header (Sack option) for notifying the sender of up to usually three blocks of non-contiguous set of data received and queued by the receiver. The first block reports the most recent packet received at the receiver, and the next blocks repeat the most recently reported Sack blocks. The sender keeps a data structure called $scoreboard$ to keep track of the Sack options (blocks) received so far. In this way, the sender can infer whether there are missing packets at the receiver. If so, and its congestion window permits, the sender retransmits the next packet from its list of missing packets. In case there are no such packets at the receiver and the congestion window allows, the sender simply transmits a new packet.

Like TCP Reno, the Sack implementation also enters fast recovery upon receipt of generally three duplicate acknowledgments. Then, its sender retransmits a packet and halves the congestion window. During fast recovery, Sack monitors the estimated number of packets outstanding in the path (transmitted but not yet acknowledged) by maintaining a variable called $pipe$. This variable determines if the sender may send a new packet or retransmit an old one, in that the sender may only transmit if $pipe$ is smaller than the congestion window. At every transmission or retransmission, $pipe$ is incremented by one, and it is decremented by one when the sender receives a duplicate ACK packet containing a Sack option informing it that a new data packet has been received by the receiver.

The fast recovery is over when the sender receives an ACK acknowledging all data that were outstanding when fast recovery was entered. If the sender receives a partial ACK, i.e., an ACK that acknowledges some but not all outstanding data, it does not exit fast recovery. For partial ACKs, the sender reduces $pipe$ by two packets instead of one, which guarantees that a Sack sender never recovers more slowly than it would do if a slow start had been invoked.

If it happens that a retransmitted packet is dropped, the Sack implementation reacts exactly as the Reno implementation. In such cases, the sender times out, retransmits and enters slow start. Fig. 2.9 illustrates how the algorithm of TCP Sack works. Like before, the packets with sequence number 25 and 28 were intentionally dropped. The mechanism works as follows:

1. The first 24 packets are transmitted and acknowledged properly.
2. Packets 25 and 28 are dropped.
3. The duplicate acknowledgments generated by packets 26, 27 and 29 for packet 24 trigger the fast retransmit/fast recovery at the sender.
4. The sender sets the $ssthresh$ and $cwnd$ to one half the current $cwnd$ (fast recovery), in this case ten packets, and $pipe$ is set to seven ($pipe = cwnd - ndup = 10 - 3 = 7$).
5. The sender retransmits packet 25 (fast retransmit), and increments $pipe$ by one ($pipe = 8$).
6. The next four duplicate ACKs for packet 24 lead $pipe$ to be reduced by four.
7. When the fifth duplicate ACK for packet 24 is received, the sender is allowed to retransmit a new packet since $pipe = 4$ is less than the $cwnd = 5$. $pipe$ continues set to four because one ACK was received but one packet was sent. The same occurs with the last two duplicate ACKs for packet 24.
8. The ACK for packet 28 arrives. As this is a partial ACK, $pipe$ is decreased by two. Thus, the sender may send two new packets (packets 39, 40), and $pipe$ remains set to four.
9. Sender gets the ACK for packet 31, which is not a partial ACK given that it acknowledges all packets that were outstanding when fast recovery started. This ACK drives the sender out of the fast recovery.
10. After then, the $cwnd$ will grow up to its specified limit (10 packets) and the equilibrium will be reached.

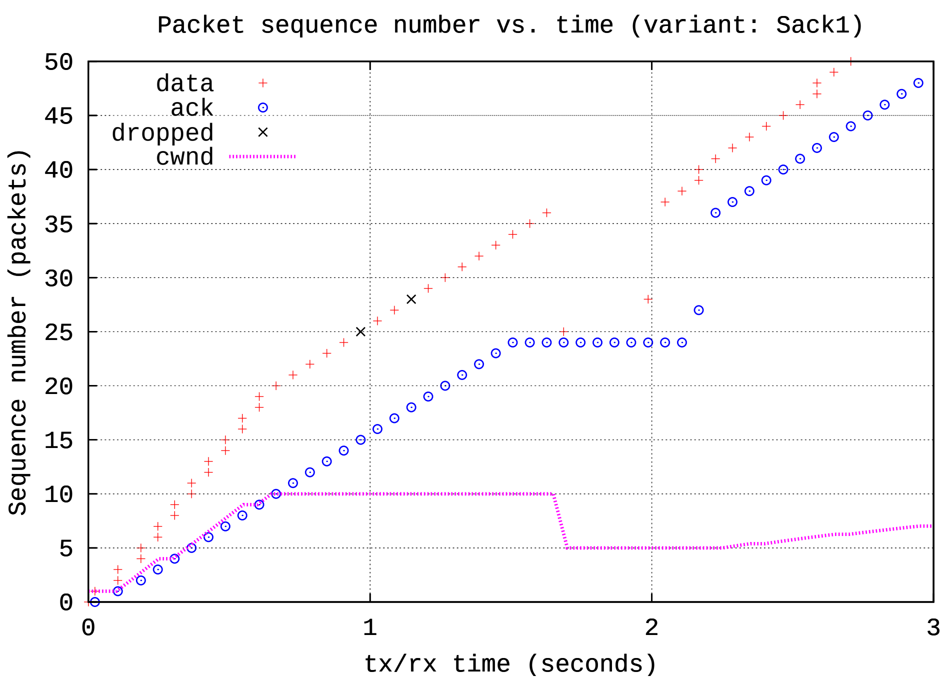


Figure 2.9: TCP Sack reaction to two dropped packets

Sack incorporates all the advantages found in NewReno and may recover multiple lost packets in a window of data in just one single RTT. A Sack implementation requires changes at both sender and receiver, though.

2.5.5 TCP Vegas

Differently from the four TCP versions above, TCP Vegas [BMP94, HBG00] is not an ACK-clocked congestion control. That is, TCP vegas does not need increase its congestion as a function of the number of ACKs received. Yet, while the previous TCP variants detect network congestion by lost packets, TCP Vegas does so by monitoring the changes in the RTTs associated to the packets that it has sent previously through the connection.

If the observed RTTs increase, the Vegas sender infers incipient network congestion and so it reduces the congestion window $cwnd$ by one. Otherwise, if the observed RTTs decrease, the sender interprets that as an indication that the network is free of congestion, and so it rises the $cwnd$ by one. There is a RTT range in which the $cwnd$ remains unchanged. The extension of this range is determined by two parameters: α and β. The dynamics of the $cwnd$ in TCP Vegas is illustrated in (2.4) [HMM99a]

In (2.4), rtt means the measured RTT, $base\_rtt$ is the smallest value of observed RTTs so far and α and β are the minimum and maximum thresholds respectively, for the permitted range on RTT variation without changes in $cwnd$.

Provided that the monitored RTTs (divided by $base\_rtt$) are between α and β, TCP

Vegas infers that its sending rate is matching the network capacity. Under such circumstances, $cwnd$ is kept unchanged in order to prevent losses inside the network. The key idea here is to use the actually available network bandwidth without causing excessive traffic within the network.

$$cwnd = \begin{cases} cwnd + 1, & \text{if} \quad Diff < \frac{\alpha}{base\_rtt}; \\ cwnd - 1, & \text{if} \quad Diff > \frac{\beta}{base\_rtt}; \\ Unchanged, & \text{if} \quad \frac{\alpha}{base\_rtt} < Diff < \frac{\beta}{base\_rtt}; \end{cases} \tag{2.4}$$

With,

$$Diff = \frac{cwnd}{base\_rtt} - \frac{cwnd}{rtt}$$

TCP Vegas was first introduced in the early 1990s, but no significant work on it toward commercial implementations has been carried out. It seems to be a robust protocol as it optimizes bandwidth utilization without incurring in any cost to the network concerning overhead. The main problem to be overcome by the Vegas algorithm has to do with the unfairness that arises when multiple connections are sharing the same communication channel [HMM99b, HMM99a, OG01]. The investigation in [HMM99b] concluded that TCP Vegas can be quite unfair when the competing connections face different RTTs. This occurs because of the Vegas high dependence on RTT measurements as confirmed in [OG01].

2.6 TCP Extensions

2.6.1 Delayed Acknowledgments (DA)

As stated in RFC 813 [Cla82], the acknowledgment mechanism is at the heart of TCP. When data arrives at the receiver, the protocol requires that the receiver sends back an acknowledgment of that for reliability reasons. The data packets are sequentially numbered so the receiver can acknowledge data by sending to the sender the sequence number of the highest data packet it has in its buffer. The acknowledgment scheme is cumulative, which means that by receiving the highest sequence number the sender infer that all prior data were successful received. Thus, a TCP receiver does not necessarily have to transmit an acknowledgment for every incoming data packet.

RFC 813 introduced the basic concepts for optimizing transmission efficiency by reducing the number of acknowledgments generated by a TCP receiver. This RFC shows that reducing the number of ACKs provides two benefits: lower processing overhead at the sender and robustness against the well-known Silly Window Syndrome (SWS). Measurements of TCP implementations, in particular on large operating systems, suggest that most of the overhead involved in a packet handling is not in the TCP or IP layer processing. In fact, the most significant processing occurs in the scheduling of the handler that must deal with the packet at the sender [Cla82].

The Silly Window Syndrome can arise during large data transfers if the receiver does not enlarge enough its *rwin* to allow the sender to send data in large packets. This may happen because of lack of buffer space in the receiver, which leads to high packet fragmentation into small packets, impacting the transfer efficiency seriously. Delaying the transmission of small packets results in the buffers being freed and consequently higher *rwin* being advertised by the receiver. This in turn leads the sender to transmit large packet sizes and so higher throughput is achieved.

To delay ACKs at the receiver is therefore recommended if the network is in good state, since it prevents unnecessary ACKs from being transmitted. However, if the network is facing constraints, additional mechanisms are needed to make sure that the receiver does not lead the sender to miss ACKs. Hence, RFC 813 recommends the use of a timer at the receiver to trigger ACK transmissions for data packets that do not arrive at the receiver in due time. This timer should be reset at every new income data packet and its duration could be either a fixed interval on the basis of the channel characteristics such as typical RTT or be adaptive to the channel conditions.

Although RFC 813 establishes the foundation for the delayed ACK mechanism, it does not specify clearly the actions to be taken by the receiver under a constrained channel. For instance, it does not specify any action to out-of-order data packets or how many packets may be delayed in sequence. The standard Delayed Acknowledgment (DA) strategy was first defined in RFC 1122 [Bra89] and refined in RFC 2581 [APS99]. The former specifies that a TCP receiver should acknowledge every other data packet but should not delay more than 500 ms. In addition, RFC 1122 clearly states that delayed ACKs can substantially reduce protocol overhead by diminishing the overall number of packets to be processed. However, delaying ACKs excessively can disturb the Round-Trip Time estimation as well as the packet “clocking” algorithm in the sender. The term packet “clocking” refers to the sender’s dependence on ACKs to transmit new data packets, i.e., every ACKs trigger a new transmission at the sender.

RFC 2581 further specifies the concept of delayed acknowledgments by including responses of the receiver for out-of-order packets. In order to speed up the loss recovery at the sender, a TCP receiver should immediately acknowledge data packets that are either out-of-order or filling in a gap in the receiver’s buffer. Out-of-order packets are most likely the result of dropped data packets and so it is reasonable to acknowledge them promptly in order to accelerate the sender reaction and avoid timeout. Data packets that are filling in a gap in the receiver’s buffer are retransmitted packets for a missing data at the receiver. These data packets must also be retransmitted immediately to mitigate disturbances for the sender. The standard DA proposed in RFC 2581 is appropriate to improve performance in multihop wireless networks as it is shown in chapter 7.

2.6.2 Explicit Congestion Notification (ECN)

The ECN scheme specified in RFC 3168 [RFB01] proposes to use network feedback to assist a TCP connection in reacting to congestion effects. By using this mechanism, TCP does not need to await a dropped packet due to buffer overflow to detect congestion and properly slow down. Rather, it is informed by the intermediate nodes (routers) when incipient congestion starts. ECN can prevent time wastage at the sender that, without

ECN, always has to wait for either three duplicate acknowledgments or a timeout timer expiration.

The implementation of ECN requires specific flags in both IP and TCP headers. Two bits are used in each header for proper signaling among sender, routers and receiver, as depicted in Fig. 2.10. The active queue management (AQM) [BCC$^+$98, RFB01] inside the routers marks packets when congestion reaches a given threshold. The receiver simply echos back the congestion indication into the ACKs to the sender which reduces its sending rate to prevent severe congestion.

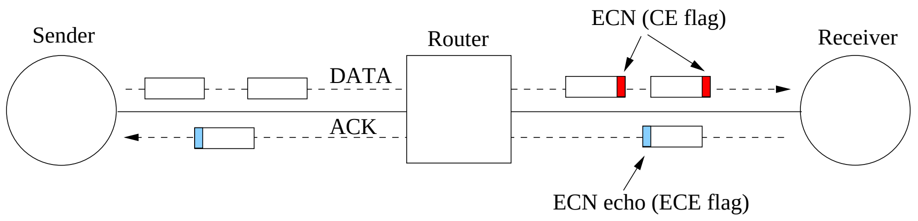


Figure 2.10: ECN bit

The new flags guarantee a smooth interaction among the involved network entities concerning ECN use. This is needed to ensure that non-complaint devices may coexist with ECN-capable devices efficiently. The two bits, ECT and CE, within IP header define four flags, as illustrated in table .

Table 2.1: ECN bits within IP header

ECT	CE	Meaning
0	0	Not-ECT
0	1	ECT(1)
1	0	ECT(0)
1	1	CE

The end nodes and routers set these flags as follows. If no device changes any of these bits, then the flag not-ECT is always active indicating that the corresponding packets are generated by a data sender that is not ECN compatible . The flags ECT(0) and ECT(1) are equivalent and set by the data sender to indicate that the end nodes are ECN-capable. The CE flag is set by a router to inform the end nodes that the network is facing congestion.

The TCP flags are limited to the end-to-end signaling between the end nodes. The two last bits in the reserved field of the TCP header are defined as follows [RFB01].

- ECN-Echo (ECE).
- Congestion Window Reduced (CWR).

The ECN-Echo flag is used by the data receiver to inform the data sender that a CE packet (packet with the CE flag set) has been received. Similarly, the data sender uses the CWR flag to announce to the data receiver that its congestion window has been reduced.

In the TCP connection setup phase, the source and destination TCPs exchange data to define their willingness to use ECN. The ECT flag in the IP header is used by a TCP sender to indicate to the network that it is capable of processing ECN tasks for the packet being transmitted. If the ECT flag is turned off, the routers infer that the sender is not ECN-capable. This allows for coexistence of devices that are ECN-capable and not ECN-capable.

ECN is appealing to be used in the Internet since it does not render any overhead regarding the current IP flows. Its drawback lies in the fact that to be effective, it requires changes to every network element . Additionally, the complexity involved when IPSec tunnels are in place appears to be a great challenge to be overcome. This happens because the intermediate nodes (routers) running IPSec are prevented from accessing the TCP header.

2.6.3 Limited Transmit

RFC 3042 [ABF01] specifies the "Limited Transmit" as an enhancement for TCP loss recovery when a connection's congestion window is small, or when a large number of packets are lost in a single transmission window. Without limited transmit, when a packet is dropped the sender starts receiving duplicate ACKs from the receiver and only retransmits the lost packet when it gets the third duplicate ACK. The sender does not transmit any data packet when it receives the first and the second duplicate ACKs. The problem is that if the receiver works with small $cwnd$, it might happen that it will not receive sufficient (three) duplicate ACKs since there are just a few data packets in transit in such cases. If that happens, the sender can only react to the dropped packet when its retransmission timer expires.

Using the limited transmit, a TCP sender should send a new data packet for each of the first two duplicate acknowledgments received at the sender. Provided that these two new data packets get at the receiver, they trigger two extra ACKs to be received by the sender. This procedure aims at increasing the probability that TCP can recover from a single lost packet using the fast retransmit/fast recovery algorithms instead of using a costly retransmit timeout.

Specifically, the limited transmit algorithm imposes that a TCP sender only sends new data packets in response to incoming duplicate ACKs if the following conditions are met:

- The receiver's advertised window allows the transmission of the new packet.
- The amount of outstanding data would remain less than or equal to the $cwnd$ plus 2 packets, i.e., the sender can only send two packets beyond the $cwnd$ size.

Furthermore, the algorithm specifies that the $cwnd$ must not be changed when these new data packets are transmitted. This increases the probability that the sender infers loss using the standard fast retransmit threshold of three duplicate ACKs. RFC 3042 reports that limited transmit could have reduced up to 25% of the RTO-based retransmissions in measurements on a busy web sever performed in [BPS$^+$97]. Limited transmit is implemented as a default mechanism in the ns2 simulator.

2.7 Summary

This chapter described the main features of TCP that is a fundamental protocol widely deployment in today's Internet. TCP provides reliable end-to-end data transmission and a robust congestion control for easing network overload. Although TCP is a connection-oriented protocol, it does not need any support from the intermediate nodes to set up its connection. In fact, TCP relies on IP routing to transport its data and acknowledgment packets. IP does not guarantee data delivery but TCP does by requiring acknowledgments from the data receiver and carrying out retransmissions.

TCP implements both flow control and congestion control. The former prevents the TCP receiver's buffer from being overflowed while the second avoids congestion collapse within the network. The congestion control continuously probes the network for resources. Modern TCP implementations use the additive increase multiplicative decrease algorithm (AIMD), which aims to speed up loss recovery and ensure network stability. An alternative algorithm to AIMD is the Equation-based congestion control. The latter attempts to reproduce TCP behavior analytically, but an accurate modeling is too hard to achieve. This is a subject of great interest in the research community nowadays.

TCP has evolved over the years, and so various variants of the basic TCP Tahoe mechanism have been developed. TCP Reno adds the fast recovery mechanism to the basic Tahoe. This mechanism avoids the congestion window from resuming from the minimum size of one packet after a dropped packet detection. It also allows packets to be sent while the sender waits for the retransmitted packet, thereby improving throughput. TCP NewReno improves the Reno performance when multiple packets are dropped from a window of data, and so does TCP Sack. TCP NewReno can recover only a packet per RTT, while TCP Sack does not have this limitation. Nevertheless, while NewReno refinements are limited to the sender side, Sack requires changes at both end nodes. Finally, TCP Vegas relies on RTT variations only to adjust its sending rate as opposed to the ack-clocked algorithm used in the other TCP variants.

Some important TCP extensions are: Delayed Acknowledgments (DA), Explicit Congestion Notification (ECN) and Limited Transmit. DA proposes to minimize the amount of ACKs by having the receiver acknowledging every other data packet only. ECN proposes to use feedback from the intermediate nodes to assist a TCP sender to detect congestion before a packet is actually dropped. Limited Transmit proposes to permit the sender to send new data packets while it waits for the third duplicate ACK, in order to improve performance in scenarios were the $cwnd$ is small or the amount of drops is high.

Chapter 3

Multihop Wireless Networks

3.1 Introduction

Multihop wireless networks are emerging as a promising framework for future wireless communications in a broad sense. These networks are foreseen as an essential extension of today's wired Internet and substantial investments are being applied to leverage the development of such networks. Scenarios for multihop wireless networks range from short-range wireless links, common in home building communications for example, to complex mesh of wireless links to connect moving users at changing speeds. These communications arise many difficulties to the involved protocols, which has been the focus of considerable researches over the past few years. This chapter introduces multihop wireless networks and discusses important issues concerning the design of such networks.

3.2 Wireless Data Communications

Wireless networks have been pursued since the beginning of the computer networks history. In the early 1970s, the first experimental radio packet network, called ALOHANET, was set up at the University of Hawaii [Abr85, JT87]. In the mid-1980s, TCP/IP-based wireless networks were introduced, and ever since these networks have been increasingly evolving. Wireless technology is being deployed in many communication systems to either replace or extend wired-based infrastructures. Advantages of wireless networks over wired networks include mobility, simpler setup, easier maintenance, etc.

Wireless networks are, however, much more susceptible to performance degradations than their wired predecessor. As specified in [IEE99], wireless networks have fundamental characteristics that make them significantly different from traditional wired networks. The physical media used in wireless networks pose unique challenges to the design of these networks due to the following features.

- The wireless physical medium used does not have well defined boundaries for specifying precise communication ranges.
- The medium is unprotected from outside signals.

- The communication takes place over a medium significantly less reliable than a typical wired medium.
- The network topology may change dynamically due to either propagation constraints or nodes mobility.
- These networks lack full connectivity, since some nodes may be temporarily hidden from each other.
- Wireless networks also have time-varying and asymmetric propagations properties.

Today's wireless networks can handle these challenges relatively smoothly for scenarios having a singe wireless link. This is enough for two network devices to communicate directly in very short networks that span a few meters. On the other hand, sustainable communications over reasonable distances is still a challenge. Under such distances, there is no feasible technology today that can offer deployable end-to-end connectivity over a single wireless link.

To achieve end-to-end wireless communication extending to distances of several single link capacity, **multihop wireless networks** come into play. The term multihop means that the end-to-end connection may span multiple wireless links (also called hops). The key idea of such networks is to have the intermediate nodes, between sender and receiver, relaying data to the end nodes.

In recent years, multihop wireless networks have been emerging as a viable solution for many applications in **ad hoc networks** and **sensor networks**. There are substantial ongoing researches addressing the challenges facing these networks, which indeed suggests a promising future for this new wireless technology. This chapter focuses on ad hoc networks since these are the primary target of this thesis. A very brief introduction on sensor networks is given for completeness only.

3.3 Ad Hoc Networks

Ad hoc networks [CM99, MPC01, Haa02] are self-organizing wireless networks as they do not require any fixed infrastructure to communicate. These networks are appropriate for scenarios in which wired networks are not possible or not desirable such as disaster recovery, battlefield, short-lived networks as in conference spots, etc. The absence of a centralized entity for controlling the communication among the nodes poses crucial requirements on the protocols in ad hoc networks. Such protocols need to be distributed and to provide smooth coordination among the communicating nodes. As the communication range of the nodes is usually limited, and the communication channels are spatially reused, mutual interference among concurrent transmissions is hard to avoid.

The nodes in ad hoc network work as both hosts and routers, and so each node is able to forward data for its neighbors. This model design is needed because each node counts on a limited transmission range to reach its intended destination node. Hence, when a given node sends data to another node that is not in its transmission range, one of its neighbors forwards the data toward the destination. This process may involve multiple intermediate nodes, thereby establishing a multihop connection as depicted in Fig. 3.1.

Specific routing protocols coordinate the route discovery and maintenance for connecting senders and receivers of the flows to be transmitted. The route maintenance in ad hoc networks is a difficulty task, because the wireless links taking part in the end-to-end route can be interrupted by either wireless medium induced losses or lack of connectivity due to nodes mobility. As a result, various routing protocol proposals for ad hoc networks have emerged over the last few years [PBRD03, JMH04, PB94, PC97, HPS02, CJ03, OTL04]. AODV and DSR are two fundamental routing protocols discussed in detail below.

Another important aspect of ad hoc networks is the medium access control. Since the medium is assumed to be highly error-prone and the nodes in place may move unpredictably, the medium access control protocol must be not only highly adaptive but also tolerant to transmission failures. The IEEE 802.11 [IEE99] standard was designed to meet some of such requirements, but many problems remain to be addressed, as shown next.

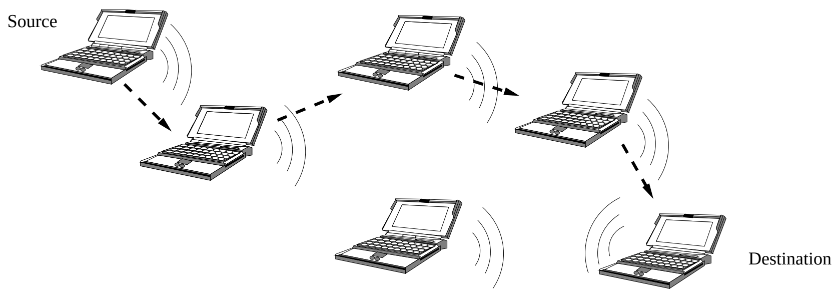


Figure 3.1: Ad hoc networks

3.3.1 IEEE 802.11 MAC Protocol

The IEEE 802.11 “Distributed Foundation Wireless Medium Access Control” (DFWMAC) [CWKS97, IEE99, CGL00] is the standard Medium Access Control (MAC) layer protocol adopted for ad hoc networks. The IEEE 802.11 standard specifies both the wireless LAN MAC and physical layer mechanisms for an efficient shared broadcast channel through which the involved mobile nodes can communicate.

The communicating nodes within a wireless LAN (WLAN) are termed stations, and the IEEE 802.11 standard is mostly termed simply 802.11 or 802.11 MAC protocol. The unit of information used for the MAC messages is frame. These nomenclature are used throughout this section for compliance with the standard description.

In 801.11, priority may be given to stations but in general all stations receive equal right to access the medium. Collisions are prevented instead of detected after they happen, and multiple hops communication is allowed. Moreover, power management functionalities are also included in the standard. The main novelties of 802.11 include: 1) use of acknowledgment for data frames (link layer’s ACKs), 2) possibility of using RTS/CTS (request-to-send/clear-to-send) control frames, 3) a virtual carrier sensing mechanism. These mechanisms aim at mitigating medium collisions and obtaining efficient bandwidth utilization, as explained below.

MAC Frame Formats

Before addressing the MAC mechanisms in detail, it is important to understand what is carried inside each MAC frame exchanged among the WLAN stations. There exist three types of frames: control, data and management. Control frames are used for tasks such as medium reservation, power saving, etc. Data frames are the ones which actually carry useful data from the user's application perspective. Management frames are needed for authentication, deauthentication, association, and disassociation, in a managed WLAN which is not addressed in this thesis.

Fig. 3.2 illustrates the general MAC frame format. Among the frame categories mentioned above, only some of them contain the fields Address 2, Address 3, Sequence Control, Address 4, and Frame body. The MAC frames consist of the following basic parts:

- A *MAC header* that is composed of frame control, duration, address, and sequence control information.
- A *frame body* containing information specific to the frame type
- A *frame check sequence* (FCS) comprising an IEEE 32-bit cyclic redundancy code (CRC)

The fields in the general MAC frame have the following purposes:

- **Frame Control:** This field contains basic information that are needed in every frame for smooth cooperation among the stations. Its specific subfields are explained below.
- **Duration/ID:** This field has two purposes: to indicate the duration the medium is going to be used for transmitting the frame under consideration; and only for Power-Save Poll messages, to indicate the ID of the station transmitting the frame.
- **Address1,2,3,4:** These fields are used to identify a certain station. These addresses define for example which station is transmitting, receiving, generating the data or being the final receiver of the frame.
- **Sequence Control:** This field carries information needed for transmitting fragmented frames and identifying frame duplicates. It is divided into two subfields: Fragment Number and Sequence Number. The latter is assigned to every frame transmitted by a station and the former identifies the fragmented part that is being sent. Both fields are kept unchanged under retransmissions.

- **Frame Body:** In this variable length field is conveyed the upper layer payload. This field is not used for control and management frames.

- **FCS:** This is the *Frame Check Sequences* field which contains a 32-bit Cyclic Redundancy Check (CRC) for checking the received frames' integrity

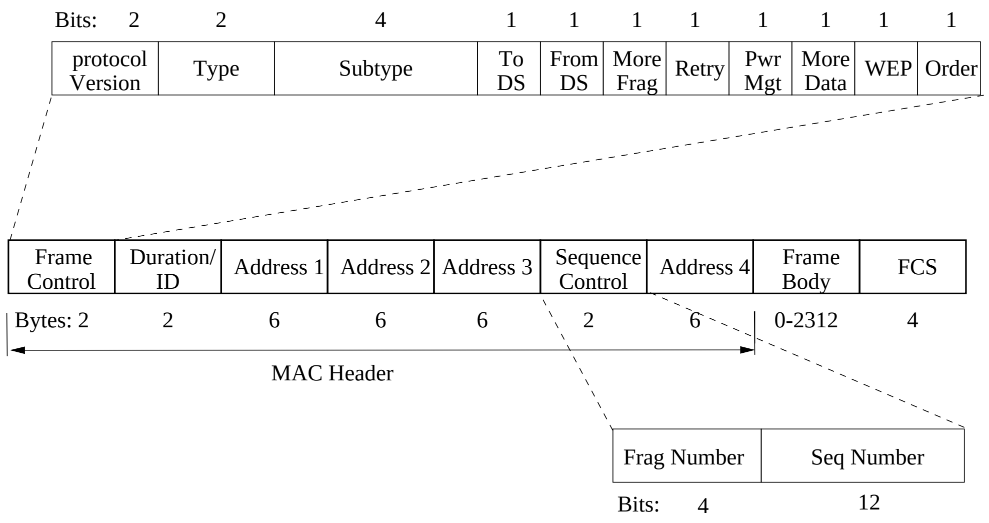


Figure 3.2: MAC frame format

Likewise, the **Frame Control** field is subdivided into eleven subfields as depicted in Fig. 3.2. The role of each of these subfields are as follows:

- **Protocol Version:** This field indicates the protocol version in use. It is currently set to zero, and may be changed to indicate possible future versions.

- **Type and Subtype:** These fields combined define the function of the frame. This is necessary because there are several specific frames within the general classes of control, data, and management frames.

- **To DS** and **From DS:** These fields indicate if a frame is destined to or from a distributed system (DS), respectively. Both fields are relevant to infrastructure networks only, for ad hoc networks they are set to zero.

- **More Frag:** This field is set to one when more segments associated to the current frame is to follow this one. It is set to zero in all other frames.

- **Retry:** This bit is set to one in retransmitted frames, and set to zero otherwise. Using this information, a receiving station is able to eliminate duplicate frames.

- **Pwr Mgt:** Indicates if the transmitting station will be in power-save mode or active mode after the current transmission.

- **More Data:** In infrastructure networks, the Access Point (AP) may buffer frames destined to certain stations toward power saving. This bit is set to one when the AP has data buffered to the receiving station.
- **WEP:** When set to one indicates that the frame is encrypted by the WEP algorithm.
- **Order:** When set to one indicates that the frames and fragments are sent in order. This field is used for some specific protocols that might need such a service.

Distribute vs. Point Coordination Function Access Methods

The IEEE 802.11 standard provides functionalities for both managed WLANs and self-organizing WLANs. The former is generally referred to as an infrastructure network while the latter is termed ad hoc networks. In particular, The IEEE 802.11 standard specifies two MAC access methods: Distributed Coordination Function (DCF) and Point Coordination Function (PCF). DCF specifies functionalities for both ad hoc networks and infrastructure networks. It is a contention-based access method in which every station contents concurrently for the medium. PCF is designed for infrastructure networks and uses the DCF rules as the basic ones for coexistence of DCF and PCF when there is an overlapping area involving an infrastructure network and an ad hoc network. In CFP, an Access Point (AP) is normally the Point Coordinator (PC) which controls the medium access, providing a contention-free access method.

When operating concurrently, PCF has access priority over DCF, and both access methods operate alternatively. In this way, a contention-free (CF) period is followed by a contention-period (CP). CFP is not further explained here as the target environment of this thesis is ad hoc networks where neither PC nor AP are in place. DCF is thus assumed to be the access method used throughout this thesis. The details of DCF are discussed below.

Distributed Coordination Function (DCF)

DCF implements a *carrier sense multiple access with collision avoidance* (CSMA/CA) protocol for controlling the shared medium access among the competing stations. The CSMA/CA distributed algorithm includes a random backoff procedure to minimize the probability of collisions by simultaneous transmission attempts just after the medium has been sensed idle. Actually, the CSMA/CA imposes that a minimal idle interval exist between contiguous frame sequences. Hence, every station must wait for a specific Inter-frame Space (IFS) after the medium is determined idle, and then delay a random backoff interval before transmitting. This reduces collision probability considerably.

In addition to the CSMA/CA protocol, the access method of DCF uses *positive acknowledgments* in that the receiving stations respond to a frame reception with an acknowledgment frame (ACK frame). The transmitting stations are able to retransmit frames that are dropped or have their respective ACK frame dropped. The DCF retransmission mechanism is persistent in the sense that it attempts for several times to recover locally an unsuccessful transmission.

The features explained above render DCF a robust protocol for the challenging wireless networks. It mitigates problems such as the classical hidden node as well as typical performance degradation by the high bit error rate inherent in wireless communications. The DCF mechanisms are addressed below, where the need of each mechanism will become clear.

Carrier Sense Mechanism

Similarly to the traditional Ethernet (802.3) based LANs, WLANs relying on 802.11 DCF access method also implements a *carrier sense multiple access* mechanism for detecting transmission in the shared medium. Nonetheless, while the 802.3 performs congestion detection (CD), 802.11 DCF carries out congestion avoidance (CA). Hence, the carrier sense mechanism used by 802.11 DCF is termed CSMA/CA in contrast to the mechanism used by 802.3 that is called CSMA/CD. The DCF access method of 802.11 cannot implement CD because in a radio network a station is not able to hear the medium while transmitting.

The CSMA/CA is referred to as a *physical carrier sensing* mechanism since it is associated to the instantaneous physical medium condition. In other words, CSMA/CA detects whether there is or not an actual transmission going on when a station intends to transmit by analyzing, for instance, the signal strength of other stations. CSMA/CA is, nonetheless, not the only carrier sensing mechanism included in 802.11. There exist a *virtual carrier sensing* mechanism as well. The latter is concerned with maintaining stations silent for a certain estimated period that should be sufficient for the ongoing transmission to end.

The *virtual carrier sensing* is referred to as the *Network Allocation Vector* (NAV). The principle of NAV is to include information into the frame headers so that a transmitting station may inform the other stations for how long the medium is going to be busy with the current transmission. For that, a field is reserved in the frames header as shown in Fig. 3.2. Whenever a station transmits into the medium, except for the PS-Poll control frame, it sets the **Duration/ID** field to the estimated time (in microseconds) the medium is going to be busy. The other stations overhearing this transmission, set their respective NAVs to this announced time value. In this way, all stations will refrain from transmitting for at least NAV interval.

In fact, 802.11 in DCF takes into consideration both the physical and the virtual carrier sensing mechanisms to conclusively decide whether the medium is busy or idle. If either of the mechanisms perceive a busy medium, then the station must assume that the medium is indeed busy. Otherwise, the medium is assumed to be idle. The frame exchanges involving carrier sensing mechanisms are explained below.

After sensing the medium idle, a station may still await a random backoff time before being allowed to transmit. This is the congestion avoidance (CA) part of CSMA/CA. The CA is necessary to reduce the probability of having various stations transmitting at the same time, which would lead the medium to experience collision. The details of the random backoff procedure are discussed below.

Interframe Spacing

The IEEE 802.11 standard allows for medium access priority levels by specifying different time intervals for accessing the wireless medium after a station perceiving it as idle. These time intervals are called interframe spacing (IFS), and three of them are relevant for the DCF access method.

- **SIFS (Short IFS):** The shortest of all interframe spaces, used for high priority traffic such as ACK frame and clear to send (CTS) control frames.
- **DIFS (DCF IFS):** Longer than SIFS and defines the time duration the medium must be idle before a station may transmit or decrease its backoff timer.
- **EIFS (Extended IFS):** The largest interframe space, used by a station that received a corrupted frame. EIFS prevents this station from colliding with other stations since the unfortunate station could not update its NAV information (received frame was corrupted).

The IFS intervals above promote smooth access to the medium even under multirate-capable physical layers. The interframe space should thus be independent of the station data rate and fixed for a specific physical medium. Different IFS periods may also be used for providing levels of priority for distinct flows. This feature is very appealing for QoS approaches [LAS01].

Backoff Procedure

In order to avoid collisions among concurrent stations when the medium becomes idle, after the mandatory waiting period SIFS, the stations have to further delay their transmission in a random fashion. The backoff time procedure is designed for this purpose. If the transmission attempt fails, then the sending station retransmits the corresponding frame in sequential attempts up to a certain number of times. At every attempt, the range of randomness is enlarged as explained below.

The backoff procedure used by 802.11 computes increasingly random interval ranges as the station continues trying to retransmit the dropped frame. To accomplish that, the backoff interval is computed by (3.1), where i is the number of retransmission attempt, $random()$ is a uniform variable in (0,1) and int(x) represents the non-fractional part of x.

$$\textit{Backoff\_time} = int((2^{i-1} - 1) * random()) * slot\_time \tag{3.1}$$

The equation above shows that the backoff time is an integer value corresponding to the number of time slots. The term $(2^{i+1} - 1)$ is denoted Contention Window (CW) and is limited to a maximum value CWmax. CW represents the range from where the random backoff time is computed. Hence, for successive attempts, CW assumes the following values:7, 15, 31, 63, 127, 255, 255, etc. If CWmax is reached, the subsequent attempts use the same value for CW set to CWmax. It is important to note that by increasing CW the probability of having a larger backoff interval is higher. Nevertheless, the effective value is chosen randomly from $random()$ for keeping statistical independence of the computed numbers among the competing stations.

Once a backoff time has been chosen, the station keeps track of the medium transmissions for updating its backoff timer that it set to the backoff time value just computed. Whenever the medium is idle for DIFS interval, the backoff timer is decremented by one, and when the medium is busy the backoff timer is frozen. This process continues until the backoff timer reaches zero, which then allows the station to transmit. If the transmission is unsuccessful, a new random backoff time within the next larger range is computed. The station decrements its backoff timer like before and attempts transmission when the backoff timer comes to zero. When the transmission or retransmission occur successfully, the CW is reset to its initial value. As a consequence, the next random backoff time will be computed in the range 0-7. To summarize the ideas behind the backoff procedure, the following remarks may be useful:

- **Contention Window (CW):** A counter that specifies the range from which a random backoff time is computed. It increases exponentially ($2^{i+1} - 1$) at every unsuccessful retransmission attempt i, and is reset to its initial value upon successful transmission.

- **Random backoff time:** A random interval computed within a range defined by the CW.

- **Backoff timer:** A timer that is initialized to the random backoff time and decrements under idle medium until zero to allow transmission by the station.

Basic Access Method

As mentioned above, transmissions in 802.11 are accomplished using positive acknowledgment from the receiving stations. For every received data frame, an ACK frame is returned. This increases robustness in error-prone environments by allowing any peer of communicating stations to exchange messages to make sure that the sent frame has been indeed received at the receiving station. Positive acknowledgment provides functionalities necessary for MAC layer retransmissions as well.

In the basic DCF access method, without PCF, a station determines whether it may transmit by monitoring the medium through the CSMA mechanism described above. If the medium is sensed idle for DIFS period, the station may transmit immediately. On the other hand, if the medium is sensed busy, the station has to defer transmission for DIFS period followed by a random backoff period as described above.

Fig. 3.3 illustrates the basic access method. The medium is assumed to be initially idle for DIFS period, and so the sending station is allowed to transmit its data frame. Upon successful reception, the receiving station awaits a SIFS interval and then sends back an acknowledgment frame (ACK frame) to the sending station. As a SIFS period is shorter than any other IFS, the ACK frame is transmitted with highest priority over any other frame. This assures that under normal circumstances, data and corresponding acknowledgment frames are transmitted in subsequent time slots for transmission completeness.

The other stations overhearing the ongoing data transmission defer their transmission until the medium becomes idle for DIFS interval. Every station that overhears the ongoing transmission update their respective virtual carrier sensing mechanism through the NAV field. Recall that the frame header carry a duration field announcing to the other stations the estimated time the medium is going to be busy for the current transmission. The value to update NAV includes the SIFS period as well as the ACK duration, as shown in Fig. 3.3. Once the medium is sensed idle for DIFS interval, the other stations invoke their backoff procedure and transmit when their respective backoff timer reach zero.

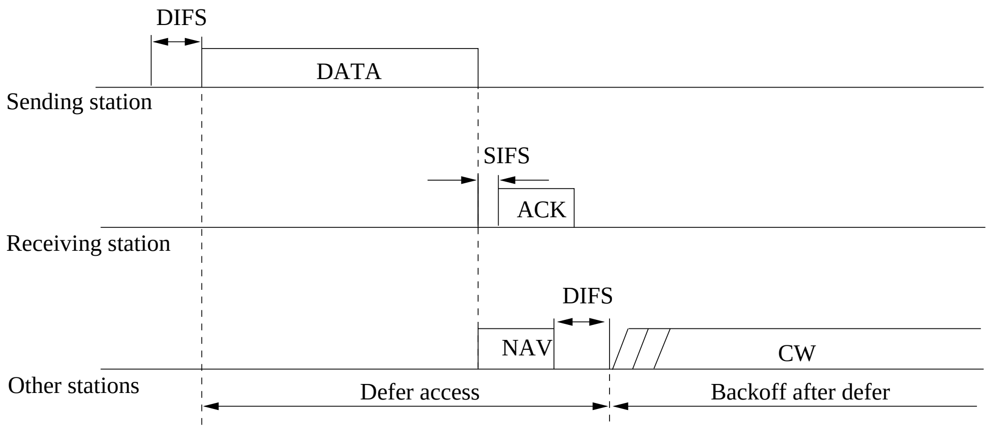


Figure 3.3: Basic access method in 802.11

RTS/CTS Access Method

The basic access method described above may be inefficient for transmission of large frame sizes. Since a station in a wireless network (radio network) may not listen to the medium while transmitting, it has to wait long before detecting that its transmitted frame collided with other stations's transmission. A station has to wait for the estimated time for both the data frame and corresponding ACK frame to be transmitted to take action toward recovering the unsuccessful transmission. Hence, the larger the data frame the longer the time to recover from a failed transmission, which incurs in waste of bandwidth.

To address the waste of bandwidth involved with corrupted frames, the 802.11 in DCF access mode can use short control frames to reserve the wireless medium prior to actual data transmission. There are two specific frames for that: Request-To-Send (RTS) and Clear-To-Send (CTS). These frames are exchanged by every peer stations involved in a transmission before the data frame may be sent. In case of frame corruption, the waste of bandwidth is much less substantial because both RTS (20 bytes) and CTS (14 bytes) sizes are significantly smaller than the maximum date frame size (2346 bytes).

Fig. 3.4 illustrates the RTS/CTS access method. Analogously to the basic access method, the sending station may transmit upon detection of idle medium for DIFS interval. In this case, however, the station transmits first an RTS frame and waits for the CTS frame from the receiving station. For the same purpose mentioned in the explanation of the basic access method, immediately after the reception of the RTS frame the receiving station waits only a SIFS period to transmit the CTS. By receiving this reply, the sending station infers that the medium is reserved for it and so it transmits the data frame and waits for the ACK frame. Note that all the intermediate interframe spaces involved in the transmission of a complete data frame are SIFS. This procedure leads the medium to be reserved for the whole frame exchanges associated to a successful data transmission.

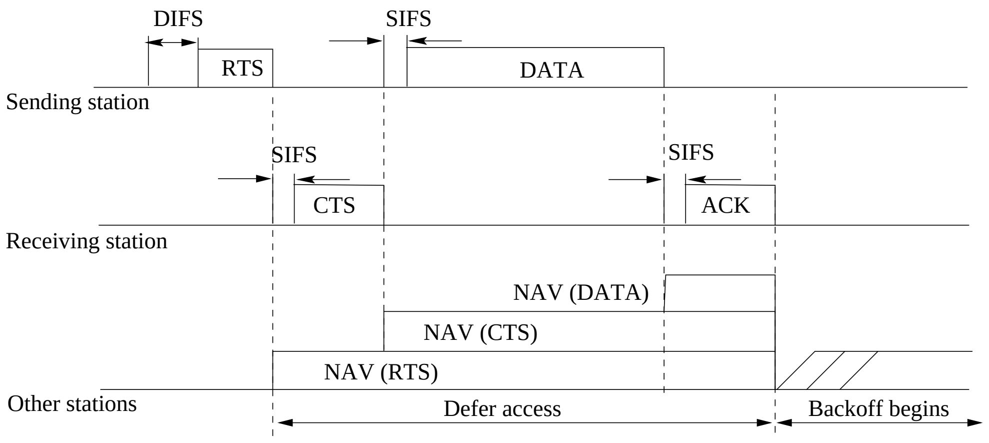


Figure 3.4: IEEE 802.11 Timing diagram

The technique explained above is called *four-way Handshake* because every data frame transmission requires that a sequence of RTS-CTS-DATA-ACK be exchanged between sending and receiving stations. Fig. 3.4 also shows that the other stations update their waiting time by setting their NAV values in accordance with the duration value announced in the header of the frames exchanged between the two communicating stations. Thus, the other stations remain silent until the end of the *four-way Handshake* plus DIFS period. Afterwards, the backoff procedure is invoked and eventually these stations may transmit.

The use of the short RTS/CTS control frames is also appropriate for mitigating the well-known hidden node problem [CGL00]. This problem occurs when two hidden nodes from each other wish to communicate simultaneously with a third common node, which would result inevitably in collision. As shown in [WPL$^+$02, GK04], RTS/CTS prevents neighboring nodes that can hear either the sending station or the receiving station from transmitting. As illustrated in Fig. 3.5, the RTS frame silences the stations reached by the sending station's transmission, while the CTS frame does so for the stations hearing the receiving station's transmission.

The main drawback of using RTS/CTS is that the overhead associated to them renders these control frames inefficient in transmission with relatively small packet sizes. Furthermore, the RTS/CTS control frames do not fully solve the hidden node problem for scenarios where long chain of nodes are in place as not every node can hear each other's

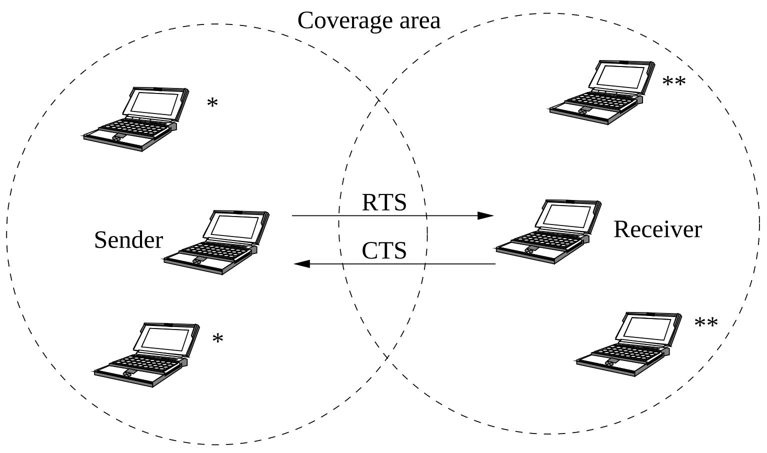

Coverage area
Sender
Receiver
RTS
CTS
*
*
**
**
* Silenced by sender transmission of RTS
** Silenced by receiver transmission of CTS

Figure 3.5: CTS/RTS to prevent hidden node problem

transmission. This problem is discussed in detail below, and section 5.3.1 addresses it further from a TCP perspective.

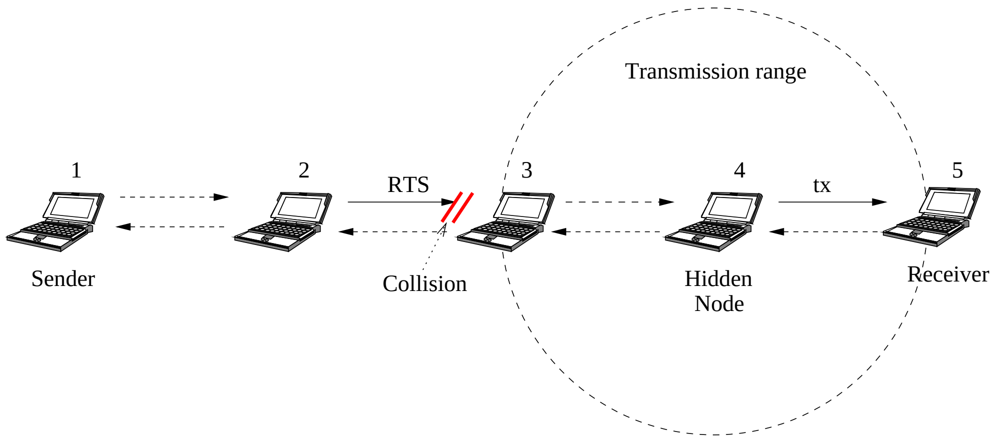
Hidden Node and Exposed Node Problems in multihop networks

Fig. 3.6 illustrates a chain of nodes topology (also called a string topology). In this scenario, each node can only communicate with its adjacent neighbors and only a single connection exists. Suppose node 1 starts transmitting to node 5, as shown in Fig. 3.6(a). After a while, there might be a condition in which node 2 wishes to communicate with node 3 whereas node 4 is transmitting to node 5. As node 2 cannot hear node 4 transmission, it senses the medium idle (both physically and virtually from NAV) and so attempts its transmission by sending an RTS toward node 3.

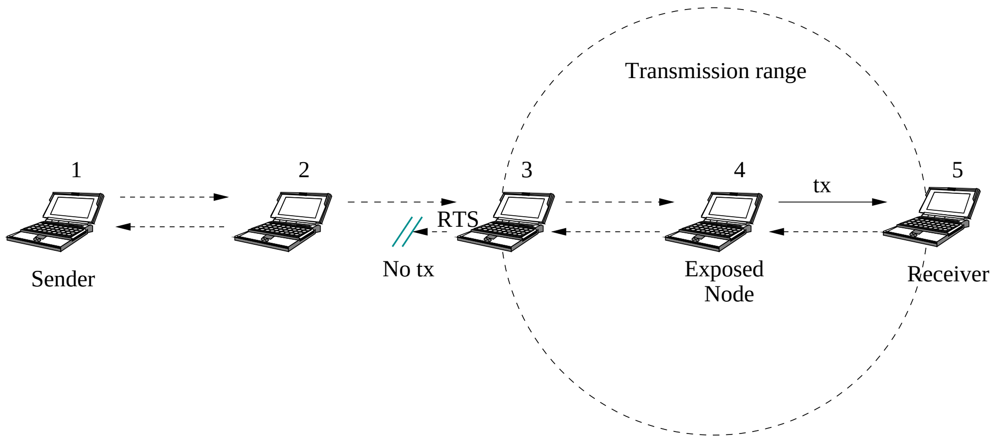
Nevertheless, since node 3 is within the interference range of node 3 (i.e., the transmission of node 4 affects node 3 reception) it cannot receive any data because the data are are dropped by collisions. This is a typical hidden node problem, where node 4 is the hidden node (in relation to node 2).

Figure 3.6(b) depicts a particular condition of the exposed node problem, where node 3 has a data frame (related to a TCP ACK) to send to node 2. As node 4, which is within the sensing range of node 3 (i.e., node 4 transmission affects node 3 transmission ability), is transmitting to node 5, node 3 must wait for the end of current transmission and then contend for the medium. In this case, node 4 is the exposed node relative to node 3. Interestingly collisions occurs only at the receiver, and so node 2 could receive the frame from node 3 correctly despite the simultaneous conversation of node 4 with node 5 since at first node 4 does not interfere with node 2.

The explanation above is a very simplified way of describing the hidden node and exposed node problems. The actual propagation model of 802.11 counts on two commu-



(a) Hidden node



(b) Exposed node

Figure 3.6: 802.11 shortcoming in multihop networks (simplified model: only transmission range)

nication ranges: the *transmission range* and the *interference range*. The former defines the range within which a certain node can successfully communicate with all nodes inside the area limited by this range. The interference range is bigger than the transmission range and defines the area within which nodes can hear the sender transmission but cannot establish an effective communication. Yet, the interference range is typically slight larger than twice the transmission range. Section 5.3.3 shows how important these concepts are for optimized performance in multihop networks.

3.3.2 Routing Protocols

Development of routing protocols for ad hoc networks has been one of the hottest topics within this area in recent years. These protocols face tough challenges in these constrained environments where a proper tradeoff between responsiveness and accuracy in finding routes connecting source and destination is needed. Various protocol proposals have emerged recently [PBRD03, JMH04, PB94, PC97, HPS02, CJ03, OTL04]. However, just a few of them appear to be promising approaches for future ad hoc networks.

In particular, two proposals have been evaluated extensively in the literature and are under process of standardization in the Internet community. These proposals are the Ad hoc On-demand Distributed Vector Routing (ADOV) [PR99] and the Dynamic Source Routing (DSR) [JM96] protocol. [CJ03] and [OTL04] are also expected to be considered in future ad hoc networks as they have recently received the status of RFCs. However, we do not address these protocols here since our goal is to provide only an overview on routing protocol principles rather than an complete review on them. AODV and DSR routing protocols are described below as an insightful introduction to this broad subject.

Both AODV and DSR are reactive routing protocols in that they work on an on-demand basis. In this concept, routes are only established when necessary by the source nodes. As a result, minimal state variables are required since these variables are related to the active routes only. Whenever a node first needs a route to a destination it starts a route discovery procedure which eventually finds a route between source and destination provided that such a route exists. The established route may be kept until it is either not valid or not desired any longer.

On-demand Distributed Vector (AODV)

AODV [PR99, PBRD03] extends the principles adopted by the Destination Sequenced Distance Vector (DSDV) [PB94] routing protocol which is a table-driven algorithm based on the classical Bellman-ford routing mechanism. The key idea of AODV is to reduce the elevated number of required broadcasts typical of DSDV for keeping up-to-date routing tables. By updating routes on an on-demand basis, AODV provides a optimized routing strategy for large ad hoc networks. In fact, AODV is denoted *a pure on-demand route acquisition system* because it only involves nodes in a selected path to discover and keep the related routing information. The other nodes are completely unaware of such routes existence.

The routing procedure in AODV involves two phases: route discover and route maintenance. Whenever a node needs to find a route to a destination to which it has no table entry, it begins the route discover algorithm. Once the route has been established, the route maintenance algorithm takes care of the route's state variables until the route is not needed anymore.

A node starts route discovery by broadcasting a route request (RREQ) packet to its neighbors. The receiving neighbors forward the RREQ to their neighbors and so on, until the message reaches either the destination itself or an intermediate node that contains a fresh enough table entry pointing to the destination. Every node under AODV keeps two counters: a node sequence number and a broadcast ID. These counters are incremented whenever a new RREQ is generated. The RREQ message includes five identifiers: *source IP address, source sequence number, broadcast ID, destination IP address, destination sequence number*. The *source IP address* with the *broadcast ID* uniquely identify an RREQ. The *source sequence number* is needed to guarantee loop-free routes and to maintain freshness information about the reverse route to the source, as explained below. The *destination sequence number* is checked by the intermediate nodes to determine whether one of its routes to the destination is fresh enough to be used.

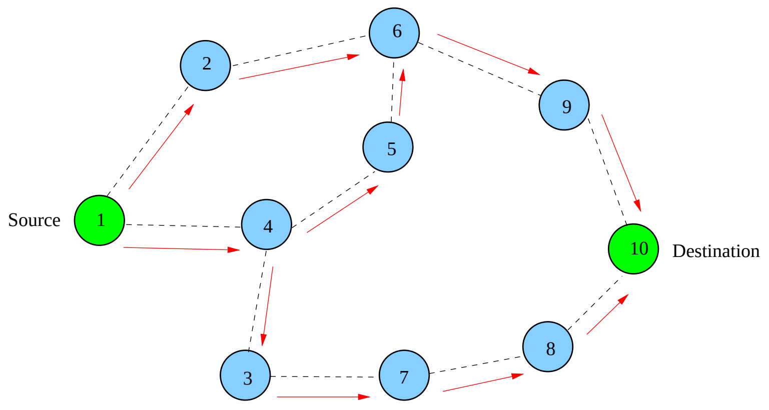
In its way to the destination, the RREQ message may be received by various intermediate nodes. If an intermediate node does not contain any "fresh enough" table entry to the destination, it simply forwards the RREQ. Otherwise, the intermediate node replies to the source node with a request reply packet (RREP). The term "fresh enough" refers to the table entries that both have routes to the destination and whose *destination sequence number* is larger than or equal to the one inside the RREQ. Fig. 3.7(a) illustrates the RREQ propagation across the networks when a broadcast is initiated.

The intermediate nodes prevent route loop by discarding RREQ containing the sequence unique identifier, namely the same *source IP address* and *broadcast ID*. Such redundant RREQs are also not broadcasted for obvious reasons. Moreover, the intermediate nodes establish a reverse path to the source by saving in their respective route tables the address of the nodes from which they first received the RREQ. In this way, either the destination or an intermediate node with a fresh enough route to the destination may unicast an RREP back to the source via the reverse established path. This feature renders AODV a routing protocol that supports symmetric links only, since forward and backward paths are fixed during the route's lifetime. When forwarding the RREP back to the source as shown in Fig. 3.7(b), the intermediate nodes also update those forward route entries in their route tables pointing to the nodes from where the RREP was received.

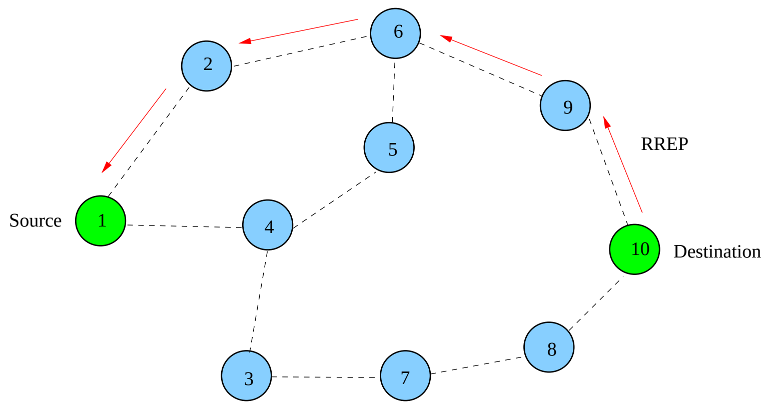
The route maintenance in AODV deals with the effects of node movements and link failures due to the medium. When a source node moves, it can initiate the route discovery procedure to establish a new route to the destination. A route failure may be triggered by either a movement of a node along the route or a link failure. In both cases, the nodes located upstream the point of failure detects the problem and propagates a link failure notice message (an RREP with infinite metric) to each of its upstream neighbors. The process repeats until the link failure notification reaches the source node. A route discovery to that destination may be initiated by the source node if the failed route is still needed. AODV contains a timer associated to each of its route so that the routes not used for a certain period can be purged. *Hello* messages may also be optionally used in AODV to allow every node to inform its neighbors that it is active.

Dynamic Source Routing (DSR)

DSR was designed for small scale networks of up to about 200 nodes [JM96, JMB01, JMH04]. It employs the concept of source routing instead of hop-by-hop routing. This means that every packet carries in its header the complete, ordered list of nodes through which the packet must pass. This feature releases the intermediate nodes from the task of



(a) RREQ broadcasted



(b) RREP sent back to the source

Figure 3.7: AODV route discovery

keeping route tables to forward packets because the packets themselves already contain such information. Thus, DSR does not need any period message exchanges common in many other routing protocols for maintaining accurate route tables. In DSR, every node contains a cache of source routes it has learned or overheard in order to speed up route discovery when a route breaks.

As in AODV, DSR also consists of two phases: route discovery and route maintenance. Nodes initiate route discovery when they need a route that is not found in their respective route caches. Broken routes are detected by proper route maintenance mechanisms.

Analogously to AODV, a source node initiates route discovery by broadcasting a *route request* message. This message contains: source node's IP address, destination node's IP address, and a unique identifier. Every node receiving this *route request* message first verifies whether it has, in its route cache, a route to the destination. If it does, then a *route reply* message is sent back to the source node. Otherwise, the node adds its IP address to the list of nodes through which the packet has already passed and forwards the message. Before forwarding the *route request* message, however, the node first checks if neither its IP address is already in the ordered list nor the message has already been received by the node. In either case, the node does not forward the redundant message in order to avoid network overload.

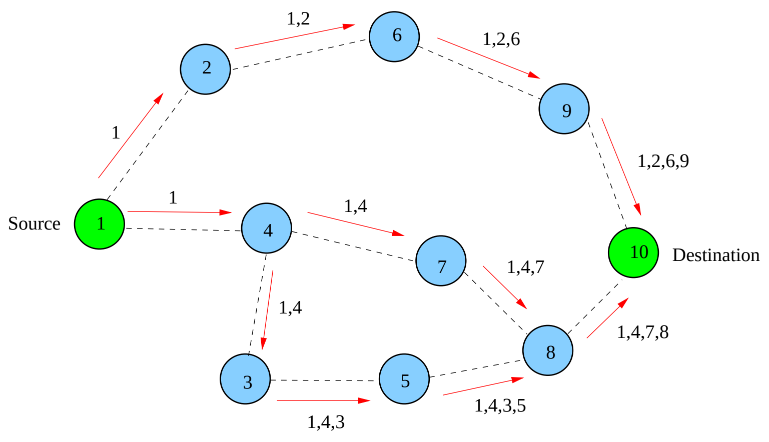
As the packet carrying the *route request* message propagates through the network, it builds the route through which the message has passed as shown in Fig. 3.8(a). The *route reply* message may be generated by either the destination node or an intermediate node containing a valid route to the destination. If the destination node is the generator, it includes the source route received in the *route request* message in the *route reply* message to be transmitted to the source node. In case the *route reply* is generated by an intermediate node, the message is formed by appending the node's cached route to the ordered list just received. Fig. 3.8(b) depicts the route reply transmission carrying the associated sequence of hops defined by the route request. The *route reply* message can be transmitted to the source node either by the reverse route defined by the *route request* or piggyback on a new route request from destination to source. The term piggyback refers to procedure in which a secondary data is transmitted combined with the main data in order to save network resources.

Route maintenance in DSR is performed by means of failure signaling mechanisms and transmission acknowledgments. If a given route is broken, the affected nodes remove the corresponding parts from their cached routes and sent a *route error* message to their neighbors which repeat the process until all impacted routes in cache have been updated. The transmission acknowledgments may be performed at the link layer as it is the case in 802.11 which uses a positive acknowledgment scheme as described above. Alternatively, passive acknowledgments in which the sending node overhears the forwarding of the next hop node may be used. In either way, if no acknowledgment is perceived at the sending node a *route error* message is generated.

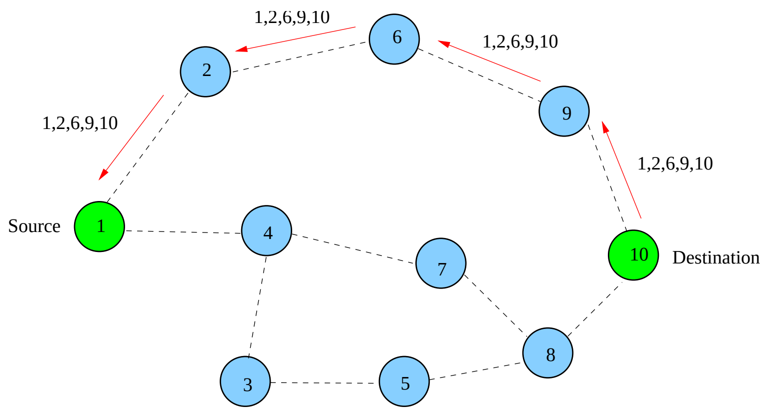
By the descriptions above, it is not difficult to see that there are various similarities between ADOV and DSR. The key advantages of DSR over DSR include the possibility of asymmetric paths and faster route recovery by keeping several source routes to the same destination without any timer defining their lifetime. The disadvantages seem to be the high overhead for carrying the full path inside the packet headers, the limitation of being specific to small scale networks, and the lack of multicast support.

3.4 Sensor Networks

Wireless sensor networks represent another promising scenario for multihop networks [EGHK99, KKP00, SGAP00, PK00, ASSC02, AWSC02, TAGH02, CK03]. Even though these networks are not explicitly evaluated in this thesis, they are briefly discussed here for completeness. Sensor networks are also self-organizing networks intended largely



(a) DSR broadcast for route discovery



(b) DSR reply message

Figure 3.8: DSR route discovery

for monitoring systems used in application scenarios such as environment monitoring in inaccessible terrains, operations in hostile fields, patient monitoring in hospitals, traffic surveillance, and so on. As in ad hoc networks, the nodes comprising a sensor network must be able to forward data for their corresponding neighbors and may move unpredictably. However, despite the similarities between ad hoc and sensor networks, the latter have particular features making them a very special environment.

Sensor networks rely on nodes with very limited computational capability, low battery autonomy, and short transmission range. To compensate these limitations, the sensor nodes are in general densely deployed. This procedure not only increases the probability of end-to-end connectivity but also saves energy expenditure in every node. Because of such features, protocols for sensor networks are more focused on extending the lifetime of the communicating nodes (by efficient use of the nodes capabilities) than on highly efficient throughput achievements.

In conformity to the description in [AWSC02], a wireless sensor network is typically composed of the following elements: the sensor nodes scattered in the field, a data processing center for handling the data generated by the sensor nodes (and if needed issuing commands to the sensor nodes), a sink node for collecting data from the sensor nodes and forwarding commands from the data processing center, and a conventional network to connect the sink node to the data processing center for data delivery. In this general architecture, packets may be transfered not only from the sensor nodes to the processing center, through the sink node, but also in the reverse path. This depends on the network purpose. It is important to note that only the sink node communicates with the processing center, which implies that this node is more powerful than the regular sensor nodes with regarding data processing and transmission range. The conventional network can be any network technology available such as the Internet, a satellite network, a cellular network, etc.

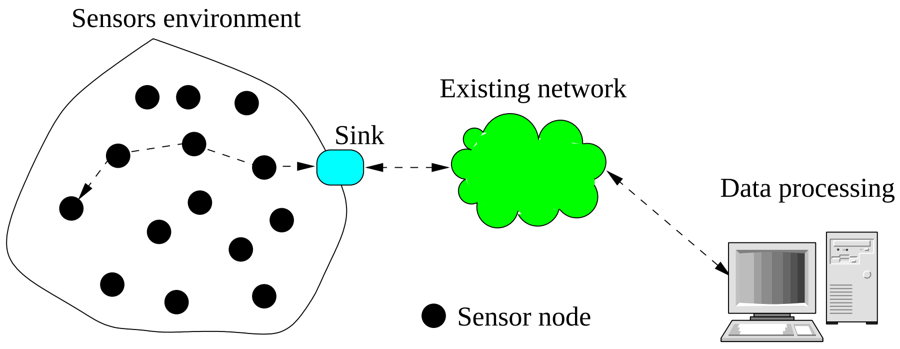


Figure 3.9: Typical sensor network architecture

As mentioned above, the protocols for sensor networks have to meet some stringent requirements inherent in these networks. Specifically, the MAC layer protocols should not waste neither energy nor bandwidth in overhearing the medium for long idle periods or transmitting with high power, because the nodes are usually close to each other [SGAP00]. Regarding the routing protocols, their design should take into consideration the low mobility pattern associated to typical sensor networks. As a result, table-driven schemes are more appropriate for sensor networks since they avoid the substantial energy expenditure involved in the routing discovery and maintenance of on-demand routing protocols such as AODV and DSR. Various routing protocols to meet such requirements have been proposed in recent years [HKB99, SGAP00, HCB00, PB01].

Concerning transport protocols for sensor networks, reliability (as that provide by TCP) is not necessarily needed in many of existing scenarios. In simple scenarios such as temperature and wind speed monitoring, it is tolerable to have some lost data. However, in some circumstances it may be crucial to assure data delivery by using a reliable protocol. A typical example is the one in which the sensor nodes can be reconfigured by receiving commands from the processing center. Then the configuration data must be reliably delivered. In general, the transfer of control and management data require reliability. The key challenges for transport protocols in these environments include the need of both small code due to the limited processing/memory capabilities of the sensor nodes, and tailored retransmission strategy for energy efficiency [WATC02, SH03, SAA03, DVRA04].

The explanation above indicates the need of new schemes tailored for sensor networks, but effective solutions are still to come. Actually, it is not yet sure if future sensor networks will make use of the TCP/IP protocol suite for the sake of the interoperability with the global Internet. There have been some researches showing that TCP/IP may be viable for sensor networks [DVRA04]. Nevertheless, further developments are needed to define the adoption of protocols for future sensor networks. Development of protocols for sensor networks represents certainly a wide open research area.

3.5 Summary

This chapter introduced multihop wireless networks. These networks are necessary for establishing long-range wireless communications in networks such as ad hoc and sensor networks. In particular, ad hoc networks mechanisms were addressed in greater detail because these are the primary target environments of this thesis.

The most known routing protocols for ad hoc networks are AODV and DSR. Both are on-demand protocols, and the former minimizes broadcast messages typical of table-driven routing protocols but supports only symmetric paths. The latter works on a hop-by-hop basis and does not need conventional route tables because the end-to-end route is carried in every packet header. This may incurs in prohibitive traffic overhead. DSR also supports non symmetric paths and allows nodes to have several alternative paths to the same destination.

The IEEE 802.11 standard specifies the MAC protocol for ad hoc networks. In 802.11, nodes may share the wireless medium smoothly due to the CSMA/CA mechanism in place. The design of 802.11 also includes functionalities for avoiding typical hidden node problems by the use of the RTS/CTS control frames. The efficiency of 802.11 is, however, to be improved for multihop scenarios relaying on several end-to-end hops. The problems that arise under such scenarios are discussed in chapter 5 from a TCP perspective. The next chapter introduces fuzzy logic as a background to chapter 6 that proposes a fuzzy logic based approach.

Chapter 4

Fuzzy Logic

4.1 Introduction

A perceptible number of applications based on fuzzy logic have emerged in recent years. Typical applications include pattern recognition in a broad range of fields such as voice, image, and handwriting; electronic system controls as in a process temperature controller or in an regulated electric power distribution; decision making processes such as routing decisions in a complex, large computer network; and many more. These applications rely on data that are commonly characterized by imprecision and uncertainties. Fuzzy logic is appropriate to handle these data by performing reasoning in a more human-like way than traditional systems. We introduce in this chapter the details comprising fuzzy logic as a background to the fuzzy logic based approach presented in chapter 6.

4.2 Fuzzy Logic Principles

Fuzzy logic may be seen as a superset of conventional logic (Boolean) that has been extended to handle the concept of partial truth. It was first introduced by Lotfi Zadeh in the 1960s [Zad65] as a means to model the uncertainties of natural language, and has been widely used for supporting intelligent systems [Jan01, Kul01, Mat02]. Fuzzy logic efficiency in dealing with uncertainties existing in physical systems makes it very attractive for decision making systems [Jan93, Cox94, Zad96, DLJL00, CSH$^+$01, CM01, OB04c, OB04a].

Zadeh observed that conventional system modeling becomes increasingly difficult as the system complexity increases, leading to a situation in which a precise modeling is too costly (or even impossible) and not really relevant. The concepts involved in fuzzy logic theory combine a formal mathematic theory with a representative system description based on observable reality. In fuzzy logic, realistic transitions among the various states that may characterize a system are considered. This allows smooth decision making processes that work similarly to the way human beings do. As a consequence, fuzzy logic based modeling are generally simple, tractable, and efficient. Yet, fuzzy logic is an evolving theory and so many concepts in it are sometimes open to allow alternative methods by the research community.

4.3 Fuzzy Sets

Fuzzy sets are seen as an extension of the mathematical concept of set. A set is a collection of objects called elements of the set. In the conventional set theory, an item from a given universe is either a member or non-member of a given set. This concept allows practically any collection of elements in real life to be associated to a mathematical definition of set. To further illustrate these definitions, every of the examples below are well defined collections of elements that may be called sets:

- The set of positive integers less than 5. This is a finite set containing 5 elements: 0,1,2,3,4.
- The set of 200-year old men. This set has surely no member, and so it is an *empty* set.
- The set of sampled temperatures higher than 40 Celsius. Despite being an infinite set, it is always possible to determine whether a given sample is a member or not.

Therefore, a set is fully characterized by its elements. A finite set is generally represented by a list of elements such as A=$\{0,1,2,3,4\}$ which has clearly 5 elements. An infinite set cannot be represented by a list though, and thus a property (or properties) characterizing the elements in the set has to be employed. In the last example above, for example, the elements of the infinite set is completely specified using a *predicate* such as $x > 40$. So, there are two manners to describe a set: a list of elements of the set or using a predicate.

By the definition above, there are only two possibilities for an element relative to a set: either member or non-member of the set. Zadeh's motivation for a more general definition arose based on the fact that many sets have more than a simple *either-or* criterion for membership. For instance, in the set of *young people*, a baby is clearly a member (true), a 70-year old man is certainly not a member (false), but people aged 30-40 are not easily associated to either membership groups. Another typical example is the people height. In this case, it is unrealistic to say that two individuals who differ in height by less than a millimeter belong to different sets such as short and tall sets. With these problems in mind, Zadeh proposed the concept of *degree of membership*. By this concept, the degree of membership is changed gradually from *false* to *true* rather than abruptly.

4.4 Universe of Discourse

The *universe of discourse* or simply *universe* contains the elements that may be considered for membership of a given set. The purpose of this definition is to prevent undesirable measurements from being taken as a set element. For instance, a negative value in a system measuring water level. Typical examples of *universe of discourse* in fuzzy sets include:

- The set $s > 10$ could count on all positive measurements as its *universe of discourse*.
- The set of young people could have all human beings in the globe as its *universe of discourse*. Alternatively, the set could have the numbers in the range [0,100] as the ages of the *universe of discourse*.

4.5 Membership Function

The mapping of every element in the universe of discourse into the fuzzy set space is accomplished through the use of *membership functions*. These functions follow a predetermined curve shape which should reflect closely the event they represent. For every element mapped into a *membership function* there is a corresponding degree of membership in the range [0,1]. If the degree of membership is different from zero, the element is said to belong to a set called *support* of the fuzzy set. Fig.4.1 depicts a membership function which could represent the *young people* example mentioned above with the *x-axis* representing the people ages. The *membership function* is usually represented as $\mu(x)$ and may be chosen from a number of shapes depending on the system purpose.

There are a number of curve shapes that have been used in the research community, but the most important ones are: the s-shaped (also called a s-curve), the reverse s-curve (called z-curve), the bell-shaped (may obtained by either a π-curve or Gauss-curve), triangular, and trapezoidal. The choice of the most appropriate curve shape is subject of discussions, but the beauty of these open possibilities is that tailored solutions may be achieved by using customized shapes for the membership function. The non-linear and exponential membership functions above are mostly generate using the formulas below.

s-curve: It is obtained using three parameters: its zero membership value (α), the complete membership value (γ), and the third piece of information called inflection or crossover point (β). The inflection is the point in which the degree of membership is 50% true. The value of the curve for an element x is computed as:

$$s(x,\alpha,\beta,\gamma) = \left\{ \begin{array}{ll} 0 & ,x < \alpha \\ 2((x-\alpha)/(\gamma-\alpha))^2 & ,\alpha \geq x \leq \alpha \\ 1-2((x-\gamma)/(\gamma-\alpha))^2 & ,\beta \geq x \leq \gamma \\ 1 & ,x \geq \gamma \end{array} \right\} \tag{4.1}$$

The characteristic of the s-curve is shown in Fig. 4.1. The inflection point is normally chosen by an analyst to represent a desired distribution of interest. The curve moves from no membership at its extreme left side to full membership at its extreme right side. In fact, the membership function is pivoted around its 50% degree of membership which is exactly the point of inflection. The reverse s-curve, called z-curve, may be easily obtained from 4.1. The z-curve is illustrated in Fig. 4.2.

bell-shaped-curve: The pi-curve is one the most used membership function of the family of bell-shaped curves. However, its computation is slightly more demanding than the one requested by the Gaussian membership function. Both curve shapes provide a smooth descent gradient from the central value to a zero membership degree along the universe of discourse. The Gaussian or exponential curve is defined by two parameters: a constant k from the universe of discourse defining the center of the curve, and another single value γ specifying the width of the curve. The value of the Gaussian curve in the universe of discourse x is given by (4.2), and its shape is shown in Fig. 4.3. Note that the inflection point in this case is automatically determined.

$$s(x, k, \delta) = e^{\frac{-(x-k)^2}{2\delta^2}} \tag{4.2}$$

Apart from the membership function curves above which are termed *continuous* membership functions, there exist *discrete* memberships as well. The latter is built from a list of points (vector) and may be more efficient in terms of processing, but requires more storage space. Discrete membership functions are not further addressed in this thesis.

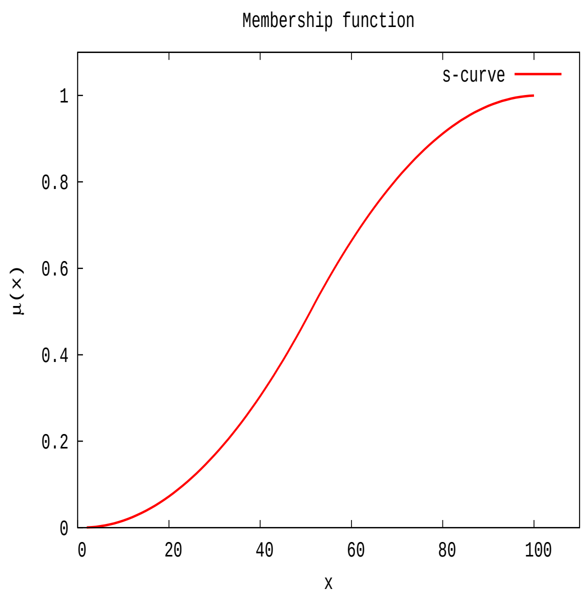


Figure 4.1: Typical membership function shape (S-curve)

4.6 Singleton

A fuzzy set may be described as a collection of elements in the universe of discourse with their corresponding degree of membership given by the membership function. Thus, it is correct to say that a fuzzy set A is a collection of ordered pairs, as follows.

$$A = \{(x, \mu(x))\} \tag{4.3}$$

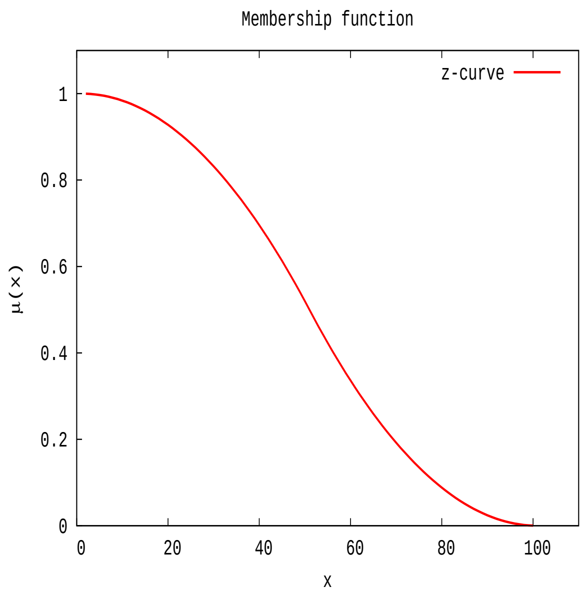


Figure 4.2: Typical membership function shape (Z-curve)

In the equation above, item x is part of the universe of discourse and $\mu(x)$ is its degree of membership in A. The single pair $(x, \mu(x))$ is denoted a fuzzy *singleton*. There are situations in which a fuzzy singleton is more appropriate to represent a discrete fuzzy output. Having defined a fuzzy singleton, the complete set A can be seen as the union of its constituent singletons. In such cases, it is convenient to specify a set A as vector **a** as shown in (4.4), in which each position i (1,2,...,n) corresponds to an element in the universe of n points.

$$\mathbf{a} = (\mu(x_1), \mu(x_2), ..., \mu(x_n)) \tag{4.4}$$

4.7 Linguistic Variables

While an algebraic variable takes numbers as values, a *linguistic variable* takes word or sentences as values. *Term set* is the name given to the set composed of the values that a linguistic variable may contain. Using linguistic variables, a variable called height would assume values such as “short” with degree of membership 0.1, “tall” with a degree of 0.7, and “very tall” with a degree of 0.9. This concept was introduced by Zadeh to abstract the inference process, so the fuzzy logic computation may be as simplified but representative as possible. As a further example, let x be a linguistic variable called temperature. The term set T of such a linguistic variable might be the one below. Words like *very, a bit, not so, etc*, are called linguistic *modifiers*. They are operators on fuzzy sets that change the meaning of the term to emphasize its tendency.

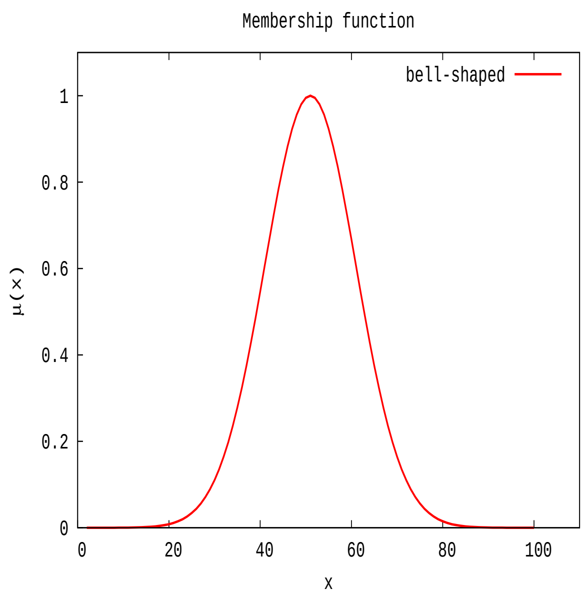


Figure 4.3: Typical membership function shape (bell-shaped curve)

$$T = \{\text{“}\textit{very cold}\text{”}, \text{“}\textit{cold}\text{”}, \text{“}\textit{not so cold}\text{”}, \text{“}\textit{not cold at all}\text{”}, \text{“}\textit{warm}\text{”}, \text{“}\textit{a bit hot}\text{”}, \text{“}\textit{hot}\text{”}, \text{“}\textit{very hot}\text{”}\}$$

4.8 Fuzzy Set Operators

Just like conventional sets, fuzzy sets also count on a specific number of operators for combining and modifying their membership functions in an appropriate manner. A fuzzy set operation constructs a new set from one or more input sets. For instance, Fig. 4.4 shows that the intersection of sets A and B is a new fuzzy set having its own membership function. Following the conventional fuzzy logic operations initially defined by Zadeh, three basic operations fulfill the needs of most typical fuzzy logic based systems. Let A and B be fuzzy sets on a mutual universe of discourse with membership functions $\mu_A(y)$ and $\mu_B(y)$ respectively.

Intersection: the *min* operator represents the intersection of A and B. The elements of A and B are operated one-by-one and the minimum of them is taken as the output.

$$A \bigcup B = min(\mu_A(x), (\mu_B(y))$$

Union: the *max* operator represents the union of A and B. The elements of A and B are operated one-by-one and the maximum of them is taken as the output.

$$A \bigcap B = max(\mu_A(x), (\mu_B(y))$$

Complement: a fuzzy set complement is obtained by subtracting from 1 each element comprising the set.

$$\overline{A} = 1 - \mu_A(x)$$

The fuzzy set X, resulting from an operation of two or more sets, is said to be a subset of a set Y if its membership function is less than or equal to the membership function of Y. This association is represented mathematically as $X \subseteq$ Y. In Fig. 4.4, for example, $(A \bigcap B) \subseteq (A \bigcup B)$.

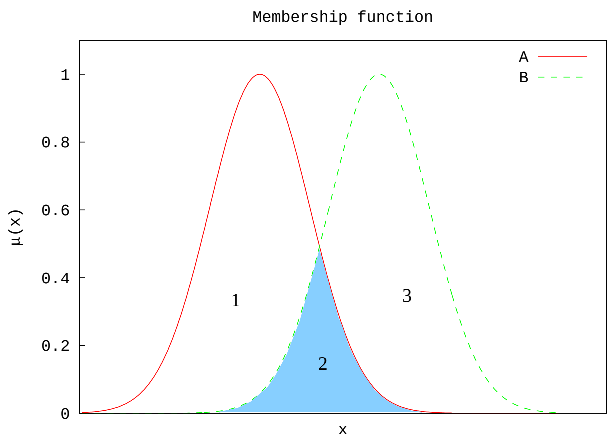


Figure 4.4: Basic fuzzy set operations (1:$A \bigcup B$, 2:$A \bigcap B$, 3:$\overline{A \bigcup B}$)

4.9 Inference Process

In order to reason about the data under consideration and produce appropriate outcome, the inference process relies on an *implication method* and a *rule base*. The *modus ponens* inference technique used in classical logic is also employed in fuzzy logic to infer the existence of a consequent state given an antecedent or premise state, as shown below. Various implication technique have been propose in the literature, but the most widely used one is the min-max inference method which is addressed in detail below.

Fuzzy rules: a fuzzy system contains generally a certain number of rules specifying the system behavior against the input variables. This defines a *rulebase* over which the inference process produces outputs. Each of the rules in the rulebase may have p *antecedent* clauses that define conditions and one *consequent* clause defining the corresponding action. Typically, a complete rule consisting of q consequents may be decomposed into q rules, each having identical antecedents and one distinct consequent. The general form of the n$_{th}$ fuzzy rule in the rulebase is:

$$R^n\text{:}\quad \textbf{if}\,\underbrace{\{(x_1\ is\ F_1^n)\ \textbf{and}\ (x_2\ is\ F_2^n)\ \textbf{and}\ldots(x_p\ is\ F_p^n)\}}_{\text{antecedents}}\ \textbf{then}\ \underbrace{(y\,is\,G^n)}_{\text{consequent}} \tag{4.5}$$

In (4.5), F_1^n and G^n are fuzzy sets associated to the input and output variables x_K and y, respectively, with $k = 1, 2, ..., p$. These rules are referred to as *if-then* rules. As an example of (4.5) we could have: **if** (*temperature* is warm) **and** (*humidity* is high) **then** (room is hot). A fuzzy system is said to be parallel because it performs reasoning on every rule in this rulebase toward a final inference. However, the operations performed on these rules are really simple, which is advantageous regarding computational processing.

Implication: The operation of applying the result of the antecedent to the consequent in the *if-then* rule is the implication. The most well-known implication method is called *Mamdani implication*. This method composes the *min-max* inference method explained in the following.

Min-max inference method: This is by far the most employed and investigated inference method. The min-max inference method consists of two parts: the implication and the aggregation. As mentioned above, *Mandani implication* is used in it. This implication method shapes the consequent (the output fuzzy set) on the basis of the antecedent as follows. Every rule is executed in sequence and the corresponding consequents receives the minimum of all antecedents. At the end of this process, there are a certain amount of fuzzy sets (output fuzzy sets) containing the result of the implication process. The *Mandani implication* is formally defined as shown in (4.6), where μ_{ant} is the the minimum among all antecedents and μ_{conseq} is the current value of the consequent that is going to change if $\mu_{ant} < \mu_{conseq}$.

$$\mu_{conseq}(x_i) = min(\mu_{ant}, \mu_{conseq}(x_i)) \tag{4.6}$$

Once the output fuzzy sets have been obtained, the next step is the *aggregation* of these fuzzy sets toward a single final fuzzy set. Thus, the aggregation unifies the outputs of all the rules. The inputs for the aggregation are either truncated or changed output fuzzy sets that are generated by the implication process. In the min-max inference method, the aggregation process results in a fuzzy set containing the maximum value among those generated by the implication process. This aggregation method is formally obtained as shown in (4.7), where μ_{aggr} is the current value in the aggregation fuzzy set and $\mu_{conseq}(x_i)$ is the element i obtained in the implication process.

$$\mu_{aggr}(x_i) = max(\mu_{aggr}, \mu_{conseq}(x_i)) \tag{4.7}$$

Since the output of the aggregation is a fuzzy set, it is needed a defuzzification process to generate a crisp (discrete) value. The defuzzification process then receives the fuzzy set created by the aggregation and produces a discrete value as the final inference process output. The most commonly employed defuzzification method is the centroid which is discussed below.

Fig. 4.5 illustrates the method considering two inputs and two rules only. There are two fuzzy sets labeled ”low” and ”medium” (linguistic variables) for both the input (x1, x2) and output (y) membership functions. The crisp input values are mapped into the membership functions (fuzzification) and assessed according to the rules in place.

Each rule (see Fig. 4.5) is applied to the involved membership functions in x1 and x2 and the minimum (min) of them is mapped into the associated output membership function in y (low or medium). The output of each rule is aggregated (max) into the deffuzifier which gives the final crisp value that will indicate in this case whether the outcome is to be assigned to ”low” or ”medium”. Several schemes for defuzzification exist, and for this simplified sort of output membership function (single value in y), the centroid (also called gravity-of-mass or center of gravity) method gives the weighted average over all output values in y.

4.10 Defuzzification Methods

Defuzzification is the last step of the fuzzy reasoning process. It converts the fuzzy reasoning output, which is a fuzzy set, into a proper crisp value that should accurately represent the whole inference process outcome. Hence, the defuzzification process receives a fuzzy set produced in the aggregation procedure and gives as output a single number. There are various techniques for defuzzification purposes, and the most known is the centroid method. Alternative methods include the height, the maximum, and the means of maxima, among many others. The four methods work as follows:

Centroid: this method finds the “balance point” of the solution fuzzy region by calculating the weighted mean of the fuzzy region [Cox94]. In fact, the method determines the center of gravity (centroid) y of a set B to be the output of the fuzzy logic system. Considering a continuous aggregate fuzzy set, the centroid is arithmetically given by:

$$y = \frac{\int\limits_{s} y_i \mu_B(y) dy}{\int\limits_{s} \mu_B(y) dy} \tag{4.8}$$

Where S is the support of $\mu_B(y)$. If discrete variables are used, then the equation above may be simplified by replacing the integrals with summation, as follows:

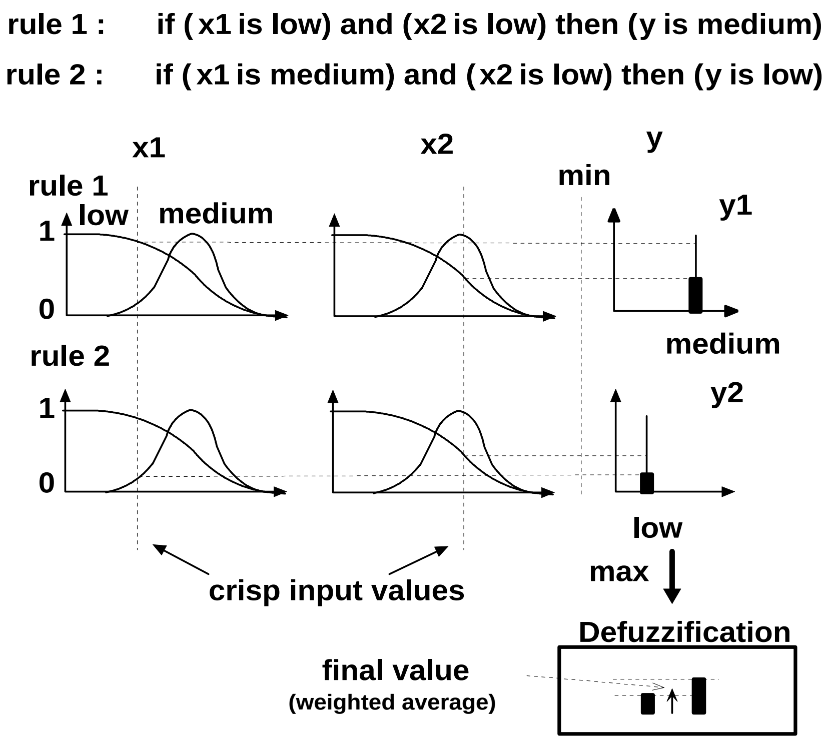

rule 1 : if (x1 is low) and (x2 is low) then (y is medium)
rule 2 : if (x1 is medium) and (x2 is low) then (y is low)
x1
x2
y
min
rule 1
low
medium
1
0
y1
medium
rule 2
1
0
y2
low
max
crisp input values
Defuzzification
final value
(weighted average)

Figure 4.5: An example of the min-max inference method

$$y = \frac{\sum_{i=1}^{n} y_i \mu_B(y_i)}{\sum_{i=1}^{n} \mu_B(y_i)} \tag{4.9}$$

Centroid defuzzification is the most widely used method because it has exhibits effective properties including: 1) the defuzzified values tend to move smoothly around the output fuzzy region, 2) it is relatively easy to compute, 3) it can be used for both singleton and nonsingleton fuzzy sets.

Height: This method evaluates first the $\mu_{B_i}(y)$ at y_1 and then computes the final crisp value, where y_1 denotes the gravity center of fuzzy sets B_i. The output y_{th} is thus given by:

$$y = \frac{\sum_{i=1}^{m} y_i \mu_B(y_i)}{\sum_{i=1}^{m} \mu_B(y_i)} \tag{4.10}$$

Where m denotes the amount of output fuzzy sets obtained from the *implication* and y_i is the centroid of fuzzy region i. This method is easy to use since the center of gravity in commonly used membership functions are know in advance.

Maximum: This method scans the aggregate fuzzy set and takes the output y for which $\mu_B(y)$ is the maximum. The maximum method is applicable in only a few classes of problems. Its output value is sensitive to a single rule that has most influence on the fuzzy rule set.

Center of maxima: This method finds in a multimode region, the highest plateau and then the next highest plateau. The midpoint between the centers of such plateaus is then selected as the output crisp value.

4.11 Fuzzy Logic Systems

Having introduced the concepts forming the fuzzy logic theory, the purpose of this section is to describe how a typical fuzzy logic based systems works. Fig 4.6 [CSH$^+$01] illustrates the main elements of a system based on fuzzy logic. In principle, a fuzzy logic system may be seen as composed of four stages:

- Fuzzification.
- Inference or reasoning.
- Aggregation.
- Defuzzification.

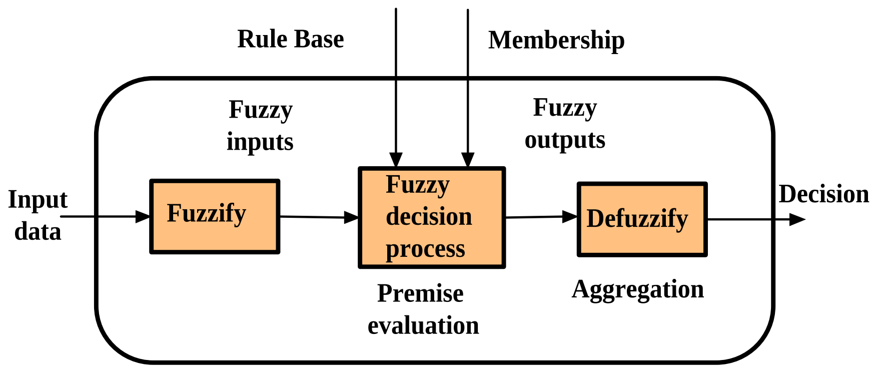


Figure 4.6: A fuzzy logic system

The fuzzification is the process by which the crisp input values are mapped into the membership values, thereby assuming values in the range [0-1]. Once the input data is already in the form of fuzzy set, the implication process modifies the output fuzzy sets (consequents) based on the antecedents of the predefined *rulebase*. Afterwards, the aggregation is performed over the implication outcome aiming to unit all the fuzzy rule outputs into a single fuzzy set. At last, the defuzzification process converts the aggregate fuzzy set to a single discrete value which is the result of all processing in the fuzzy system.

The process in Fig 4.6 works as follows. The system first receives crisp data, processes it and returns a crisp value pointing out its reasoning decision. The fuzzy decision process depends naturally on the contents of the rulebase as well as on the shape of the membership functions. A controller system based on fuzzy logic receives normally a feedback from the output so it may work on closed loop manner to maintain output stability.

4.12 Summary

The chapter briefly reviewed Fuzzy Logic theory. The basic concepts toward understanding fuzzification, inference, aggregation, and defuzzification were introduced. These elements make up a fuzzy logic based system. The concepts of this chapter will be used in chapter 6 where our proposal for packet loss discrimination using fuzzy logic is introduced. Before introducing our fuzzy logic based approach for enhancing TCP in multihop networks, we address in the next chapter the main problems faced by TCP over these networks.

Chapter 5

TCP over Multihop Wireless Networks: Problems & Evaluations

5.1 Introduction

It is estimated that TCP flows still account for about 90% of the total Internet traffic today [TMW97]. In the near future, more and more TCP flows are expected to be transmitted over wireless networks. The problem is that a regular TCP implementation cannot properly handle the medium related constraints inherent in multihop wireless networks. Even though TCP [Ste94, Dar81, Bra89, Ste97, APS99] has evolved significantly over the years toward a robust reliable service protocol, the focus has been primarily on wired networks. As shown in chapter 3, wireless networks are susceptible to high bit error rates and rely on very limited bandwidth when compared with their wired counterparts. Thus, traditional protocols like TCP must be adjusted to fit such new environments. This includes smooth coordination among the mechanisms in the Internet protocol stack layers, as well as efficient loss recovery strategies. This chapter introduces TCP over multihop networks, details the main related problems to be addressed by TCP, and discusses the major related work.

5.2 Impact of Wireless Transmission Medium on TCP

Wireless media are characterized by high, variable bit error rates (BERs), ranging typically from much less than 1% to over 20%. Compared with wired networks, wireless networks are susceptible to loss rates that are about two orders of magnitude higher. Wireless induced losses are caused primarily by fading, interferences from other equipments, and diverse environmental obstructions. Any of these factors may induce either single or bursty packet losses.

As opposed to wireless media, wired media have negligible BERs. In wired networks, any packet loss may be safely associated to congestion in the network. For this reason, regular TCP always handles packet losses as if they were caused by congestion. There are situations, however, in which TCP should not simply fully reduce its transmission rate in the face of a lost packet. Rather, it should determine whether this is indeed the best action to be taken on the basis of the actual reason causing the lost packet [CRVP98, HV99a, LS01]. Furthermore, the very scarce bandwidth in wireless channels requires that upper layer protocols like TCP avoid unnecessary, redundant transmissions into the medium toward high bandwidth utilization. These issues are discussed further below.

5.3 Interaction between TCP and Medium Access Control

The interaction between TCP and the IEEE 802.11 medium access control protocol is one of the most crucial problems to be addressed in multihop wireless networks. The fact is that 802.11 relies on the assumption that every node can reach each other or at least sense any transmission into the medium, which is not always true in a multihop scenario. Consequently, in some conditions the hidden node and exposed node problems can arise inducing capture effects [CGL00, TG99], which can impair not only TCP throughput but also the fairness among simultaneous TCP connections. We explain next, by means of particular examples, how these problems can take place.

5.3.1 Impact of Hidden Node and Exposed Node Problems

As described in section 3.3.1, the 802.11 MAC protocol uses the short RTS/CTS control frames to prevent the hidden node problem and consequently the exposed node problem. While this strategy works efficiently for scenarios where a maximum of three hops can be established, it does not scale for larger scenarios. We discuss here how these problems can impair a TCP connection.

For simplicity, we consider here the same simplified model introduced in section 3.3.1 in which only the transmission range is considered. To include the interference range here complicates the explanation and does not contribute much to the discussion. Fig. 5.1 illustrates the same chain topology discussed in section 3.3.1, where each node can only communicate with its adjacent neighbors. Yet, only a single connection exists in Figs. 5.1(a) and 5.1(b). In this scenario, node 4 is the hidden node as it interferes with node 3 preventing it from receiving the RTS originated by node 2, as shown in Fig. 5.1(a). Likewise, Fig. 5.1(b) illustrates a situation in which node 4 is the exposed node by not allowing node 3 to transmit the RTS to node 2.

Both problems above can affect TCP throughput as follows. In a hidden node condition, as illustrated in Fig. 5.1(a), the MAC layer of node 2 invokes its exponential backoff mechanism which attempts for a maximum number of times (typically 7 times) to retransmit (locally) the lost frame. In case it does not succeed, due to high traffic between nodes 4-5 or persistent disruption, node 2 drops the packet and sends back a route failure message to node 1. Then the routing protocol in node 1 attempts to find a new route to the destination, which by itself delays the forwarding and in the worst case leads the TCP sender to time out, further delaying the retransmission. Under the exposed node condition depicted in Fig. 5.1(b), a similar problem occurs. That is, if the traffic between nodes 4-5 is relatively large, the pending TCP ACK can be delayed more than the TCP timeout

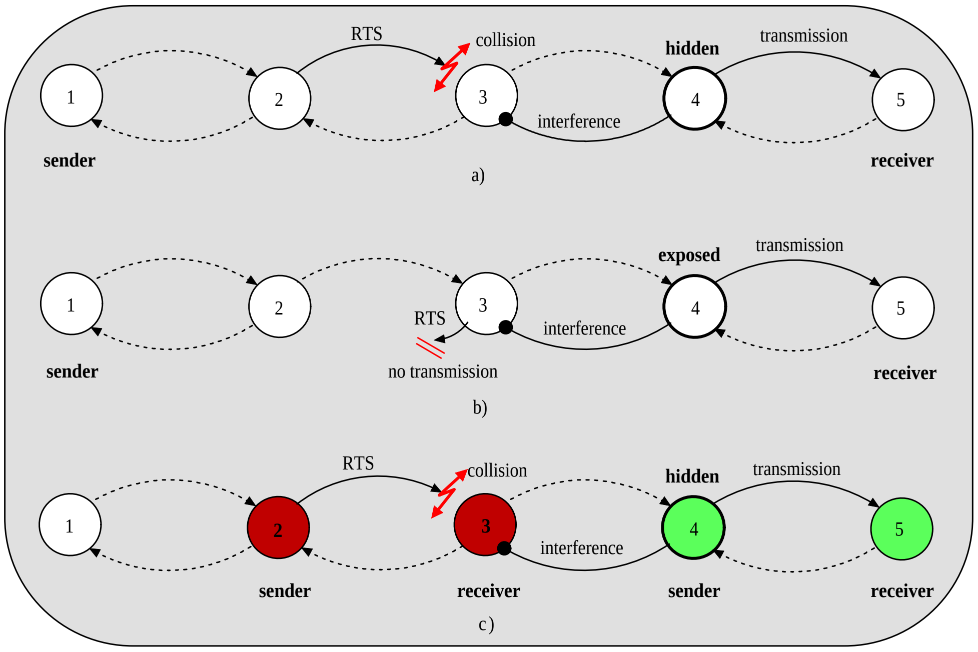


Figure 5.1: MAC layer problems affecting TCP performance: a) hidden node, b) exposed node, c) capture effect

interval. As a result, the TCP sender at node 1 also backs off by timeout.

In short, hidden node and exposed node problems may cause a lack of ACKs at the TCP sender, leading it to retransmit by timeout. A TCP sender should only invoke the timeout procedure as the last resort, since this implies in restarting the transmission rate from the lowest level. Retransmissions by the fast retransmit mechanism are less harmful to the end-to-end throughput. In the event of a packet loss, this mechanism just halves the $cwnd$ instead of resetting it to one. However, the fast retransmit mechanism may be prevented from being invoked if a single packet is dropped just before one of the two problems above begins. In other words, the fast retransmit mechanism may not receive 3 duplicate ACKs because of either hidden node or exposed node problems, which degrades performance as a retransmission by timeout is needed in such cases.

The problems above get worse as the number of hops increase. As a consequence, TCP end-to-end throughput decreases significantly as the number of hops grow as depicted in Fig. 5.2. This graphic was obtained by our simulation using the ns2 simulator [EFF$^+$00, EHH$^+$00]. TCP NewReno is the version simulated with the settings shown in table 7.2. One can see that the throughput decrease is pronounced for short number of hops (up to 4 hops), and then becomes less aggressive. This happens because the first nodes (from source to destination) in the chain interfere among themselves just as explained in the example of Fig. 5.1(a). As a result, the subsequent nodes do not get enough packets to substantially contribute to the hidden and expected nodes problems, as we explained in [OB05].

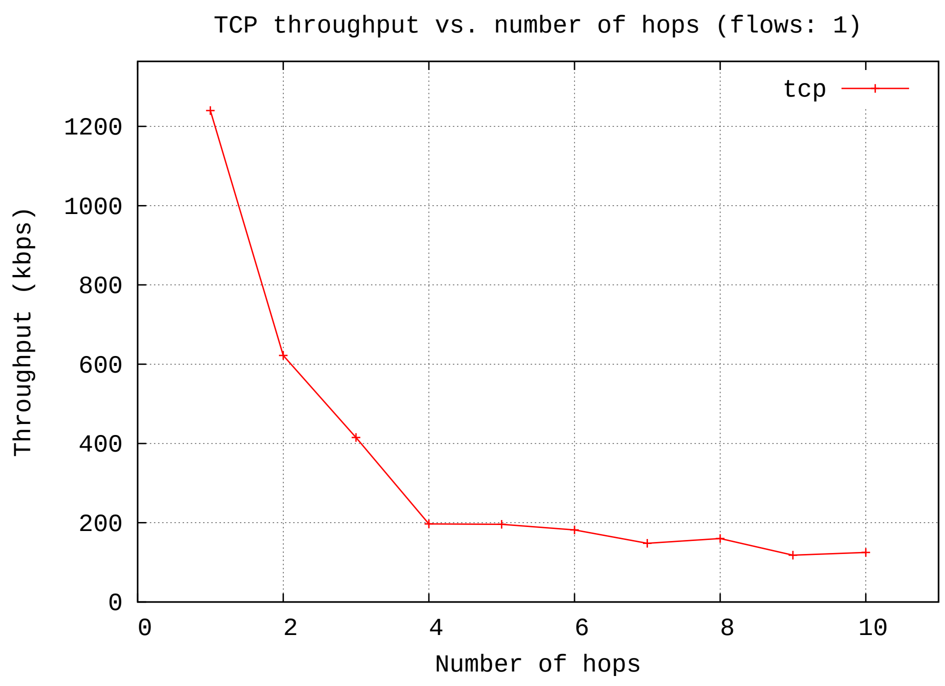


Figure 5.2: TCP throughput decreases as the number of hops increase

By using appropriate $cwnd$ size for TCP, as explained in section 5.3.3, the effects of such problems may be mitigated [XS01c, XS01a, CXN03, FZL+03, OB05]. The key point to understand the interaction between TCP and the MAC layer protocol is to take into consideration the fact that both hidden node and exposed node problems impose a limited spatial reuse property [OB04b] for multihop networks. Section 5.3.3 addresses these features in more detail. For simplicity, and if not strictly needed otherwise, we will hereafter refer to both problems simply as hidden node problem. This is commonly used in the literature because both disruptions cause the same undesirable effect in the network performance.

5.3.2 Capture Effect

The IEEE 802.11 standard has serious problems of unfairness. Its binary exponential backoff mechanism is not efficient for multihop networks because it suffers from the capture effect phenomenon. A capture effect refers to the situation in which a node monopolizes the medium at the cost of the other nodes. This problem may be caused by either hidden node or exposed node problems [XS01a, FZL+03].

Fig. 5.1(c) illustrates a capture effect situation due to the hidden node problem. There are two independent TCP connections, one between nodes 2-3 (connection 2-3) and another between nodes 4-5 (connection 4-5). Assuming that connection 2-3 experiences collisions due to the hidden node problem caused by the active connection 4-5, node 2 backs off and retransmits the lost frame. As explained earlier in section 3.3.1, at every retransmission the binary exponential backoff mechanism tends to impose an increasingly (although random) backoff interval, reducing the possibility of success for the connection 2-3.

Besides, if the MAC retransmission scheme fail at all, TCP eventually times out and also invokes its exponential backoff mechanism, further increasing the delay for the next attempt. As a consequence, the connection 2-3 hardly obtains access to the medium, while the another connection captures the medium. The MAC protocol is designed in such a way that if the connection between nodes 4-5 has a large data to transfer, it fragments and transmits it in smaller data frames with higher priority over all the other nodes. This is done by using a short IFS between the transfer of each fragment. Clearly, this procedure also contributes to the unfairness behavior described here.

Fairness problems due to capture conditions have been investigated in various researches including [GTB99, TG99, TCG01]. These proposed solutions attempt to adjust the length of the interframe space of the MAC protocol. Better fairness is achieved at the cost of bandwidth, though. It appears that no effective solution is viable by simply configuring either TCP or MAC parameters. Rather, hidden and exposed node problems have to be deeply addressed toward a robust approach that would provide not only a fairer but also throughput effective MAC protocol.

5.3.3 Transmission Interference in Multihop Wireless Networks

In addition to the transmission range in the 802.11 propagation model, there is also a interference range that is typically slightly higher than twice the transmission range. This means that a given node can only communicate with nodes placed at a maximum distance defined by its transmission range but can interfere with other nodes located as distant as its interference range. As a consequence, the IEEE 802.11 standard establishes a spatial reuse property defining the maximum end-to-end capacity that can be achieved in multihop networks. Note that for simplicity, interference range was not considered in the discussions above.

Taking the spatial reuse property into consideration, the design of upper layer protocols can be optimized to mitigate the effects of the hidden node problem for scenarios with more than three hops end-to-end. This subject has been investigated in detail in [LBC+01] and revisited in [FZL+03, CXN03, OB05] from a TCP perspective. In short, a TCP sender should limit its congestion window based on the number of hops end-to-end in order to improve performance, as explained next.

Fig. 5.3 [LBC+01] depicts a chain topology of 6 nodes. The transmission range is represented by the full line and the interference range by the dashed line. According to the explanation above, each node in Fig. 5.3 can only transmit to its immediate neighbors within its transmission range but can prevent transmission of the other nodes placed up to 2 hops away from it. From Fig. 5.3, it is clear that the transmission of nodes 2 and 3 will prevent node 1 from transmitting since node 1 is within their transmission and interference ranges, respectively.

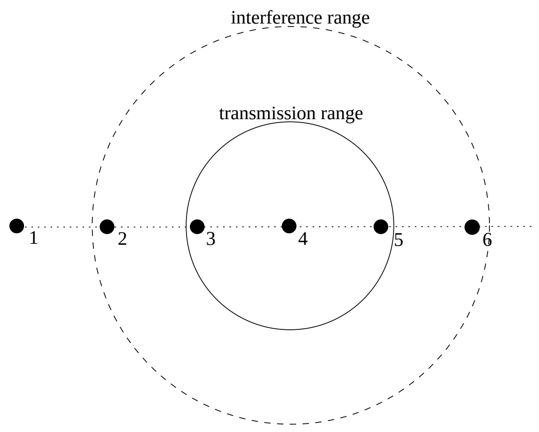


Figure 5.3: Spatial reuse in multihop networks

Furthermore, the transmission of node 4 will prevent node 2 from sending a response (CTS) to node 1, thereby impacting node 1 transmission as well. Thus, node 1 will only succeed in its transmission when node 4 is done with its transmission to node 5. Hence, in a group of 4 subsequent hops, only one can be active at a time, which imposes a maximum channel utilization of 1/4 relative to the 1-hop capacity.

This analysis considers an ideal MAC protocol that can schedule the nodes transmission evenly. Nevertheless, 802.11 does not work so smoothly, and because of that if the sender at the beginning of the chain is not able to pace its transmission rate to match the channel capacity, the channel utilization drops. In fact, the factor of 1/4 can only be achieved if the sender is able to inject into the network the exact amount of data the channel can forward [LBC$^+$01, OB05].

The explanation above indicates that a TCP sender should limit the size of its congestion window ($cwnd$) in order to achieve better performance. As described in section 2.3.2, $cwnd$ defines the maximum number of data packets a TCP sender may inject into the network at once without waiting for an ACK from the receiver. Thus, a limit of $h/4$ for $cwnd$ is presumably an optimal setting, where h is the number of hops in the chain topology. This means that long chains of nodes with many groups of consecutive 4 hops have higher limit for $cwnd$ than short chains. Interestingly, a 4-hop chain has an ideal limit of 1 packet for the TCP's $cwnd$. We evaluate in 7.2.1, through simulation, the optimal value for $cwnd$ in short multihop networks. The results are in line with the findings described here.

5.4 Disturbance of Routing Protocol Strategy on TCP

Routing protocols play a key role on TCP performance in environments in which the network topology changes rapidly. In the event of such changes, the routing protocol is responsible for discovering a new route connecting sender and receiver. Moreover, whenever a link interruption takes place, the route discovery has to be performed in a quick fashion to prevent the ongoing TCP connection from being disrupted [OB02]. Even if the topology is static, the routing protocol strategy may impact TCP performance as explained below.

As mentioned in chapter 3, DSR [JMH04] and AODV [PBRD03] have been considered the most prominent routing protocols for ad hoc networks so far. Both protocols work on an on-demand basis in order to minimize broadcasts messages in the bandwidth constrained medium. These protocols may affect TCP performance as follows.

In DSR routing, every node keeps a cache of routes it has learned or overheard in order to minimize the transmission overhead involved in periodic route advertisements. The problem with this approach is the high probability of stale routes in this environment where mobility as well as medium constraints are normally present. That can happen, for instance, when a route reply message is in its way back to the sender but the replied route is no longer valid due to either an involved node that has moved away or a link that has somehow been interrupted. The problem is exacerbated by the fact that other nodes can overhear the invalid route reply and populate their buffers with stale route information. So, unless the stale route can be detected and recovered in a fast manner, TCP can be led to backoff state, which deteriorates its performance critically.

The DSR problem above was studied in [HV99a, HV99b] where the authors showed that it can be mitigated by either manipulating TCP to tolerate such a delay or making the delay shorter so that TCP can deal with it smoothly. These studies showed that disabling route replies from caches can improve route accuracy at the expense of the routing performance in terms of transmission overhead, since every new route discover implies in a new broadcast to be sent. On the other hand, such an additional overhead is, in general, outweighed by the accuracy in the route determination, mainly for high mobility conditions, resulting in enhanced TCP throughput.

Concerning AODV, Although it is an on-demand routing protocol, it maintains a timer associate to each route for updating purposes. In environments where the sending node has regular intervals without any data to send, the overhead caused by routing updates may be prohibitive. Hence, the main drawback of AODV seems to be the transmission overhead it imposes for keeping the routes fresh, primary in highly loaded conditions [BMJ$^+$98].

In summary, the primary difference between both routing protocols is the route cache scheme used only in DSR and the maintenance scheme with time expiration used only in AODV. Yet, the characteristics of DSR and AODV above suggest that the routing efficiency and consequently the TCP performance depend not only on the network load but also on the mobility pattern in place. AODV is expected to perform better under moderate network load and high mobility since it updates routes periodically. On the other hand, DSR is likely to outperform AODV in scenarios facing high load and low mobility due to its route cache scheme.

The discussions here so far focused on mobile scenarios. We discuss now the performance of AODV and DSR in static scenarios since this may be significant from a TCP standpoint. As explained in chapter 3, DSR messages carry in the packet header the full route information from source to destination. As a result, the more hops the higher the traffic overhead. This may render DSR inefficient in scenarios where nodes are static and the network load is moderate, because its cache of routes plays no significant role in such scenarios where practically no link interruptions occurs. To further clarify this issue, we conducted some simulations for comparing AODV and DSR in a static chain topology. Fig.5.4 illustrates the outcome in which throughput against number of hops is presented. These simulations were conducted using the ns2 simulation with the general settings depicted in table 7.2.

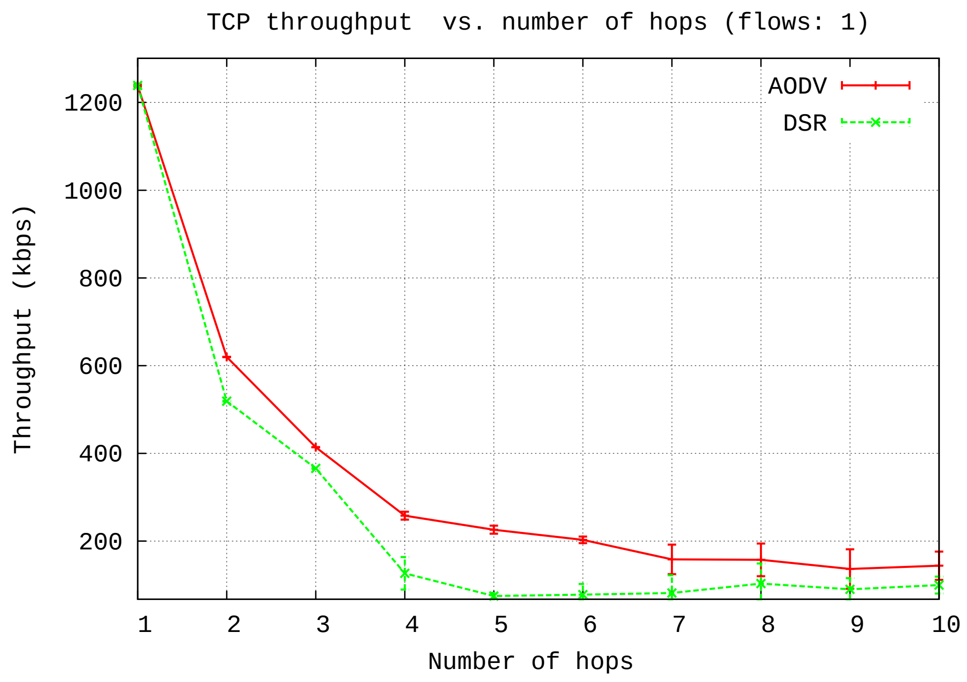


Figure 5.4: Comparison between AODV and DSR

The results in Fig. 5.4 confirm the predictions above in that DSR overhead may impact TCP performance to a great degree in such conditions. In this particular scenario where only one flow is present, AODV performs much better than DSR. Hence, from a TCP perspective, it is indeed difficult to say which protocol is the best. In fact, there are tradeoffs that must be taken into consideration to meet the application needs. We do not address this issue further in this thesis, and use only AODV in all subsequent evaluations.

5.5 TCP Dedicated Response to Wireless Constraints

TCP mechanisms are fine tuned not to cause collapse in the network over which it is running [Dar81]. As discussed in chapter 2, TCP was first designed to work in wired networks, where packet loss may be safely associated to congestion. This assumption has been the basis for TCP developments over the past two decades. As described in chapter 2, The main mechanisms aimed at avoiding aggressiveness toward the networks are:

- Slow start and congestion avoidance.
- Fast retransmit and fast recovery.
- Exponential backoff mechanism.

These mechanisms lead a TCP sender to slow down in the event of a packet loss and then to increase the transmission rate gradually. The drawback with such approach arises when TCP is working in a wireless network. In such networks, packet losses are not only caused by congestion but also by the lossy nature of the wireless medium, as mentioned in section 5.2.

As a result, TCP performance may be impaired in scenarios where single packet drops induced by the medium occur often. By too conservatively slowing down at every single drop, TCP may waste bandwidth since the communication channel is not really facing congestion. This problem can be avoided if the TCP sender is informed about the nature of any dropped packet. In other words, a TCP sender should be able to discriminate between congestion and wireless medium induced losses. Long link interruptions also have to be identified for the sake of good performance. Having these information allows a sender to take proper decisions when reacting to losses. This subject has been investigated by many researchers as outlined in section 5.7. Packet loss discrimination is also a contribution of this thesis being addressed in chapter 6.

Another problem originated by the highly conservative behavior of TCP is the use of the exponential backoff mechanism. This mechanism may induce the protocol to very long intervals of unnecessary backoff after a relatively long link interruption, as follows.

Fig. 5.5 depicts an example of a long link interruption triggering the TCP exponential backoff mechanism. The purpose of this figure is sole to aid our explanation rather than take into account all the details actually involved. More details can be found in [Ste94]. Besides, for the sake of simplicity, we use the term packet instead of segment as explained in section 2.3. With that in mind, Fig. 5.5 shows how the delayed answer of the exponential backoff mechanism can lead the TCP sender to a long idle period subsequently to the link restoration. We call this idle period ”dead time” because it is a real waste of time.

The example in Fig. 5.5 shows that both packet 3 (P3) and the acknowledgment of packet 2 (ACK3) are dropped due to a link failure (left vertical bar). As the sender does not receive the confirmation of packet 2 (P2) receipt, it retransmits P2 by timeout after 6 seconds (6 seconds is the typical initial RTO value which changes over time according to measured RTTs), and doubles its retransmission timeout (RTO) value. Whenever the timeout period expires, TCP sender retransmits P2 and doubles the RTO up to the limit of usually 64 seconds which is the maximum timeout allowed. After generally 12 unsuccessful attempts TCP gives up.

This example shows that shortly after triggering its timer for 64 seconds the link is recovered, but it is too late for TCP and it will stay over 1 minute frozen, which is indeed a dead time for the assumed starving connection. In terms of percentage, roughly 61% (100s/163s) of the interruption is due to link failure and the remaining 39% (63s/163s) is completely caused by TCP, which is certainly too much wastage.

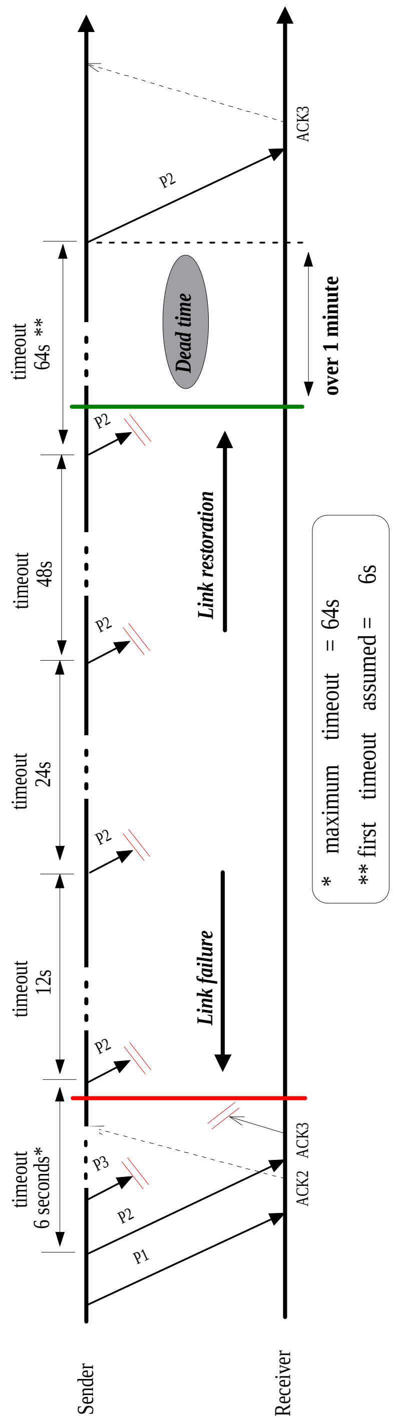


Figure 5.5: Drawback of TCP exponential backoff mechanism in multihop networks

5.6 Traffic Redundancy Avoidance in TCP over Multihop Paths

The reliability provided by TCP is particularly costly in multihop wireless networks where bandwidth is a very scarce resource. The access to the shared medium is indeed complicated due to the hidden node problem explained above. Because of that, the ACK packets used by TCP, to guarantee reliability, render a considerable burden to the wireless channel, thereby wasting precious bandwidth. In fact, the overhead caused by ACK packets is not negligible because the receiver must content for the medium using the RTS/CTS control frames exactly as the sender does. In additional, any transmission into the medium is subject to the random backoff mechanism.

Fig. 5.6 exhibits typical delays experienced by TCP data and ACK packets in a chain of nodes composed of three hops. This is the result of a simulation run we conducted using the ns simulator with the general settings illustrated in table 7.2. The figure emphasizes how substantial the impact of ACK packets may be. One can see that data and ACK delays are in the same order of magnitude. In this particular example, the ACK average delay accounts for over 40% of the total transmission time [OB05]. Thus, less than 60% is reserved for the useful data packets. We elaborate further on this in chapter 7.

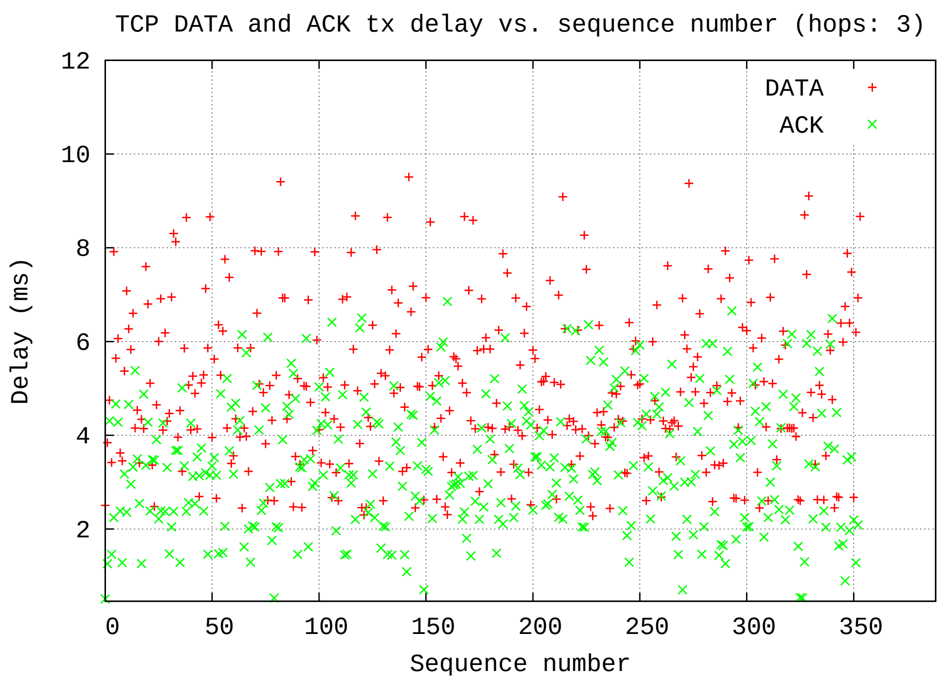


Figure 5.6: DATA and ACK delay in a typical wireless channel

These observations suggest that minimizing the number of redundant ACKs may result in better performance for TCP. In addition, the processing overhead largely at the receiver may be reduced substantially by using delayed acknowledgments, as mentioned in section 2.6.1. Note that the cumulative acknowledgment strategy used by TCP permits that the receiver does not transmit an ACK for each data packet it receives since later ACKs confirm the receipt of early ACKs.

The practice of reducing ACK transmissions must, however, be very well designed to prevent adverse effects since essential ACKs are always needed to guarantee effective TCP performance. A number of delayed acknowledgment strategies have been proposed in the literature to address this issue, which are presented in section 5.7. This is also a topic addressed by this thesis in chapter 7, where the advantages of our dynamic adaptive acknowledgment strategy are explained and evaluations are conducted.

5.7 Related Work

TCP performance in wireless environments has been investigated in several studies such as [BB95, BSAK95, BS97, TB00, ZT01, MCGW01]. Most of them are meant for cellular networks where only the last hop communicates through a wireless link. In this section, we focus on multihop networks in which long chains of wireless links may exist. However, we also discuss a few related work on cellular and wired networks that are relevant to our research described here. For the sake of clarity, we roughly classify the main related work into four classes as follows.

In section 5.7.1, we describe the techniques to improve bandwidth utilization by decreasing the number of medium access requests since that is costly in a shared medium. In section 5.7.2 we gather the proposals that deal with packet loss discrimination in wireless environments as a means to prevent TCP performance degradation in the face of wireless induced losses. In section 5.7.3, we address the schemes that either propose changes to the MAC layer or simply propose to adjust TCP parameters pursuing better TCP and MAC interaction. Yet, in section 5.7.4 we describe some important investigations on the performance of TCP over multihop networks.

5.7.1 Reduced Traffic and Medium Access Requests

TCP DATA and ACK Combination

This work [YYS$^+$04] proposed a technique to combine TCP data and ACK packets into a single packet in the intermediate nodes that have data and ACK to transmit in opposite directions. The rationale of this approach is that ACK packets should not use the same full time slot as data packets since the former is much smaller than the latter by containing only packet header. When using the same time slot as data packets, ACK packets induce considerable bandwidth wastage.

Thus, the proposed technique specifies two queues for the intermediate nodes, one for data packets and another for ACK packets. Whenever an intermediate node has a packet in both queues, it combines both into a single packet and transmits. Each receiving neighbor checks the combined packet and handles the packet if it is addressed to itself. Otherwise, the packet is simply discarded. In case an intermediate node does not have data and ACK in each queue, the node performs the regular transmission. A specific time slot sequence is needed to permit the sending node (intermediate) to overhear the transmission of the neighbors forwarding the combined packet that has been transmitted.

Simulation results report gains of up to 60% over multihop networks scenarios, but the simulated scenarios were quite restricted. It is also important to mention that this approach requires changes in every node in the network, which is a drawback as far as deployment is concerned. Nonetheless, the idea of reducing the amount of medium access requests in multihop wireless networks is very important for alleviating collisions. In chapter 7, we present a proposal to reduce such requests by adaptively adjusting the receiver to bring down the number of redundant ACKs it transmits.

Large Delayed Acknowledgment

The investigation in this work assessed through simulations the impact of delaying several ACKs on TCP performance in multihop wireless networks. We call this approach Large Delayed Acknowledgment (LDA) [EA03] because it proposes to enlarge the amount of delayed ACKs proposed in the standard delayed acknowledgment (DA). The purposed of this proposal is to reduce the number of ACKs transmitted by a TCP receiver in order to mitigate collisions between data packets and ACKs.

The changes proposed here relative to the standard DA specified in RFC 1122 consist of two procedures. The first one is to combine more ACKs than the two specified in the standard DA. The second procedure is associated to the session startup. During startup, the receiver starts not delaying any ACK and then as the incoming data packets arrive the receiver gradually begins delaying ACKs. First it delays one packet only, then two ACKs and so on until a maximum of four merged ACKs into a single ACK. The increase in the number of ACKs to be delayed is triggered by the sequence number of the incoming data packets. After reaching its maximum, this number is kept fixed. The rationale for the gradual ACK delay is that at the beginning of the session, there is no data packets in flight to request ACKs from the receiver. So, it makes no sense to delay the first ACKs in groups of four at startup but only in steady state condition.

The receiver uses a fixed timeout interval of 100 ms and does not react to packets that are out-of-order or filling in a gap in the receiver's buffer, as opposed to the recommendation of RFC 2581. Note that RFC 2581 refines the standard DA by adding response to out-of-order packets. To prevent the sender from missing ACKs when losses occur, the sender congestion window is roughly twice higher than the amount of ACKs that can be delayed at the receiver. The sender limits its $cwnd$ to ten packets.

The main limitation in this approach is the fact that it does not adapt to the wireless channel. Since it does not react to our-of-order packets, it always has to wait for its timer to expire before transmitting the buffered ACK. This may be quite inefficient in scenarios facing moderate to high loss rates. Another problem is the fixed timeout interval in the receiver. This interval should be based on the gap between the data packets arriving at the receiver. However, this gap changes not only with traffic conditions but also with the channel data rate.

With regards to the simulation results, improvements of as much as 50% over the standard TCP were reported. However, only a single flow were considered in all evaluations, which somewhat limits the relevance of the results. Our proposed technique introduced in chapter 7 addresses this issues by being adaptive to the channel and providing enhancements in a variety of relevant scenarios.

Extended Delayed Acknowledgment Intervals

This work [Joh95] investigated the impact of using extended delayed acknowledgments intervals on TCP performance in wired networks. It has performed various experiments in a testbed implemented on a SunOS 4.1.2 workstation. In the experiments, the kernel's TCP algorithm was changed to allow different number of combined ACKs by the receiver instead of only two as proposed in the specification of RFC 1122. The sender was adjusted to delay higher number of ACKs, ranging from 1 to 20 ACKs.

The main outcome is that delaying ACKs in large numbers is always beneficial in short-range networks but may be inappropriate for long distance networks, especially if congestion is occurring. This is a consequence of the high interference of the delayed ACKs on RTT estimation at the sender. The longer the end-to-end connection the longer the time for the TCP sender to detected lost packets, which may jeopardize the gains obtained by delaying more ACKs. Since the evaluations were restricted to wired networks, the results cannot be directly applied to multihop wireless networks. Nevertheless, the investigations in this work suggest that it may be efficient to reduce redundant ACKs in multihop wireless networks. We confirm this hypothesis in our approach in chapter 7.

Generation and Use of TCP Acknowledgments

This work [All98] conducted an extensive simulation evaluation on Delayed Acknowledgment (DA) strategies in wired networks. This work showed that TCP performance may be hurt by delayed ACKs especially during the slow start phase. One reason is that the exponential growth of TCP $cwnd$ in that phase produces burst of data in the network inducing packet drops in the routers' buffers. Another problem lies in the ACK-clocked strategy of TCP, in which the sender only increases its $cwnd$ by one upon each received ACK. This limits the sender data rate in scenarios where slow start is often invoked.

The author proposes two mechanisms to handle the side effects of delayed ACKs: *delayed ACK after Slow Start* and *byte counting*. The former requires signaling between sender and receiver to keep the receiver informed about whether slow start is active or not at the sender, so the receiver only delays ACKs when slow start is over. This speeds up data rate recovery during slow start. *Byte counting* allows the sender to increase its $cwnd$ on the basis of the number of bytes acknowledged by each ACK rather than on the amount of ACK themselves. This procedure can lead to prohibitive bursty traffic conditions, and so the author also suggested to limit the number of packets sent in response to each incoming ACK to two packets.

The simulation results showed that both mechanisms above can improve performance for implementations using delayed ACKs, but the main concern is the potential increase in packet drops that may happen. Even though this work was only simulated in wired scenarios, we believe its findings may be useful for wireless environments as well. We have not investigated that in depth, though.

5.7.2 Packet Loss Discrimination

TCP-Feedback

TCP-F [CRVP98] is a feedback based scheme in which the TCP sender can distinguish between route failure and network congestion by receiving Route Failure Notification (RFN) from the intermediate nodes. The idea is to push the TCP into a snooze state when such messages are received. In this state the TCP sender stops sending packets and freezes all its variables such as timers and $cwnd$ size, which makes sense since there is no available route to the destination. Upon receipt of a Route Re-establishment Notification (RRN), via routing protocol, indicating that there is again an available path to the destination, the sender leaves the frozen state and resumes transmission using the same variables values prior to the interruption.

In addition, a route failure timer is used to prevent infinite wait for RRN messages. The timer is triggered whenever an RFN is received, and in case it expires the frozen timers are reset allowing the TCP congestion control to be invoked normally. Results from this approach showed gains over standard TCP in conditions where the route re-establishment delay is high. This is a result of the low retransmissions involved in the protocol. Its performance was superior for scenarios with high rates as well. Nevertheless, the simulation scenario was quite simplified and so the results might not be much representative. For example, the RFN and RRN messages should be carried by the routing protocol, but no such a protocol was considered in the evaluation. Therefore, it is necessary more analysis toward this goal. Our two contributions in this thesis avoid such problems by being entirely end-to-end.

ELFN-based approach

In this approach [HV99a], TCP also interacts with the routing protocol in order to detect route failure and take appropriate actions when that is detected. This is done via Explicit Link Failure Notification (ELFN) messages that are sent back to the sender from the node detecting the failure. Such messages are carried by the routing protocol that needs to be adapted for this purpose. In fact, the DSR route failure message was modified to carry a payload similar to the host unreachable ICMP message.

Basically, the ELFN messages contain sender and receiver addresses and ports, as well as the TCP sequence number. In this way, the modified TCP is able to distinguish losses caused by congestion from the ones due to mobility. When the TCP sender receives an ELFN message it enters a stand-by mode, which implies that its timers are disable and probes packets are sent regularly towards the destination in order to detect the route restoration. Upon receiving an ACK packet, the sender leaves the stand-by mode and resume transmission using its previous timer values normally.

This scheme was only evaluated for the DSR routing protocol where the stale route problem was found to be crucial for the performance of this modified TCP. Additionally, the length of the interval between probes packets and the choice of which sort of packet to send as a probe were also evaluated. Only the former showed to be really relevant. It suggests that a varying interval based on RTTs values could perform better than the fixed probe interval used in this algorithm.

Another interesting investigation performed by this study was the impact of the ARP protocol on TCP efficiency, which call for improvements. In general this approach provided meaningful enhancements over standard TCP, as shown in the mentioned reference, but further evaluations are still needed. For instance, different routing protocols and under congestion conditions should be considered. Yet, more appropriated values for the probe interval should be determined. As mentioned above, we take a different approach to tackle this problems from an end-to-end perspective, as discussed in chapters 6 and 7.

ATCP Protocol

Differently from the previously approaches discussed above in this section, ATCP [LS01] does not impose changes to the standard TCP itself. Rather, it implements an intermediate layer between network and transport layers in order to improve TCP performance and still maintain interoperability with non-ATCP machines. This principle is similar to the that developed in [BSAK95] for cellular networks. ATCP relies on the ICMP protocol and ECN scheme to detect network partition and congestion, respectively. In this way, the intermediate layer keeps track of the packets flowing to and from the transport layer so the TCP congestion control is not invoked when it is not really needed. ATCP works as follows.

When three duplicate ACKs are detected, indicating a lossy channel, ATCP puts TCP into persist mode and quickly retransmits the lost packet from the TCP buffer. After receiving the next ACK, the normal state is resumed. In case an ICMP Destination Unreachable message arrives, pointing out a network partition, ATCP also puts TCP in persist mode which only ends when the connection is reestablished. At last, when network congestion is detected by the receipt of an ECN message, ATCP does nothing but forwards the packet to TCP so that it can invoke its congestion control normally.

This scheme was implemented in a testbed and evaluated under different constraints such as congestion, lossy scenario, partition, and packet reordering. In all cases the transfer time of a given file by ATCP yielded better performance relative to an unmodified TCP. However, the evaluation scenario was somewhat special, since neither wireless links nor ad hoc routing protocols were considered. In fact, the experiments were conducted in a wired network where the actual noisy pattern of wireless channels could not be reproduced. Moreover, some assumptions such as ECN-capable nodes as well as sender node being always reachable might be somehow hard to be met. Lack of connectivity can prevent an ICMP message from getting at the sender, leading it to retransmit continuously instead of entering persist mode. This is a typical problem of network oriented approaches. As already mentioned, we avoid these sort of problems by using end-to-end signaling only.

Packet Loss Discrimination Using Inter-Arrival Times at the Receiver

The key idea of this approach [BV98] comes from the observation that the receiver is better suited to detect losses than the sender due to the cumulative acknowledgment scheme used by TCP. If the receiver monitors the packet inter-arrival interval of the incoming data packets, it can perceive differences for dropped packets due to the wireless medium from the ones caused by congestion.

The receiver keeps track of the time interval between the consecutive data packets received and performs discrimination decisions as follows. First, it keeps a variable for the minimum inter-arrival interval ($Tmin$) measured so far. By checking the sequence number of the incoming packets, the receiver can infer whether a given out-of-order packet arrived in the estimated time, too early, or too late. The estimated time here refers to the number of $Tmin$ that should have been elapsed since the last in-order packet was received.

Assuming that the sender always has data to send, whenever its buffer overflows by congestion it drops a packet or a number of packets. This does not introduce any significant delay in the data stream being transmitted. Thus, the delay between the last in-sequence packet and the subsequent out-of-order packet received at the receiver should be much smaller than the expected by the receiver. In other words, this packet arrived too early indicating a loss due to congestion. Likewise, if the received out-of-order packet arrives too late, that is associated to loss by congestion.

On the other side, if the sender transmits the packet correctly but it is dropped by the medium, then the received out-of-order packet arrives around the estimated time. So, the receiver infers a loss by the medium.

The main advantage of this approach is its simplicity by requiring changes at the receiver only. Nevertheless, it only applies to scenarios where there is a wired link associated to a wireless link that is the bottleneck of the end-to-end connection. Moreover, this work has considered only one flow in its evaluations, which can hide the fluctuations in the inter-arrival intervals that may occur for various concurrent flows. We believe that smoothing the inter-arrival intervals is always needed to ensure robustness and accuracy. In our proposal in chapter 6, we use smoothed inter-arrival intervals because from our experience we learned that the measured samples are too noisy to be used directly.

Packet Loss Discrimination using Multiple Metrics

This investigation [FGML02] assessed TCP improvements by using multiple end-to-end metrics instead of a singe metric. The authors claim that a single metric may not provide accurate results in all conditions. They used four metrics: a) Inter-packet delay difference (IDD) at the receiver, b) Short-term Throughput (STT), c) Packet out-of-order delivery ratio (POR), and d) Packet Loss Ratio (PLR).

IDD is calculated by subtracting the gap between two consecutive packets received at the receiver from the same gap at the sender. IDD increases with congestion but not with random channel error or packet sending behaviors. STT is also intended for network congestion identification, and is less sensitive to short term out-of-order packets than IDD.

So, STT is assumed to be more robust to transient route changes. To use STT alone is, however, not recommended due to measurement noise introduced by bursty channel error, network disconnections, or changing TCP source rates. So, IDD and STT are combined to jointly identify network congestion.

POR is intended to indicate a route change event. During a route switching time, packets from the old path may arrive out-of-order at the receiver. POR is increased whenever a packet arrives out-of-order. PLR is computed at regular intervals to determine the number of missing packets in the current receiving window at the receiver.

To specify the values of IDD and STT to be used in associating them to congestion conditions, the sampled values were classified in high and low if they were within the top of bottom 30% of all samples. From the measured values, it was observed that congestion can be safely identified if both IDD is high and STT is low. For a non-congested channel, a route change or channel error may be the cause of a packet loss. A burst of high POR sample values is classified as a good indication of a route change, and a high PLR as a good indication of a high rate of channel error. Finally, a link disconnection can be identified by the sender if the current network state estimation is non-congestion when a retransmission by timeout occurs.

Considering the implementation of this approach, the receiver always computes the network state based on the metrics above for every received ACK. It sends the network state to the sender in every ACK transmitted. This information is only needed when a packet drop is detected. If the sender perceives a network disconnection, a probing procedure is initiated until connectivity is reestablished. Backward compatibility for interaction with traditional TCP is possible.

This scheme has been evaluated in simulation using the ns2 simulator and also via implementation in linux kernel. The results are positive but focused on mobility conditions. The interaction between TCP and the MAC layer were not investigated in detail. Our proposal in chapter 7 shows that techniques that simply detects the reason of dropped packets may not be effective in such scenarios due to the very low maximum bandwidth utilization that can be achieved using the basic strategy of conventional TCPs.

Packet Loss Discrimination using HMM

In this approach two techniques are used to discriminate packet loss in a wired/Wireless environment: packet loss pairs (PLP) and Hidden Markov Modeling (HMM). A PLP is characterized as a pair of packets sent end-to-end where one of them is lost and the other carries the network condition experienced by both packets until the instant the drop occurs. Using PLP, the distinguishable RTT distributions regarding congestion and medium error can be determined. In fact, the mean and standard deviation of the distribution formed by RTT measurements are associated to the state of the network during the time of a packet loss. An HMM is then trained using only RTT of interest, i.e., those observed in the neighborhood of the dropped packet.

The HMM design uses a 4-state model trained with 10000 loss pair RTT measurements that are collected from a simulation using the ns2 simulator. The RTT distribution for three conditions are simulated: congestion, medium error, and both. Afterward, the

HMM is trained taking the RTT measurements obtained when both constraints are in place. The RTT distribution clearly exhibits a distinguishable pattern for network congestion. That is, under congestion, the RTT of congestion loss pairs are very concentrate around the RTT value corresponding to buffer overflow. On the other hand, the RTTs of wireless loss pairs are wide spread.

Comparing the RTT distributions with the Gaussian models for each state of the trained HMM, a close match between one of the states and congestion is established. A labeling technique is then proposed to assign each of the HMM states to either congestion or medium error constraints. The evaluations showed that the proposed technique can provide high labeling accuracy in most practical network configurations. In particular, if buffer are not always overloaded, it can very accurately attribute congestion to congestion while simultaneously provide high accuracy on labeling wireless induced losses.

This proposal is interesting for not requiring any specific support from the network and also for using only passive measurements via RTTs. In addition, its results are superior to the Vegas predictor that has been proposed in the literature. The main concern with this approach seems to be the processing overhead at the sender since HMM are well-known as heavy algorithms as far as computation are concerned. Our technique in chapter 6 is greatly influenced by this work in that we also observe that RTT measurements can be useful to discriminate losses in ad hoc networks. However, instead of using HMM, we make use of fuzzy logic to alleviate computational processing at the sending node.

5.7.3 MAC and TCP adjustments

Setting TCP Congestion Window Limit on Multihop Networks

The key idea of this proposal [CXN03] lies in the observation that a TCP sender should limit its congestion window in order to avoid collisions and further degradation in the wireless channel. In fact, a TCP sender should limit its $cwnd$ to the bandwidth-delay product of the wireless channel. Such a limitation is dependent on the number of hops in place, namely the round-trip hop count (RTHP). Thus, this work proposes to set the congestion window limit (CWL) on the basis of the RTHP dynamically.

Irrespective of the MAC layer strategy in place, the upper bound for CWL is given by k·RTHP, with $1/8 < k < 1/4$. In the case of 802.11, $k = 1/5$ is found by simulation to be the best value. The conditions in which these specifications were established validate the conclusions of this work for a chain topology only. Further investigations are indeed needed to determine whether these results can be extended to more complex scenarios. We show in chapter 7 that this approach plays not significant role for short-range multihop networks of up to approximately ten hops since a fixed CWL of three packets is enough for providing acceptable performance in such cases.

The Link RED and Adaptive Pacing to Improve TCP in Multihop Networks

This work [FZL$^+$03] first investigates optimal settings for TCP over 802.11 with regards to TCP congestion window limit in order to improve bandwidth utilization. It then proposes two techniques to improve TCP performance in multihop networks: the link RED (LRED) and the adaptive pacing. The former seeks to react quickly to link overload, while the latter seeks to improve spatial reuse. The combination of both techniques improved TCP throughput as much as 30%.

By simulation and evaluation, this work shows that in multihop wireless networks TCP should not use a large *cwnd* size as in wired networks for getting high end-to-end throughput. Large *cwnd* size render degradation in spatial reuse and consequently in TCP throughput. It is observed that there is an optimal *cwnd* with which TCP achieves best throughput. For a chain topology *cwnd* should be set to approximately $h/4$, where h is the number of hops in the chain topology. This setting ensures best spatial reuse property. In addition, this work investigated the packet drop behavior in multihop wireless networks. It found that most of drops in such networks is due to the link layer contentions rather than congestion.

The LRED algorithm aims at improving wireless link's drop probability by monitoring the link drops of the 802.11 MAC protocol at each node. In the conventional RED algorithm, if the queue size exceeds a minimal threshold the drop curve grows linearly as more data arrive. LRED does not use the queue size for specifying its minimal threshold but the average number of recent retransmission attempts by the node. Thus, if this number exceeds a minimal threshold a dropping/marking probability is calculated. In this calculation, the minimal threshold as well as the number of recent retransmission attempts are taken into account and the result is limited to a maximum value. This mechanism is claimed to be proper to improve TCP throughput, to provide a fast network overload signaling to TCP, and also to improve fairness among the competing flows.

The adaptive pacing algorithm is also implemented at the link layer. Its purpose is to enhance spatial reuse by balancing traffic distribution in the network and reinforcing better coordination on forwarding nodes along the data path. In fact, this mechanism aims to prevent the exposed node problem addressed in section 5.3.1. The motivation for this mechanism comes from the observation that in 802.11 a node defers its transmission only to prevent collision with its immediate downstream node, but that is not enough to avoid that nodes two hops away from the sender induce collisions. Then adaptive pacing imposes, if needed, a additional delay for the nodes transmission.

The adaptive pacing mechanism works combined with the LRED algorithm as follows. If LRED first determines that its minimal threshold above has not been reached; it sets its drop probability to zero, computes its backoff period as usual, and starts the adaptive pacing. In the next invocations under the same condition, the backoff time is increased by the pacing mechanism. The additional delay to the backoff time is equivalent to the transmission time of the previous data packet. This is shown to provide a better coordination among the involved nodes because the higher deferral time mitigates collisions.

The evaluation results showed that the combined LRED+pacing can boost TCP throughput as much as 30% in a chain topology. In addition, this changes seem to lead TCP to stabilize at a window size near the optimal value. The longer the chain the better the improvements due to the better spatial reuse provided by these changes. In grid topologies, less improvements are observed but good fairness are obtained. Results from simulations match experimental evaluations.

This is certainly an interesting proposal but it requires changes at each node in the network. Furthermore, in the evaluations the routing algorithms were disabled for simplicity. From our experience, we believe that routing protocols can play an important role even in static topologies. This occurs because whenever a link retransmission fails completely, the routing strategy is invoked to generally start a route discovery. The time involved in such route discovery depends on the features of the routing strategy in place and may disrupt TCP performance significantly. In all our evaluations in this thesis we consider routing protocols enabled for ensuring a more realistic simulation setup.

TCP DOOR

This approach [WZ02] imposes changes to TCP code but does not require intermediate nodes cooperation. It focuses on the idea that out-of-order (OOO) packets can happen frequently in ad hoc networks as a result of nodes mobility, and it might be enough to indicate link failure inside the network. In this way, TCP OOO detects OOO events and responds accordingly as explained below.

Based on the fact that not only data packets but also ACK packets can experience OOO deliveries, this algorithm implements a precise detection of such deliveries at both entities: sender and receiver. For this, additional ordering information is used in both kinds of packets, which are conveyed as TCP options being one-byte option for ACKs and two-byte option for data packets. So, for every packet sent the sender increments its own stream sequence number inside the two-byte option regardless whether it is a retransmission or not (standard TCP does not increment sequence number of retransmitted packets).

This allows the receiver to precisely detect OOO data packets and notify the sender via a specific bit into ACK packet. Additionally, because all ACKs associated with a given missing data packet have identical contents, the receiver increments its own ack stream sequence number inside the one-byte option for every retransmitted ACK, so that the sender can distinguish the exact order of every (retransmitted or not) sent packet. Therefore, the explained mechanisms provide the sender with reliable information about the order of the packet stream in both directions, allowing TCP sender to act accordingly.

After detecting OOO events, TCP sender can respond with two mechanisms: temporarily disabling congestion control, and instant recovery during congestion avoidance. In the former, TCP sender keeps its state variables constant for a while (T1) after the OOO detection. The rationale here is that such condition might be short (route change) not justifying the invocation of the congestion avoidance mechanism. In the latter, when an OOO condition is detected TCP sender checks if the congestion control mechanism has been invoked in the recent past (T2). If so, the connection state prior to the congestion control invocation is restored, since such an invocation may have been caused by temporarily disruption instead of by congestion itself.

In terms of evaluation, different scenarios combining all the mechanisms above mentioned were simulated. Also, the effects of the route cache property of DSR routing protocol on TCP DOOR performance were considered. The main results showed that only sender detection mechanism (ACK OOO detection) should be enough for good performance. Both response mechanisms showed to be important, but the instant recovery during congestion avoidance performed better than temporarily disabling congestion control. The DSR route cache impaired the results.

In general TCP DOOR improved TCP performance significantly, 50% on average. Therefore, the overall published results were positive. Nevertheless, the assumption that OOO packets are exclusive results of route disturbance deserves much more careful analysis. As stated by the authors themselves, multipath routing algorithms (like TORA) can also induce OOO packets that are not necessarily related to route failures. Besides, diverse other factors can cause path asymmetry inducing OOO events as well.

Fixed RTO

This scheme [DB01] relies on the idea that routing failure recovery should be accomplished in a fast fashion by the routing algorithm. As a result, any disconnection should be treated as a transitory period which does not justify the regular exponential backoff mechanism being invoked, as this can cause unnecessarily long recovery delay. So, it disables such a mechanism when two successive retransmissions due to timeout happen, assuming it indicates route failure. By doing so, it allows the TCP sender to retransmit at regular intervals instead of at increasingly exponential ones.

In fact, the TCP sender doubles the RTO once and if the missing packet does not arrive before the second RTO expires, the packet is retransmitted again and again but the RTO is no longer increased. It remains fixed until the route is recovered and the retransmitted packet is acknowledged. In [DB01] the authors evaluated this proposal considering different routing protocols as well as the TCP selective and delayed acknowledgments options. They report that significant enhancement were achieved using fixed-RTO with on-demand algorithms, and only marginal improvements were noticed regarding the TCP options mentioned. Nevertheless, as stated by the authors themselves, this proposal is limited to wireless networks only, which makes it somewhat discouraging as interoperation with wired networks seems to be really necessary in future implementations.

5.7.4 TCP Evaluations

Estimating the Ad Hoc Horizon for TCP over IEEE 802.11 Networks

This work [TO04] has conducted an extensive simulation evaluation to determine the feasibility of TCP over ad hoc networks. It shows that TCP over the IEEE 802.11 standard faces a limit of 2 to 3 hops or 15 nodes to provide useful performance. This limit is termed the “ad hoc horizon”. Beyond this limit TCP performance is shown to degrade to unacceptable levels.

This work challenges some past work that have evaluated scenarios containing either very large number of nodes or very long networks in terms of hops. In contrast to these previous investigations, [TO04] considered worst cased scenarios results to show that an unlucky user may face substantial performance problems in his/her TCP connection.

Several topologies called “Beam star” were evaluated in such a way that the number of both hops and nodes could easily be adjusted and their effect evaluated. The evaluations showed that TCP connections may experience up to 30% stall-ratio when using AODV routing protocol. By disabling link layer feedback for AODV the stall-ratio rised to 60%. These degradation occurred for scenarios exceeding the ad hoc horizon. TCP fairness was another metric evaluated. These results showed that after two hops, TCP becomes quite unfair concerning the bandwidth distribution among the competing TCP connections.

A proactive routing protocol was also considered, and the results showed that it degraded TCP ever more. Short HTTP flows competing with a large FTP flow were also simulated to provide a more realistic evaluation. Similarly to the case where only FTP flows were in place, the results exhibited high degradation after 2 to 3 hops for the HTTP flows. The evaluations above did consider mobility. To assess the fluctuations of mobility, a packet drop rate of 2% was introduced in the scenario where FTP and HTTP flows shared the medium. The results for a 3-hop scenario was even worse to the HTTP flows.

The main outcome of these results is that the ad hoc horizon is really low, being in the 2 to 3 hops range and extends to a dozen nodes. Cross layer coordination could improve, but most likely not more twofold. So, routing protocols should be optimized to their small-scale features rather than large-scale ones.

The contributions in this thesis are in line with this work in that we also argue that only short-range ad hoc networks are feasible today. The ad hoc horizon is limited by the hidden node problem discussed in section 5.3.1. In addition, we confirm that TCP can be fined tuned to interact more smoothly with the IEEE 802.11 standard. In chapter 7 we introduce our smart acknowledgment strategy to mitigate such problem by minimizing collisions.

Investigating the Performance of TCP in Mobile Ad Hoc Networks

The evaluations in this work [PH02] first investigated the effect of path length on TCP throughput, the fairness among concurrent TCP flows, and the impact of two on-demand routing protocols (AODV and DSR) on end-to-end TCP throughput. Then a quantitative analysis on the effects and interactions of mobility and routing on TCP throughput was performed. In particular, the following factors were evaluated: routing, node speed, and node pause time. The purpose of this quantitative analysis is to highlight that it is important to know not only which factors impact the network performance but which one has the greatest impact. In addition, it is also helpful to know how much of influence a given factor suffers from the others. This can assist protocol design and performance tradeoffs. All evaluations were carried out by simulation using the Glomosim network simulator.

The evaluation results show that: a) TCP throughput decreases significantly as the path length increases, b) the fairness of the network is very poor among concurrent TCP flows, c) the routing protocol DSR performed better than AODV. The low performance for both long paths and concurrent flows is assigned to the inefficiency of the MAC layer. Yet, The overhead associated to AODV is suggested as the reason for its low performance. The quantitative analysis finds that among the three factors evaluated, routing protocol has the greatest impact on TCP performance. In addition, there is a strong interaction between node speed with pause-time and routing protocol because both node speed and pause-time determine the number of topology changes over a time interval.

This is an insightful work toward better understanding TCP in mobile ad hoc networks, and so it has certainly its merit. Nevertheless, the conclusions are based on simulations conducted for single chain topologies only. Different scenarios need to be included in these evaluations to render the results more general. In this chapter and in chapter 7 we elaborate on the problems of TCP in such networks, and in general our findings are in accordance with the results of this work.

5.8 Summary

This chapter discussed and evaluated key problems faced by TCP over multihop wireless networks, and presented the main related work. TCP is expected to be deployed in ad hoc networks and possibly also in sensor networks due to interoperability reasons. This chapter focused on multihop ad hoc networks as these are the primary target of this thesis.

Ad hoc networks rely on the IEEE 802.11 standard to control the wireless medium access. This standard is, however, not very efficient in relatively long-range multihop networks largely because of the hidden and exposed node problems, as well as the capture effect. The longer the network the more degradation is experienced by the end-to-end connection. Concerning Routing protocols for ad hoc networks, AODV and DSR proposals seem to be the most promising ones to be deployed in future ad hoc networks. These schemes impact TCP performance in distinct ways. The choice of a routing protocols appears to depend on tradeoffs between delay and transmission/traffic overhead.

TCP performance is affected by the inappropriateness of the original TCP mechanisms for wireless networks. By associating any packet loss to congestion, TCP may slow down too often and waste precious bandwidth when those losses are a result of channel errors. The next chapter addresses this issue further, presenting our fuzzy logic based mechanism for packet loss discrimination in multihop networks.

This chapter also showed that using large $cwnd$ may be undesirable for tcp as this may induce higher losses because of the limited spatial reuse imposed by the MAC's hidden node problem. Having an ACK for each sent packet leads to bandwidth wastage as well. Chapter 7 presents our dynamic acknowledgment strategy as a smart mechanism to manage TCP ACK generation in multihop networks.

Finally, the related work presented in this chapter made it clear that many issues remain to be addressed in this challenging networks. In general, the proposed solutions are very much focussed to solve a single problem, while the big picture is mostly not addressed.

Chapter 6

Packet Loss Discrimination Using Fuzzy Logic

6.1 Introduction

The problems faced by a regular TCP over multihop wireless networks requires adjustments in the fundamental principles of the protocol. As explained in chapter 3, multihop networks are prone to much higher bit error rates than their wired counterparts. As a consequence, the TCP congestion control mechanisms lack robustness in these lossy environments. To address this problem, we propose a technique for assisting the TCP error detection mechanism to distinguish between congestion and medium induced errors. Accurate information about the actual cause of a dropped packet might be useful for a TCP sender to avoid waste of bandwidth and not to induce congestion collapse. Our proposal relies on Round-Trip Time (RTT) measurements to estimate the internal network state through a fuzzy logic inference system. This chapter investigates RTT patterns in the related environments, introduces our fuzzy logic based mechanism, and shows the evaluation results.

6.2 Packet Loss Discrimination

TCP has been successful in the global Internet in preventing network collapse since the early 1980s. The key point for this success lies in its adaptive congestion control that works on an end-to-end basis, being independent of underlying protocols in the network. As described in section 2.3.3, a TCP sender detects dropped packets by either the retransmit timeout or the fast retransmit mechanisms. This detection procedure is generally called *error detection*. After perceiving an error, i.e., a dropped packet, the congestion control is said to enter an *error recovery* procedure. The error recovery consists of two actions: retransmission of the lost packet and reduction of the congestion window [TM02].

In summary, TCP reacts to perceived losses by reducing the sender transmission rate because the error is assumed to be a congestion result. To associate congestion to dropped packet is efficient in wired networks where the typical bit error rate due to the medium is negligible. Thus, to slow down upon a packet drop is the correct action in wired networks because that alleviates congestion within the network.

In ad hoc networks, however, packet loss can be caused not only by congestion but also by the lossy medium as well as by mobility induced link interruptions. So, the basic assumption upon which TCP was designed is not always satisfied in these multihop environments. Because of that, the TCP error detection mechanism may arise deficiencies when required to work in such complex environments. The problem is that TCP should not necessarily reduce its congestion window in half when reacting to a dropped packet. Rather, the TCP error recovery mechanism should be informed by the error detection strategy about the actual cause of any packet loss so the recovery mechanism can take the most appropriate action.

The explanations above suggest that a robust TCP sender should have dedicate actions for each of the constraints found in ad hoc networks, namely congestion, channel error, and link interruption. Considering end-to-end approaches, link interruptions will always be detected by the retransmission timer. The exact protocol response will depend on the current condition inside the network. This means that the sender has to be permanently aware of what is going on within the network so that it can take proper action upon loss detection. Hence, the TCP error detection mechanism should distinguish between congestion and medium induced losses. This task is generally termed packet loss discrimination, and various investigations on this subject have been carried out in recent years, as discussed in section 5.7.2.

The novelty of our proposal presented in this chapter regards the fact that it infers the internal network state on an end-to-end basis using an intelligent algorithm based on fuzzy logic [OB04a,OB04c]. This is challenging to design but easy to deploy as the changes are restricted to the end nodes. The designed fuzzy logic engine receives RTT measurements over predefined intervals and make inference on the measurements to decide whether the network is facing congestion or channel error effects.

RTT mean and variance are shown to have distinct behavior whether the network is under congestion or medium error constraints. However, the noisy nature of the RTT measurements requires an elaborate algorithm for efficient discrimination. By using fuzzy logic, the continuous and imprecise behavior of the information can be handled without the necessity of arbitrary rigid boundaries. Besides, it is computationally inexpensive. These features render fuzzy logic quite suitable for making inference on RTT measurements where imprecision and uncertainties are effectively present and the processing requirements (at the end nodes) must be as low as possible.

6.3 Round-Trip Time Patterns

This section investigates RTT behavior in typical ad hoc networks. The purpose of this investigation is to determine how RTT measurements may be useful to packet loss discrimination in these networks. In ad hoc networks, RTT values reflect the impact of the following distinct factors: congestion, wireless channel induced errors, changes in the number of hops crossed by the connection between sender and receiver, packet size, and lower layer protocol mechanisms such as MAC layer retransmission strategy [XS01b, OB02].

We assume in this work that the packet size of the connection and the lower layer protocols are fixed. Both assumptions should, however, not be too restrictive for future ad hoc networks. Fixed packet size can achieved through proper coordination between application and the transport protocol, and so their use should not be costly. Regarding the lower layer protocols, the IEEE 802.11 MAC protocol [CWKS97, IEE99, CGL00] is already standardized and both AODV [PR99, PBRD03] and DSR [JM96, JMH04] routing protocols are the most prominent frameworks to be standardized in the near future. So, it is reasonable to consider that these two routing protocols will be effectively present in future ad hoc networks. This is important because both AODV and DSR can ensure symmetric links, which is also a requirement of our initial approach proposed here.

Therefore, taking the two assumptions above into consideration (fixed packet size and conventional lower layer protocols such as 802.11 and AODV), RTT values might be really useful for distinguishing the effect of the other two factors, namely congestion and bit error.

We have shown in [OB03] that both the growth in the number of hops between sender and receiver and congestion conditions may impose similar behaviors to RTT measurements. Thus, sender and receiver should exchange information to detect the exact number of hops in place. We do not address this issue further here, although it can be easily implemented using the Time To Live field in the IP header [Ste94]. Actually, we focus here on the discrimination of packet losses by congestion from packet losses by channel error, having a fixed number of hops end-to-end.

6.3.1 Simulation Environment

Fig. 6.2 shows the result of two simulation runs using the ns-2 simulator [EHH$^+$00, EFF$^+$00] for a chain of nodes topology, as in Fig. 6.1. AODV and IEEE 802.11 are the routing and MAC layer protocols in place, respectively. The packet size is set to 1000 bytes, and both the main flow and the background flow are generated by FTP applications over TCP Reno version. A uniform distribution function is used as wireless channel error pattern for generating the different levels of BERs, and the wireless bandwidth is actually 1 Mbps. We use hereafter Packet Error Rate (PER) instead of BER to represent the intensity of channel error constraint because PER is the metric in fact simulated.

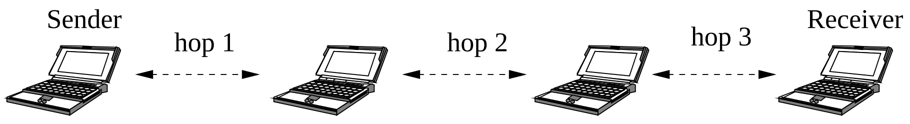
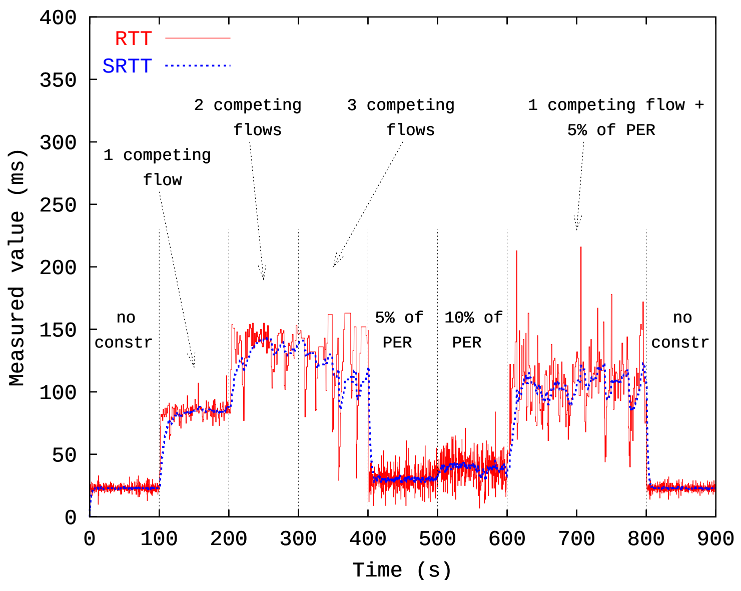


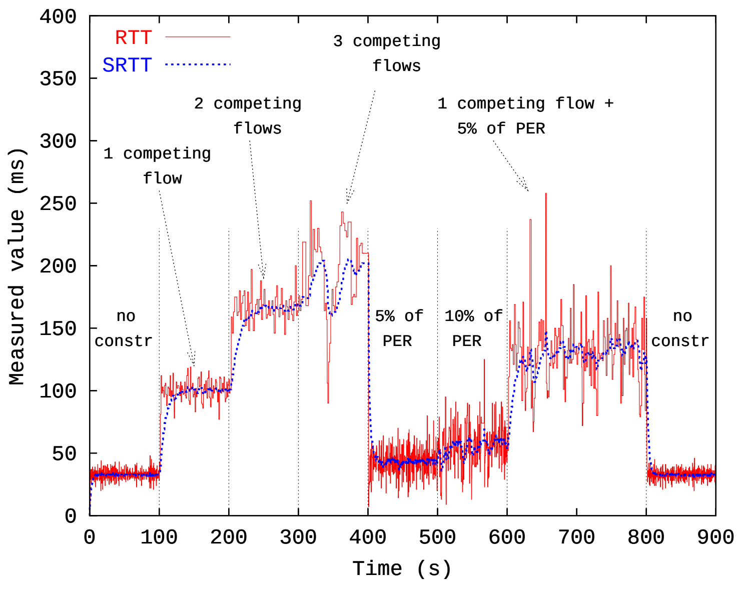
Figure 6.1: Chain topology with 3 hops (3-hop scenario)

6.3.2 RTT Measurements under Heterogeneous Conditions

Fig. 6.2(a) depicts RTT and SRTT (Smoothed or average RTT) measurements on a typical 2-hop ad hoc network under varying levels of congestion and distinct packet error



(a) 2-hop scenario



(b) 3-hop scenario

Figure 6.2: RTT characteristic under congestion and wireless losses

rates, starting at 100 seconds. Between 100 and 400 seconds there are three stages of congestion, i.e., 1 to 3 competing flows. Between 400 and 600 seconds, there is no congestion, but only two levels of packet error rate, i.e., 5% and 10%. At last, between 600 and 800 seconds there is only one competing flow with a packet error rate of 5%. In the last 100 seconds, the channel is free of any constraint again. Fig. 6.2 illustrates the same conditions but for a 3-hop scenario.

Fig. 6.2 shows clearly that under operative congestion conditions (having at least a minimum flow of ACKs) and without mobility, RTT mean values (roughly given by SRTT) should suffice to indicate congestion inside the network. The channel error constraint increases RTT as well, but at a far lower degree. The only possibility of mistake, although rare, would be a very high level of packet error rate being misdetected as congestion, since high PERs could increase the RTTs considerably. We call this possibility *critical overlap* for easing reference to it below.

We observe that the possibility above is quite low because at such a high level of PERs practically no packet gets through. Besides, typical wireless environments do not have so lossy channels in steady state conditions. If a certain scenario does face such level of losses, then the best action to take appears to be a reduction of the transmission rate. Thus, in such cases a misdetection would not cause relevant problems.

Nonetheless, to make the distinction of congestion from wireless induced losses even more robust, RTT variance can be used to address the remote possibility mentioned above, as indicated in table 6.1. The values in this table are computed over each interval using (6.1) and (6.2) presented in section 6.5.1. Table 6.1 shows how much the RTT variance increases under channel error conditions. It is worth to compare the case in which there is just one competing flow (100-200) with the instant of the highest PER (500-600). One can notice here that for both scenarios (2-hop and 3-hop) the channel error constraint induces higher RTT variance, namely 215 and 537 against 97 and 135 for both scenarios, respectively. For lower congestion level, which could be achieved with other forms of competing traffic, the discrimination would be even better since the variance under congestion would be smaller.

Table 6.1: RTT Mean and Variance

	2-hop		3-hop	
Interval	mean	variance	mean	variance
0-100 (1 flow)	22	7	32	25
100-200 (2 flows)	82	97	99	135
200-300 (3 flows)	136	304	162	246
300-400 (4 flows)	128	820	189	980
400-500 (5% PER)	32	176	47	646
500-600 (10% PER)	47	215	60	537
600-800 (2 flows + 5% PER)	115	840	136	863

In the last part of the simulation under constraint (600-800) depicted in Figs. 6.2(a) and 6.2(b), there is only one competing flow combined with a packet error rate of 5%. It can be seen that both the RTT mean (roughly given by SRTT) and variance grow (higher magnitude of the oscillations). Table 6.1 supports this observation as well. The key point here is to note that the congestion detection has priority over the channel error detection, as a TCP sender has to slow down anyway under congestion regardless of simultaneous channel error losses. In other words, if congestion is perceived, then the detection algorithm can ignore the channel error detection.

6.3.3 RTT Measurements under the Critical Overlap

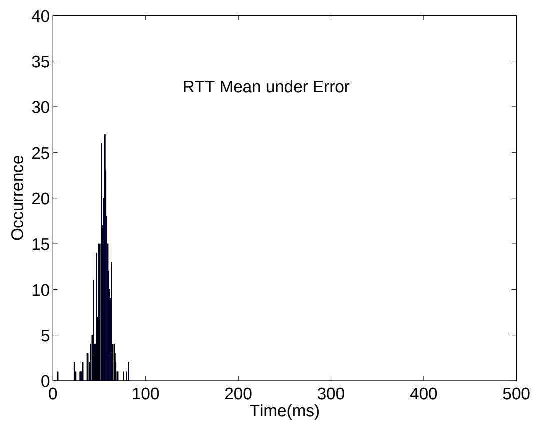
In order to identify the *critical overlap* mentioned above, we conducted further simulations focusing on the conditions of interest. We considered only the 3-hop scenario since the procedure is the same for all other cases. As discussed in the prior section, it might happen that a low congestion condition is misdetected as a lossy channel. Because of that, we took the case in which the main connection is facing only one competing flow and the one in which only a PER of 10% constraints the medium. By doing so, we can compare the lowest RTT mean for congestion against the highest RTT mean for the medium induced errors. We also ran longer runs of 500 seconds in each condition for better pattern definition.

For the sake of clarity, the simulation results, after applying (6.1) and (6.2), are depicted as histograms in Figs. 6.3 and 6.4. From Fig. 6.3, one can observe that even though the level of PER is relatively high, it is still possible to distinguish both conditions (error and congestion) by simply comparing their RTT mean values. Fig. 6.3(a) shows that the experienced RTT mean values for the high PER are concentrated around 50 ms. Likewise, Fig. 6.3(b) shows that RTT variance values for the low congestion level is centered around 100 ms.

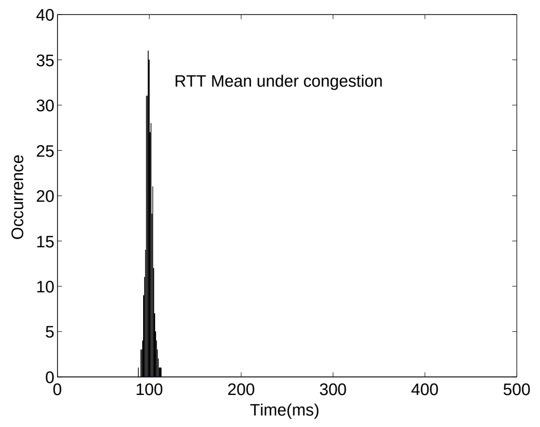
In Fig. 6.4, we have the variance distribution for a channel without any constraint, a channel facing congestion, and a channel facing channel error. Comparing Fig. 6.4(b) with 6.4(c), one can see that in this measurements the channel error constraint imposes indeed higher spread to the RTT variance. Nevertheless, there is an overlapping area roughly in the range (50,200), which has to be taken into consideration in the evaluation of these measurements. Section 6.6.1 shows how the fuzzy logic engine can be set in such cases.

6.3.4 Discussions

The evaluations above make it clear that we need a sort of pattern recognition algorithm for monitoring the RTT values and making decisions about the internal state of the network. This is not a trivial task though, given the uncertainties inherent in such measurements. Additionally, the decisions have to be made in a quick fashion if improved end-to-end performance is to achieved. Hence, we explain in the sections below how fuzzy logic concepts may be used for distinguishing between congestion and channel induced losses, taking RTT mean and variance values as input variables.



(a) Channel error (PER=10%)

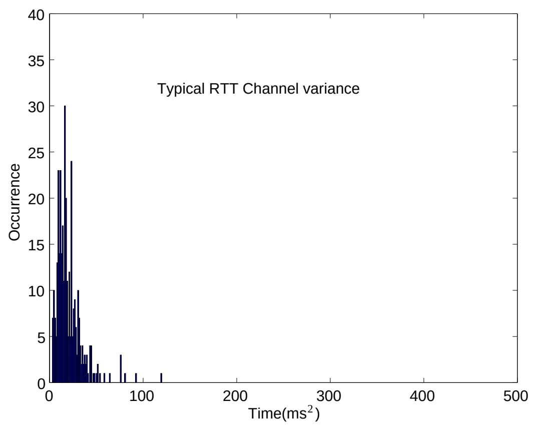


(b) Congestion

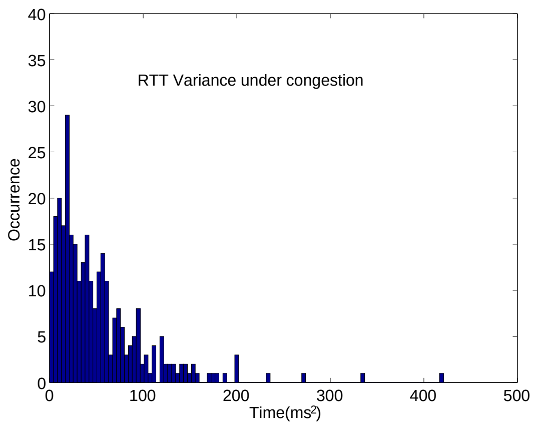
Figure 6.3: Distribution of RTT mean in a 3-hop multihop network

6.4 Fuzzy-based Error Detection

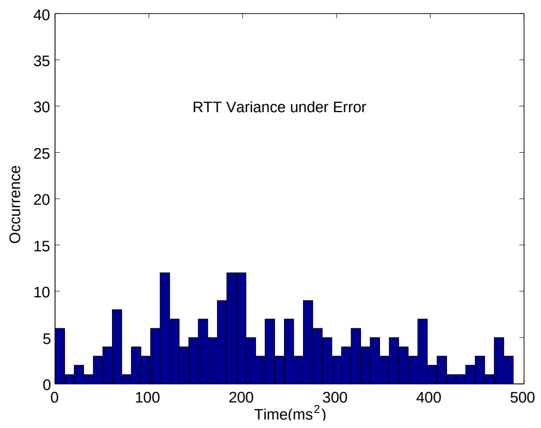
This section outlines the main features of our proposal for an improved error detection strategy based on fuzzy logic. The proposed Fuzzy-based Error Detection Mechanism (FEDM) only relies on fuzzy logic for discriminating losses due to congestion from losses by channel errors. Losses by link interruptions are identified differently, as shown in [OB03], since no ACK is received in such conditions. In fact, losses by link interruptions lead the retransmission timer at the sender to time out.



(a) No constraints



(b) One competing flow



(c) 10% of PER

Figure 6.4: Distribution of RTT variance in a 3-hop multihop network

After timing out, the sender reduces its transmission rate to a minimum, which is not completely inefficient in these cases. That is, in these situations the link has been presumably interrupted for some time, and so the time the sender takes to recover its transmission rate has relatively low relevance.

The designed fuzzy engine has to be adjusted in accordance with the number of hops traversed by the connection, because the RTT measurements are obviously dependent on the number of hops being used. Because of that, the FEDM needs an additional mechanism to monitor permanently the number of hops end-to-end.

Fig. 6.5 depicts the general architecture of our approach. The NH (Number of Hops) and RR (RTT increase Rate) blocks are needed to detect the number of hops in the end-to-end connection and steep increases in the RTT measurements, respectively. C, U and B represent the signaling flags Congestion, Uncertain, and Bit error, respectively. The purpose of each of these elements are explained below.

The NH block keeps track of the number of hops in the TCP session so that the Improved Error Detector (IED) can set the Fuzzy engine parameters properly. This is needed because such parameters change according to the number of hops in the end-to-end connection, i.e., the more hops the higher the range of the delays. This is a weakness of the model that requires the parameters for the fuzzy engine to be determined in advance. Thus, the parameters for every number of hops have to be determined in advance by simulation or experiment.

Having distinct settings for every connection length, in terms of hops, solves the problem that arises by the similarity of RTT behaviors (steep increase) when either congestion starts or the number of hops between the end nodes increase [OB03]. A sort of learning, adaptive mechanism could be used here to adjust the fuzzy logic settings more appropriately. We do not investigate further this issue because our goal here is focused on the evaluation of the fuzzy logic concept suitability for packet loss discrimination rather than on designing an optimized system. This is left for future work.

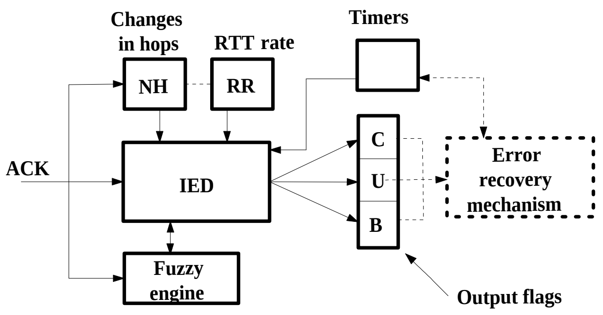


Figure 6.5: Fuzzy-based error detection mechanism

We propose to use the Time To Live (TTL) field within the IP header for identifying the mentioned number of hops, mainly because it is standard in current implementations.

This demands a simple interoperation between transport and network layer protocols and the use of either an IP or TCP option inside the packet header. Upon packet receipts, the NH block may use compute the exact number of hops crossed by the connection by comparing the current TTL with the TTL sent.

The RR block monitors the rate at which the RTT changes. The idea here is to set a threshold for triggering detections of persistently high rate increase in the RTTs faster than the fuzzy engine might do, thereby avoiding network collapse. Specifically, this mechanism detects a predefined number n of subsequent increases in RTT, in which every increment is larger than a given degree α relative to the RTT value at the beginning of the congestion.

Fig. 6.6 illustrates a typical RTT pattern when a congestion-free channel starts becoming congested, exactly as shown in the simulation results in Fig. 6.2. From simulations we noticed that when congestion starts, RTT rapidly and persistently increases. Hence, monitoring the accumulated increase α in RTT in a regular basis can be efficient in detecting congestion very quickly.

Based on our simulation evaluations, as shown in Fig. 6.2, n = 2 and α = 20% perform well with an RTT granularity of 100 ms (minimum interval between successive RTT samples). When triggered, this mechanism notifies the IED which can set the congestion flag (C) if no extra hop is detected simultaneously. NH and RR blocks work together as the RTT persistently high rate growth has to be detected only when no increase in hops is identified.

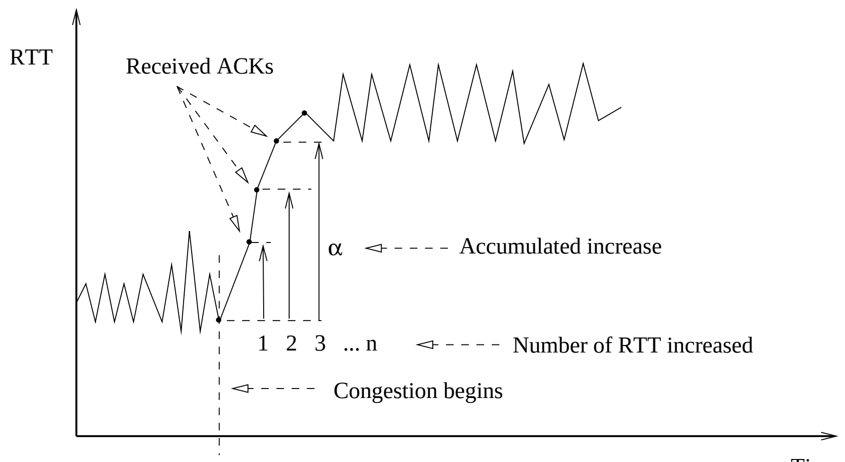


Figure 6.6: Detection of steep increase in RTT

The three blocks around the IED provide it with enough information about the state of the wireless channel, so that it may take decisions more accurately. The outcome of the decision is set in the output flags. The flag Uncertain (U) refers to the undefined output of the fuzzy engine which may not always provide a conclusive output, as discussed below.

Losses by link interruptions will be detected by the timeout timer along with the state flags. Timeout with Congestion (C) call for retransmission and slowdown, while timeout combined with bit error (B) ask for retransmission only. Timeout with uncertain (U) should also call for retransmission and slowdown considering a conservative approach. Observe that the fast retransmit mechanism is also in place for speeding up retransmissions before timeouts may occur. The exact actions to be taken depend on the error recovery mechanism strategy in use. In any case, whenever an action is taken under the uncertain (U) condition, the current TCP state variables such as congestion window and slow start threshold should be saved for possible recovery upon effective condition detection. In addition, there is always a low risk that the fuzzy engine provides a wrong result. Section 6.6.5 discusses the cost of a misdetection.

The pseudo code of the IED is depicted in table 6.2. Whenever an ACK is received, the NH variable receives information about the number of hops crossed by the packet. If it has changed since the last evaluation, the fuzzy engine parameters are updated. These parameters will be addressed in section 6.5 below. The output of the fuzzy engine is assigned to the FR variable, and the RR block output is saved into the RR variable. The IED then checks the variables and set the flags accordingly. This is a very general description of our proposal for an improved error detection mechanism. As mentioned above, The crucial element of this concept is the packet loss discrimination, and that is the focus of this chapter in the sections below.

Table 6.2: IED pseudo code

```
#Upon an incoming ACK the following actions take place

NH ← get(NH_report);              # true: number of hops remained constant, false:otherwise
if (NH == false) {
    update(fuzzy_engine);          # fuzzy parameters depend on nr. of hops
}
FR ← get(fuzzy_engine_report);    # possible values: Congestion, Uncertain and Error
RR ← get(RR_report);              # true: abrupt increase in RTT value, false:otherwise
if (FR == Congestion or (RR == true and NH == true)){
    C ← 1;                         # channel is facing congestion
    U ← 0;
    B ← 0;
} elsif(FR == Uncertain){
    C ← 0;
    U ← 1;                          # the fuzzy engine could not determine a decisive output
    B ← 0;
} else {
    C ← 0;
    U ← 0;
    B ← 1;                          # channel is facing errors
}
```

6.5 A Fuzzy Logic Engine for Loss Discrimination

The behavior of the RTT measurements presented in section 6.3 suggests that an intelligent mechanism may be able to distinguish congestion from channel error induced losses inside the network. In particular, such a mechanism should be simple and flexible in terms of easy adaptation to this highly dynamic scenario, and also computationally inexpensive. Thus, a fuzzy logic engine has been designed for handling RTT measurements toward effective packet loss discrimination. The fuzzy engine design is similar to the work presented in [CM01] where packet delay behavior is used for detecting wireless links.

6.5.1 Fuzzy Engine Input

Section 6.3 showed that RTT mean and variance values can be useful to distinguish losses by the medium from congestion induced losses. So, the input variables of the fuzzy engine are defined as the RTT mean t in (6.1) and the RTT variance δ_t in (6.2), with $i = 1, 2, ..., n$ ($n = maxSamples$). The term $n = maxSamples$ refers to the maximum number of RTT samples used in the computations shown in (6.1) and (6.2). The universe of discourse of the input variables (RTT) is specified as $[0, Tmax]$. We limit RTTs to a certain $Tmax$ value because in normal operation the maximum value assumed by RTTs is always limited by either the timeout timer or the fast retransmit mechanism. Hence, it is reasonable to assume that RTTs are limited to $[0, Tmax]$ without loosing generality.

The computations in (6.1) and (6.2) imply that the universe of discourse of the fuzzy input variables t and δ_t should be $[0, Tmax]$ for the reasons mentioned above. As described in section 4.9, a fuzzifier has to map crisp values into membership functions in the fuzzy space. Thus, the crisp values RTT mean and variance, t and δ_t, have to be mapped into membership functions.

$$t = \frac{1}{n}\sum_{i=1}^{n} t_i \tag{6.1}$$

$$\delta_t = \frac{1}{n}\sum_{i=1}^{n} (t_i - t)^2 \tag{6.2}$$

For computing simplicity and better control of the spread of the curves, as stressed in [CM01], we use in this thesis Gaussian membership functions for the input memberships (Fig. 4.3). As described in section 4.5, a Gaussian curve is specified by two parameters, a constant k from the universe of discourse defining the center of the curve, and another single value γ specifying the width of the curve.

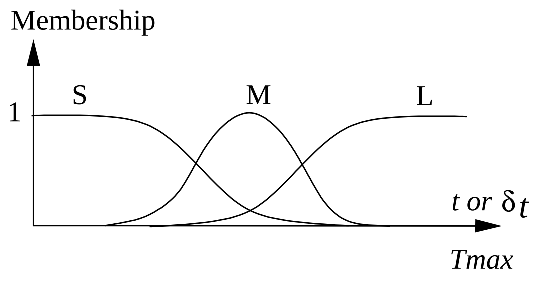
The values of k and γ are set on the basis of the measurements to be evaluated, as explained below. The universe of the fuzzy input variables t and δ_t are divided into three fuzzy sets as shown in Fig. 6.7(a). In Matlab [Mat02], which is the tool used for the engine evaluation, both the s-curve and the z-curve may be generated with approximation by using the Gaussian curve.

Observe that here we have one more fuzzy set than in the example of Fig. 4.5. The fuzzy linguistic variables used are S (Small), M (Medium) and L (Large). Hence, the input values have to be mapped to these fuzzy sets, as illustrated in the example of Fig. 4.5. Note that the more fuzzy sets, the more accurate may be the input discrimination.

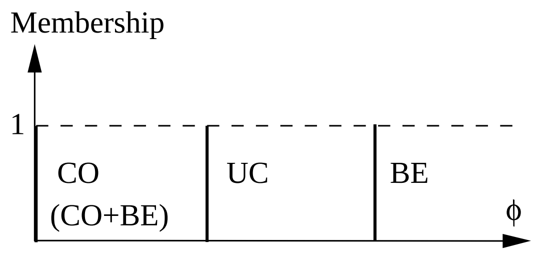
The number of fuzzy rules and complexity increase, though.

6.5.2 Fuzzy Engine Output

The output of each fuzzy rule in our fuzzy engine is assigned to a corresponding output fuzzy set. The output of the fuzzy engine might take several distinct forms. For instance, there might be two fuzzy output sets, one for bit error detection and another for congestion detection. Another possibility should be to have both detections in just a single fuzzy output. The universe of the fuzzy output has to be set accordingly. In this thesis, we use just a single fuzzy output ϕ, as illustrated in Fig. 6.7(b).



(a) Input.



(b) Output.

Figure 6.7: Fuzzy membership functions.

Hence, the output of the fuzzy engine is set as the discrimination of congestion from bit error effects, and the universe of the fuzzy output variable is split into three singleton (single value) fuzzy sets as depicted in Fig. 6.7(b). This means that each rule output will be placed at either 0 or 0.5 or 1 in the universe of the single output ϕ.

The corresponding fuzzy linguistic variables are CO (Congestion), UC (Uncertain) and BE (Bit Error). As congestion has priority over bit error, the CO variable covers also conditions of simultaneous congestion and bit error constraints. The specific fuzzy rules are shown in table 6.3.

Table 6.3 shows, for instance, that small (S) ”RTT mean” and large (L) ”RTT variance” indicate clearly that the measured flow is facing bit error constraint (BE). Likewise, large (L) ”RTT mean” suggest congestion (CO), regardless of the RTT variance value. The other rules are similarly set up according to the RTT evaluation discussed in section 6.3. The fuzzy reasoning is done on the basis of the min-max method explained in section 4.9, and the centroid is the deffuzifier in place, computed as shown in section 4.9.

Table 6.3: Fuzzy Rules Output (ϕ)

var. \ mean	**S**	**M**	**L**
S	BE	CO	CO
M	BE	UN	CO
L	BE	BE	CO

6.6 Performance Evaluations

This section presents evaluations on three aspects of the proposed FEDM: The ability of the fuzzy engine to perform accurate discrimination, the speed of the FEDM to detect abrupt changes in the network condition, and the effectiveness of not slowing down a TCP when losses are caused by the lossy medium instead of by congestion. All evaluations here are based on a 3-hop scenario only since the procedure is similar to the other cases.

6.6.1 Fuzzy Engine Parameters Configuration

The fuzzy engine parameters are set to met the requirements of the critical overlap mentioned in section 6.3.3. Since Fig. 6.3(a) shows that the experienced RTT mean values for the high PER (10%) are concentrated around 50 ms, the Gaussian membership function associated to the fuzzy linguistics variable M in Fig. 6.7(a) should have its center at 50. Because of that, we set the k parameter of the Gaussian curves for the linguistics variables S, M, and L in Fig. 6.7(a) to 10, 50, and 90, respectively. Yet, the width γ of each curve is set to 20.

Fig. 6.4 exhibits an overlapping area roughly in the range (50,200), which has to be taken into account for setting the fuzzy membership functions associated with the RTT variance. In reality, the range of this overlapping area would not be too wide if we had a more precise comparison by simulating a lower level of congestion. The smaller the congestion level, the closer the histogram of Fig. 6.4(c) from the one in Fig. 6.4(a). The Gaussian curves for RTT variance are set analogously to the way done above for the membership functions relative the RTT mean. We set the k parameter of the Gaussian curves for the linguistics variables S, M, and L in Fig. 6.7(a) (for RTT variance) to 40, 100, and 160, respectively. Yet, the width γ of each curve is set to 50.

The universe of discourse of both input variables, RTT mean and variance, are set to 500 ms because no significant number of RTT greater than 500 ms is found in our measurements.

The ranges assigned to the output parameters CO, UC and BE (Fig. 6.7(b)) are set to [0,0.3), [0.3,0.7], and (0.7,1], respectively. These values are strategically established to reflect the neighborhood of 0.0, 0.5, and 1.0 in Fig. 6.7(b). Recall that the fuzzy engine will provide an output value in the range [0,1], and the ranges above will define if such a value represents congestion, uncertain, or bit error.

6.6.2 Fuzzy Engine Correctness

For this evaluation, we first conducted simulations using the ns2 simulator for collecting representative data on which our fuzzy engine performed inference. The simulation setup is the same described in section 6.3 for the 3-hop scenario. We ran similar conditions presented in section 6.3 except that we considered only two extra FTP flows for disturbing the wireless channel instead of three. The rationale here is that three flows generate so many losses that it becomes difficult to have representative results with the short runs performed here. Each interval under constraint (bit error or congestion) simulated lasts 100 seconds.

The evaluation of the proposed engine was then conducted by providing it with RTT mean and variance values obtained under the simulations conditions above. The engine output was then compared to the actual reason of the detected packet loss.

Fig. 6.9 shows the fuzzy engine performance concerning the number of correct detections over three distinct conditions, namely under congestion, bit errors (10% of PER) and the combination of both (5% of PER + 1 competing flow). For each of the three conditions we ran three different RTT sampling rates, namely 20, 40 and 60 RTTs per sample. Here a clarification is necessary. The mentioned sampling rates correspond to the number of RTT samples considered in the calculation of mean and variance and not the number of new incoming RTTs. Fig. 6.8 illustrates this procedure.

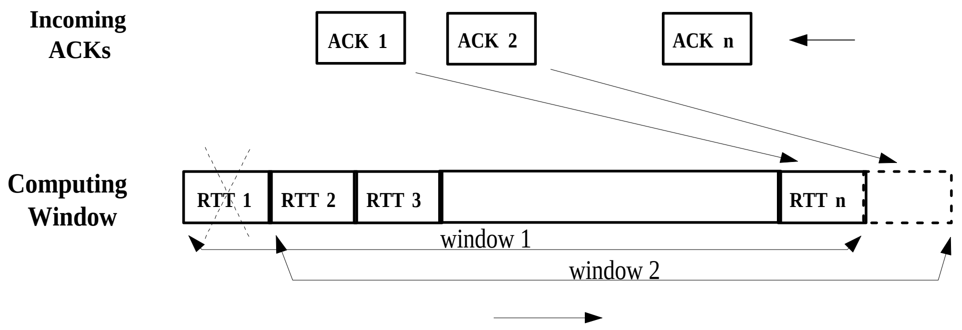


Figure 6.8: Window evaluation and updating

As depicted in Fig. 6.8, we define here a buffer called *computing window*. This buffer is fixed in size (window) and updated continuously as new ACKs arrive. In these evaluations we enabled the TCP timestamp option, so that each incoming ACK carried its experienced RTT. Upon ACK receipt, the oldest RTT value (RTT1) is discarded and the newest one is inserted into the buffer in a FILO (First In Last Out) manner. This scheme is

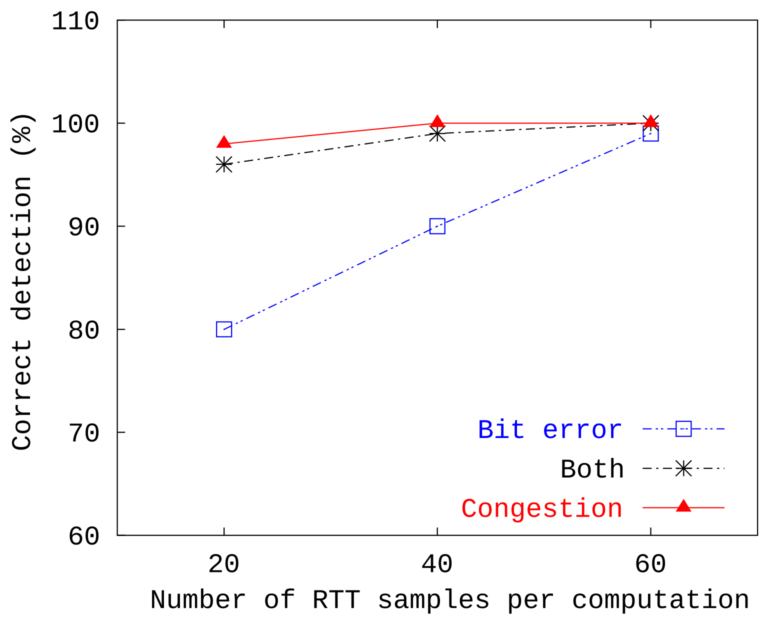


Figure 6.9: Correctness of the fuzzy engine

needed because the RTT mean and variance calculated by (6.1) and (6.2) cannot provide accuracy if the number of samples is too small. In addition, to sample all ACKs in real-time incurs in too high detection delay.

The purpose here is to determine not only the accuracy itself but also the tradeoff between accuracy and detection delay. The results show that for any of the three RTT sampling rates, congestion is detected over 98% in all cases, while bit error constraints detection decreases considerably for lower number of sampled ACKs. The results could certainly be fine tuned by using more elaborate membership functions and different settings. Besides, more accuracy from the input values could be achieved by including more fuzzy sets into the fuzzy engine.

6.6.3 Detection of Abrupt RTT Changes

The results depicted in Fig. 6.9 make it clear that the fuzzy engine may provide accurate results as long as a reasonable number of RTTs is taken in each sampling for computing the mean and variance from (6.1) and (6.2), respectively. This is effective for detections in steady state conditions. Nevertheless, abrupt changes toward congestion might lead the network to collapse if the engine does not detect that in advance. Because of that we have proposed the RR block in Fig. 6.5 to make sure that, in such cases, congestion will always be detected before the first packet loss is perceived by the TCP sender.

Hence, we simulated conditions in which the channel was initially facing some level of PER (1, 3 and 5%) and suddenly a heavy congestion started. The n and α parameters of the RR block were set to 2 and 20%, respectively. The results are depicted in Fig. 6.10 where the delays are normalized to the time the regular TCP Reno takes to detect

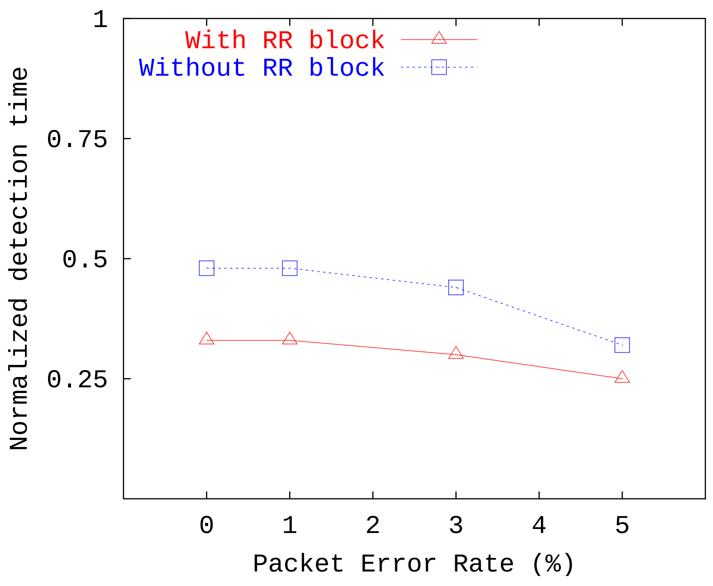


Figure 6.10: FEDM (Fig. 6.5) under abrupt congestion

the first lost packet. The main outcome is that even without the RR block, the Fuzzy-based Error Detection Mechanism (FEDM) in Fig. 6.5 is still able to detect incipient congestion quicker than the regular TCP. The improvements from the RR block are not so very significant and the detections under higher PERs are faster than the ones for lower PERs.

6.6.4 Avoiding TCP Slowdown for Medium Losses

To substantiate our discussions, we extended the simulator code so it did not slow down in the event of packet loss due to medium error. We simulated a condition in which the only constraint in the wireless channel was 10% of PER, and the run lasted 1000 seconds. The sequence numbers of the successfully transmitted packets are shown in Fig. 6.11, in which the modified TCP outperforms the TCP Reno by getting more transmissions over the same interval. The results in Fig. 6.11 were obtained in two runs, one for the unchanged TCP and another for the changed one. This evaluation simply shows that by not slowing down in response to medium error may improve performance of a single connection. However, if not carefully managed this modified sender may certainly cause degradation on the other concurrent flows. This is an error recovery issue which is not further investigated in this thesis.

In fact, the optimal TCP response to packet loss in multihop environments is subject of intense research work nowadays. Initial proposals on this subject [CRVP98,HV99a,LS01, FGML02] have proposed the aggressive error recovery strategy that we simulated above and is depicted in Fig. 6.11. These investigations claim that the sender should not slow down when reacting to losses caused by the medium in order to improve performance.

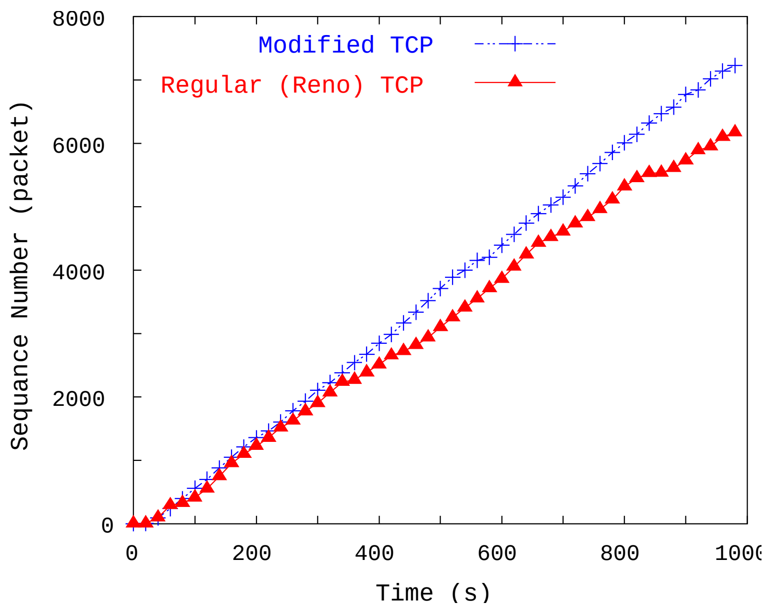


Figure 6.11: Modified TCP performance, 10% of PER

However, to the best of our knowledge, there has been no comprehensive work on these kinds of response addressing important issues such as optimal improvement bound and fairness. Our work here addresses only packet loss discrimination, leaving the proper sender reaction to medium induced losses for future work.

6.6.5 Discussions

The evaluation results show that the fuzzy engine may indeed distinguish congestion from channel error conditions with low misdetection rate. However, there is always a possibility of misdetection, which has to be considered in the error recover strategy. For instance, if congestion is misdetected as medium error and the sender simply retransmits the lost packet without slowing down, further congestion is injected into the network. In the worst case, the sender would induce more losses and should soon detect those losses correctly. The opposite case is less harmful to the network since the sender always slows down when it perceives congestion.

If corrected adjusted, the fuzzy engine should provide very high correctness rates. As the model used here for the fuzzy engine design is quite simplified, improvements are certainly possible. For instance, independent fuzzy outputs for each of the evaluated conditions (congestion and medium error) could provide more flexibility in adjusting the engine. The membership functions can be optimized by using advanced learning/training techniques such as ANFIS [Jan93]. Finally, adaptive setting models can render our approach very robust.

6.7 Summary

This chapter introduced and evaluated our proposal for an enhanced TCP error detection mechanism based on fuzzy logic. RTT patterns in short ad hoc networks were investigated and associations to the network conditions were established. The fuzzy engine performs inference on RTT mean and variance values and detects either congestion or medium error or provides a non conclusive output if the measured values are on extremely boundary areas.

The evaluations show that efficiency can be obtained provided that the input data are taken precisely enough to reflect the actual changes inside the network. This implies that a minimum number of ACKs is needed to ensure efficiency in the results, which may render this algorithm a bit slow for detecting transitions between distinct channel states. On the other hand, the proposed mechanism will be quite robust in dealing with steady state scenarios, where abrupt changes are not too frequent. We proposed supporting schemes for accelerating our mechanism, which may boost the performance of our model as a whole.

Packet loss discrimination is certainly valuable to a TCP error recovery mechanism by providing it with updated information about the conditions inside the network. Such information may be useful to prevent *congestion collapse* if the recovery mechanism is tailored to do so.

The main contribution of our investigation presented here is to disseminate the idea that it is possible to use a reasoning mechanism to infer the internal state of the network on an end-to-end basis. We are fully aware that our fuzzy engine design is very basic and so much remain to be improved. Different scenarios and more elaborate inference models have to be checked to render our proposal even more generic. Its integration with the error recovery mechanism is also an issue for future work. The RTT measurements used in our inference process represent only a simple manner to collect useful data related to the network condition. These measurements tend to be unfeasible as the number of hops in the end-to-end connection increase. Alternative measurements are indeed necessary to render the proposed engine feasible for a broad range of scenarios.

In the next chapter, we introduce another mechanism to assist TCP in multihop networks. It is also an end-to-end mechanism which aims at minimizing collisions in these environments. While the approach proposed in this chapter attempts to improve response to dropped packets, the contribution in the next chapter pursues to mitigate losses before they actually occur.

Chapter 7

A Smart TCP Acknowledgment Strategy for Multihop Wireless Networks

7.1 Introduction

Considering the background provided in the previous chapters, it is proper to affirm that only small-scale multihop wireless ad hoc networks are feasible today. These networks rely on the IEEE 802.11 MAC protocol which minimizes the well-known hidden node problem but does not eliminate it completely. Proper cooperation among the various protocols running in these networks may improve performance, but as the number of hops between sender and receiver increase the end-to-end throughput inevitably degrades toward very low levels. In this chapter we introduce and evaluate our proposal to improve TCP performance in short, feasible ad hoc networks.

TCP connections experience poor performance in these environments because of the mutual interference between the retransmission mechanisms of transport and MAC layers. This may trigger spurious retransmissions at the TCP sender causing waste of resource. In addition, the short RTS/CTS control frames required to each transmission attempt into the shared medium, associated with the backoff mechanism render ACKs transmission quite costly in these networks. The problem is that despite being much smaller, an ACK transmission causes similar medium access overhead as a data packet transmission. Contention between data and ACK is therefore considerable in multihop networks, which suggests that TCP acknowledgments should be carefully managed.

Our proposal to address these problems is an dynamic adaptive strategy for minimizing ACK induced overhead and consequently collisions. This enhances bandwidth utilization and mitigates spurious retransmissions at the sender. Using this strategy, the receiver adjusts itself to the wireless channel condition by delaying more ACK packets when the channel is in good condition and less otherwise. Our technique not only improves bandwidth utilization but also reduces power consumption by retransmitting much less than a regular TCP.

7.2 Design Decision Rationale

Before describing our mechanisms in detail, we present in this section some important evaluations to support our design decisions. Specifically, we show here that TCP should really work with small size for its congestion window limit in order to improve the spatial reuse in multihop wireless networks. In addition, the actual impact of ACK transmissions in the end-to-end performance in such networks is also shown as a relevant factor, thereby justifying reduction in the number of redundant ACKs.

7.2.1 Optimal Limit for TCP Congestion Window

The spatial reuse property discussed in section 5.3.3 suggests that a TCP sender should limit the size of its congestion window ($cwnd$) in order to achieve better performance. According to the discussion in that section, a limit of $h/4$ for $cwnd$ is presumably an optimal setting in a chain topology, where h is the number of hops between sender and receiver.

To confirm the conclusions above, we conducted a number of simulations in a typical chain topology in which distinct *cwnd* sizes and number of hops were evaluated. Fig. 7.1 depicts the results which are quite similar to the ones found in previous related work. These simulations were conducted using the ns2 simulator with default settings apart from the packet size and the channel data rate that were set to 1460 bytes and 2 Mbps, respectively.

Fig. 7.1 allows us to draw very important conclusions. First, for short chains of nodes having at most 3 hops, even a very small $cwnd$ of 1 or 2 packets is sufficient to guarantee maximum throughput. For scenarios containing up to 10 hops, which might cover many feasible scenarios, a $cwnd$ limit of 3 packets is sufficient. In contrast to some related work, such results indicate that, at least for short chains of nodes, it is does not help much to avoid decrease in $cwnd$. The problem is that $cwnd$ should be set to very small sizes anyway, and in such cases the difference in throughput for different limits of $cwnd$ is not that much, as shown in Fig. 7.1.

7.2.2 The Actual Cost of Using TCP Acknowledgments

In this section, we assess the impact of TCP ACKs transmission in wireless multihop networks. While acknowledgment packets are essential in a reliable transmission, their excessive use may be too costly in these challenging multihop networks, as shown in section 5.6. In that section, the standard delayed acknowledgment (DA) was shown to reduce ACKs induced overhead considerably in scenarios without the hidden node problem. We show here that such an observation is valid for scenarios facing hidden node problems as well. Moreover, we show that DA is not optimized for multihop wireless networks as much as our scheme is.

Before describing our mechanism thoroughly in the next section, we highlight here how much of improvement it may provide against both a regular TCP and a TCP using the standard DA. We consider then two simulation scenarios using a string of nodes topol-

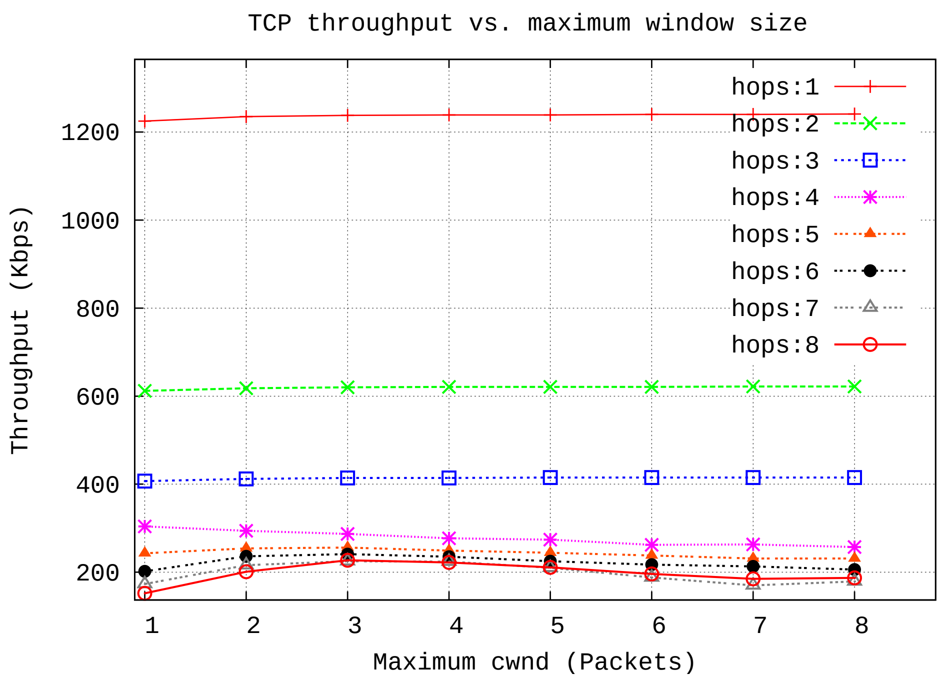


Figure 7.1: The optimal limit for the sender congestion window

ogy: no hidden node problem (3-hop) and with hidden node problem (5-hop). Table 7.1 illustrates how significant an ACK transmission might be in such environments. These results are the outcome of simulations for a single flow crossing the chain topology, and each run lasts 50 seconds.

The values in table 7.1 represent the total time the medium is busy transmitting either an ACK or a data packet. From this table, it is evident that techniques for delaying ACKs can be indeed efficient in multihop environments facing or not the hidden node problem. The last column of the table, which exhibits the ACK/DATA ratio in percentage, shows that the standard delayed acknowledgment (DA) provides significant enhancements. Likewise, table 7.1 highlights the remarkable performance of TCP-DAA by bringing the ACK overhead, relative to data packets, to about 3% in both scenarios. We show below that our mechanism sustains such a high improvement in a variety of scenarios.

7.3 Dynamic Adaptive Acknowledgment

This section introduces our proposed **dynamic adaptive acknowledgment** (DAA) strategy that we call **TCP-DAA**. As mentioned above, TCP-DAA targets feasible scenarios where the IEEE 802.11 standard may provide acceptable performance. TCP-DAA was first introduced in Infocom 2005 [OB05] as an efficient mechanism to improve TCP performance in short ad hoc networks facing moderate level of bit error rates. Recent inves-

Table 7.1: Data and ACKs transmission delay (in seconds)

	DATA	ACK	ACK/DATA
	3-hop		
No delayed ACK	90.1	59.6	0.66
Standard DA	97.6	13.7	0.14
TCP-DAA	127.5	3.9	0.03
	5-hop		
No delayed ACK	96.5	52.5	0.54
Standard DA	89.5	17.2	0.19
TCP-DAA	123.6	4.0	0.03

tigations on 802.11 have shown that this MAC protocol is effective in recovering most of the wireless induced losses in typical scenarios, but it does not scale as the number of wireless hops increase due to the hidden node problem discussed in detail in the prior chapters. There are, however, a number of important applications and scenarios in which the number of hops involved will be far below 10 hops, and the number of nodes will normally not exceed 100 nodes. Typical examples include ad hoc networks in classrooms, meeting and workshop spots, small working offices, Wi-Fi in home buildings, and many others.

TCP-DAA relies on the fact that the 802.11 protocol schedules the contending transmission requests in such a way that normally every node cannot transmit more than one frame at a time, but must contend for the medium again in order to prevent a node from monopolizing the medium. This feature allows a TCP source to send more data at once without receiving ACKs, because the MAC layer scheduling prevents potential bursts.

The procedure of delaying acknowledgments at the receiver may be counterproductive if the receiver does not adjust itself on a regular basis to fit the channel changing conditions. We claim that the standard DA as well as the work in [EA03], which proposes a similar strategy to ours, do not react appropriately to the channel disruptions. We call the latter LDA (Large Delayed Acknowledgment) to facilitate its identification in the comparisons that follow. LDA improves performance over the standard DA, but does not react to out-of-order packets as the former does. Besides, both mechanisms rely on a fixed timeout interval of typically 100 ms. This is not efficient because the packet inter-arrival interval at the receiver changes not only with the channel data rate, but also with the intensity of traffic going through the network. Another important aspect concerning delayed ACKs is the amount of ACKs to be delayed. That is, the number of ACKs merged by the receiver must also fit the network conditions adaptively in order not to adversely impact the connection performance when the medium is heavily constrained.

TCP-DAA combines the idea of higher number of delayed ACKs with the dynamic reaction proposed in RFC 2581, i.e, reaction to packets that are either out-of-order or filling in a gap in the receiver's buffer. Furthermore, our protocol adjusts itself to the channel conditions, in that it adaptively computes the timeout interval for the receiver on the basis of the data packet inter-arrival time. In this way, the receiver delays just enough to avoid spurious retransmissions at the sender, as elaborated in section 7.3.2, and is able to adapt itself to different levels of delays imposed by the wireless channel. This strategy renders the mechanism independent of both channel data rate and number of concurrent flows crossing the network. As we will shown in section 7.4, TCP-DAA outperforms the two schemes above (standard DA and LDA) in several scenarios.

7.3.1 Algorithm Overview

Before addressing the algorithm in detail, this section provides an overview on the fundamentals of the proposed approach. The key concept of our scheme is to provide the TCP receiver with an adaptive control mechanism capable of using the channel capacity efficiently and dynamically as done at the sender side. As discussed in sections 2.6.1 and 5.7.1, the standard delayed acknowledgment (DA) can enhance TCP performance in multihop networks by reducing the number of ACKs in transit and so contentions in the shared medium. This minimizes collisions and consequently increases end-to-end throughput.

Nevertheless, the wireless medium condition changes over time by diverse factors such as congestion level, fading, interference, mobility, etc. Hence, the receiver should bring down the amount of ACKs it injects into the network whenever the channel is in good shape, and the receiver should not delay crucial ACKs, or delay just a bit, if the channel is constrained. This is to some extent the same principle followed by the sender congestion control mechanism that probes the network for resource availability.

Furthermore, the sender should also avoid excessive load in the network by limiting its congestion window to an appropriate value. Section 5.7.1 discusses some past work that have shown the importance of limiting *cwnd* on the basis of the number of hops in place. As presented in section 7.2.1, TCP sender should indeed work with a relatively very small $cwnd$ to improve performance. Therefore, our mechanism limits the sender congestion window to four in order to conform to the evaluation results in section 7.2.1. Fig. 7.2 depicts the general behavior of our proposal.

The receiver contains a delaying window (*dwin*) defining the amount of ACKs to be merged into a single ACK. This window varies from two to four to fit the wireless channel condition. Whenever the channel is in good shape, *dwin* is maintained at its maximum size of four and when losses occur it is reduced to two in order to follow the standard DA behavior closely. by receiving timely, in-order data packets the receiver increases *dwin* by steps of one until four.

Similar to the sender congestion window, the receiver delaying window allows a dynamic adaptation to the channel conditions. The receiver also keeps a timeout timer to ensure that ACKs are not delayed excessively because that would trigger spurious retransmission at the sender. The timeout interval is adaptive in that it is calculated on the the basis of the data packet inter-arrival at the receiver. In fact, the receiver reacts immediately to either out-of-order packets or timeout in order to be fully responsive to the network needs. The sections below present the design details of our mechanism.

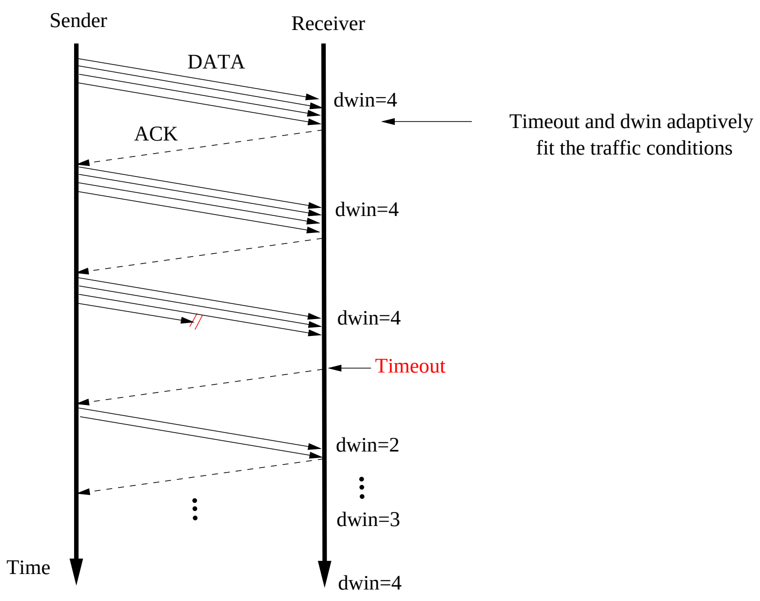


Figure 7.2: TCP-DAA approach

7.3.2 Requirements at the Sender

By delaying the acknowledgment notification to the sender, the receiver may trigger a retransmission by timeout at the sender if the receiver delays excessively. Thus, the receiver has to be well adjusted in order to avoid such spurious retransmissions. Moreover, the fewer amount of ACKs for the sender might lead TCP to low performance in typical wired scenarios where the congestion window ($cwnd$) limit is usually high. This might happen because a TCP sender may only enlarge its $cwnd$, toward the limit, upon receipt of ACKs. This problem is not so critical in our technique, however, as the $cwnd$ limit in place (4 packets) is rather low. This means that after a reduction of $cwnd$ due to a lost packet, it will quickly reach the limit again upon receiving a few ACKs, as explained in section 7.3.6.

The current development of TCP-DAA is focused on the receiver side, while a comprehensive investigation on the sender side is still to be done. In short, delayed ACKs can affect the sender side with respect to response to losses and timeout interval computation. In addition to setting *cwnd* to four, the technique we used for minimizing unnecessary retransmissions by timeout consists of two adjustments: 1) the number of duplicate ACKs for triggering a retransmission by the fast retransmit mechanism is decreased from 3 to 2 packets, which is in line with the limit transmit discussed in section 2.6.3 in the sense that we work with a small $cwnd$ limit; 2) the regular retransmission timeout interval is increased fivefold for compensating the maximum of four combined ACKs. These are the only two changes performed on the regular TCP sender, which proved to be effective in most of our evaluations.

7.3.3 Delaying Window

The dynamic behavior of TCP-DAA depicted in Fig. 7.2 implies that after startup and having no losses, the receiver always delays the maximum amount of ACKs. This means that for each four data packets received ($P_i, P_{i+1}, P_{i+2}, P_{i+3}$), the receiver replies with an ACK. The delay management is performed through a delaying window called $dwin$ which limits the maximum number of ACKs to be delayed. Fig. 7.3 illustrates the behavior of $dwin$.

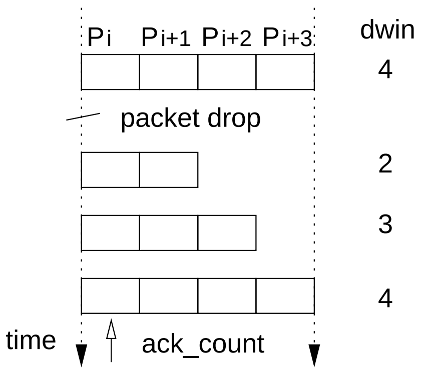


Figure 7.3: Receiver dynamic window for delaying packets

Under normal conditions, the delaying window starts set to one and increases gradually upon timely, in-order data packets receipt until it reaches four. The limit of four is imposed by the sender $cwnd$ limit that is also set to four. Higher $dwin$ would not work because the sender would not have enough data packets to transmit to meet the $dwin$ value, which would lead the sender permanently to timeout. The receiver also uses another variable called *ack_count* to indicate how many data packets are being combined by the receiver. Its purpose is explained in section 7.3.4 below.

As long as the wireless channel is unconstrained, it is advantageous to keep $dwin = 4$ for the reasons mentioned above. When facing losses, however, $dwin$ should be reduced in order to avoid further performance degradation at the sender. This occurs due to the fact that during such periods the channel has less packets in flight to trigger the fast retransmit mechanism at the sender. As a result, if the receiver does not transmit an ACK quickly, the sender may time out.

As earlier mentioned, the receiver keeps a timer that is reset whenever a data packet whose ACK is to be delayed arrives. This is needed to detect highly constrained channel conditions. Additionally, the receiver keeps track of the sequence number of the incoming data packets so it may detect a poor channel when receiving out-of-order packets.

Thus, whenever the receiver gets a packet that is either out-of-order or filling in a gap in the receiver's buffer, or when its timer expires, it sends immediately an ACK to the sender and reduces $dwin$ to the size of two packets. We chose to resume $dwin$ growth from two instead of one because we aimed, in such situations, to go back to a behavior similar to that of the standard DA. The rationale here is that the standard

DA performs satisfactorily in these environments, as described above. To reduce $dwin$ to one is more conservative and may be proper for highly noisy environments where considerable improvements are hard to achieve. Fig. 7.3 illustrates a situation in which dropped a packet is detected. Note that $dwin$ is decreased to two, then increased to three and subsequently to four as new data packets arrive.

After a reduction in $dwin$, subsequent timely data packets trigger $dwin$ growth toward the maximum size again. Timely data packets here refer to the incoming data packets that are neither out-of-order nor filling a gap in the receiver's buffer. Using this dynamic behavior, associated to the timer-based monitoring, the receiver prevents the sender from missing ACKs when packet losses occur. As mentioned in section 5.7.1, the LDA proposal [EA03] works with a fixed $dwin$'s size of four packets (except at startup), and uses a large $cwnd$ limit at the sender to keep the channel full of data packets in flight. While this procedure may prevent a lack of ACKs at the sender, it may also induce excessive number of retransmissions at the sender.

The $dwin$ growth is governed by (7.1) which shows that such an increase may be fixed at one (packet) or determined by the startup speed factor μ, with $0 < \mu < 1$. The reason for this factor is that during the startup phase (initial of the session), the sender starts with a window size of two packets and then increases it by one at every ACK received.

Although $dwin$ is initialized to one, if it starts from startup being increased at the rate of one packet per data packet received, it grows too quickly. This causes a deadlock because the receiver does not receive enough data packets to transmit an ACK, which ends up triggering a timeout at either the receiver or the sender or both.

Thus, the threshold $maxdwin$ is used to define the instant the startup phase is over, which occurs when $maxdwin$ first reaches its maximum value and becomes ***true***. From our evaluations, we noticed that by properly setting the μ parameter, our algorithm achieved better performance for short-term transmissions, as shown in section 7.4.6.

$$dwin = \begin{cases} dwin + \mu, & \text{if } maxdwin = false \\ dwin + 1, & \text{otherwise} \end{cases} \tag{7.1}$$

7.3.4 Timeout Interval Calculation

Fig. 7.4 illustrates in more detail how the receiver keeps track of the packet inter-arrival interval and handles the ACKs delay. Under normal conditions, i.e., after startup and without any loss, for every four data packets received, the receiver replies with an ACK. Whenever a given ACK $_{i,i+1,i+2...}$ is to be delayed, an associated timer is started (t_i), or restarted (t_{i+1}, t_{i+2}) if there is one already running. The receiver also measures the data packet inter-arrival gap between the packets for which the ACK is to be delayed ($\delta_{i,i+1,i+2...}$).

The receiver keeps track of the number of ACKs delayed by maintaining the ack_count variable which increases from one to the current value of its delaying window ($dwin$), as shown in Fig. 7.3. By checking the value of ack_count, the receiver is able to determine whether the received packet is the first one from the group that is going to have their acknowledgment delayed. This is possible because ack_count is reset whenever it reaches

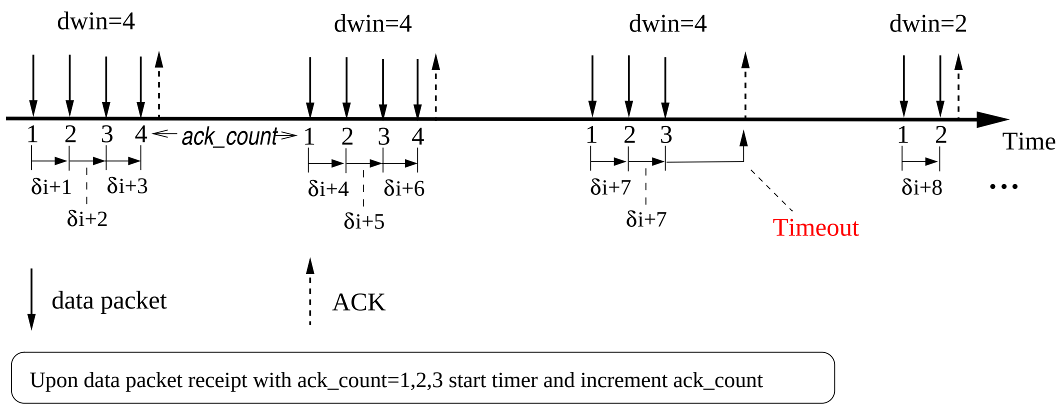


Figure 7.4: TCP-DAA receiver mechanisms

the $dwin$ value.

In case the received packet is the first one, the inter-arrival interval between the last received packet and the current one is not taken. This is needed to avoid that unnecessary intervals such as the one between δ_{i+3} and δ_{i+4} in Fig. 7.4 are improperly considered for the timeout interval computation. By using this strategy, we assure that in normal conditions, the inter-arrival measurements will reflect very closely the gap between the received data packets triggering delayed ACKs. Note that under packet loss, the receiver will not need such measurements as it will not delay out-of-order packets. Rather, it will await until it receives in-order packets again.

Similarly to the TCP sender, the receiver uses a low-pass filter to smooth the packet inter-arrival intervals. Upon arrival of a given data packet p_{i+1}, it calculates the smoothed packet inter-arrival as indicated in (7.2), where $\overline{\delta}_i$ refers to the last calculated value, δ_{i+1} is the packet inter-arrival sampled, and α is the inter-arrival smoothing factor, with $0 < \alpha < 1$.

$$\overline{\delta}_{i+1} = \alpha * \overline{\delta}_i + (1 - \alpha) * \delta_{i+1} \tag{7.2}$$

The value computed from (7.2) is used to set the timeout interval at the receiver. In our design, we established that after the receipt of a data packet that causes an ACK to be delayed, it is reasonable to wait for at least the time the second next packet is expected. The rationale for this is that the delay variations are relatively high in such environments and in case of a single dropped packet, the next data packet will arrive out-of-order, which will trigger immediate transmission of an ACK, as recommended in RFC 2581. However, if it was only a delay variation, and the data packet arrives before the expected time for the subsequent packet, no timeout is triggered and the receiver avoids sending an extra and unnecessary ACK into the network.

Hence, we use a timeout interval T_i as shown in (7.3). Note that the factor 2 in (7.3) refers to the estimated time for the second expected data packet to arrive. This equation also includes a timeout tolerance factor κ, with $\kappa > 0$, defining how tolerant the receiver may be in deferring its transmission beyond the second expected data packet. In short, the effective timeout interval T_i is at least twice the the smoothed value $\overline{\delta}_i$ and may be higher depending on the value of κ.

$$T_i = (2 + \kappa) * \overline{\delta}_i \tag{7.3}$$

7.3.5 Packet Loss Handling

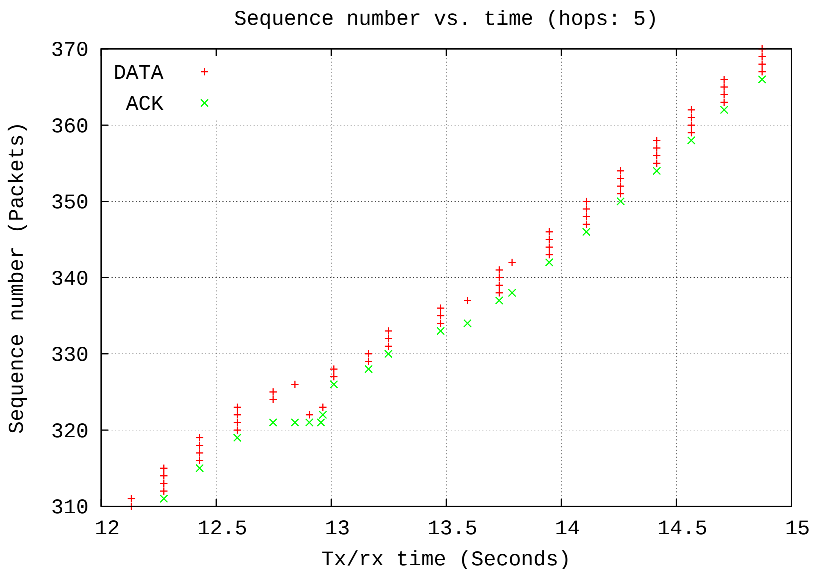
In order to better understand the concepts explained above, we show here a typical response of our mechanism when reacting to lost packets. We include the response of the LDA scheme to highlight the difference of our proposal to that one. Fig. 7.5 exhibits a part of a simulation run in which both strategies faced a dropped packed in a chain topology of 5 hops.

Let $packetn$ be the data packet of sequence number (n). Fig. 7.5(a) shows that the sender transmits four packets (320-323) at time 12.6 seconds. In this run, $packet322$ and $packet323$ are dropped. The receiver times out and acknowledges only two packets (320, 321) instead of four. The receiver also updates its $dwin$ size to two. Upon receipt of the ACK for $packet321$, the sender sends two new packets (324, 325) because two packets were acknowledged. At this moment there are only 2 packets in flight (324, 325). Since $packet324$ and $packet325$ are detected by the receiver as out-of-order packets, they trigger immediate acknowledgments at the receiver. By receiving the first duplicate ACK, the sender transmits a new packet (326) which will also be out-of-order.

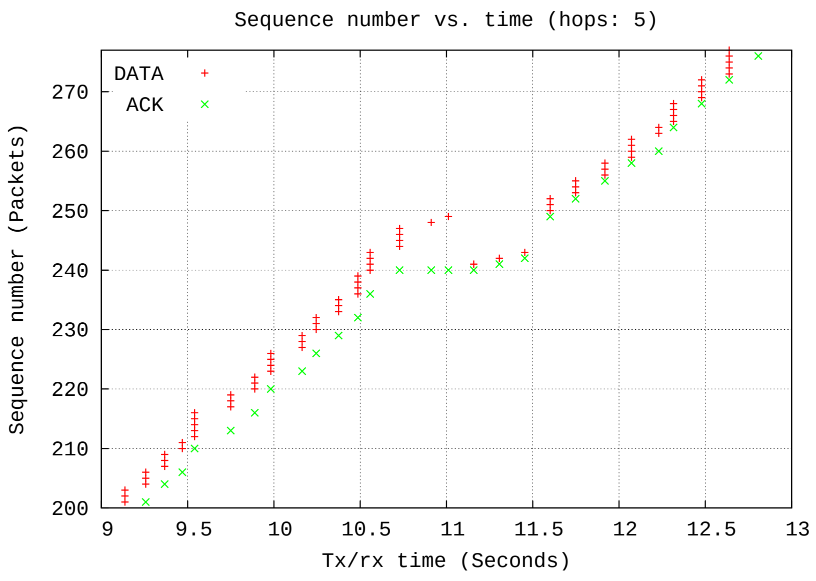
When the sender receives the second duplicate ACK at instant 12.9 seconds, it retransmits the first lost packet (322), and halves its $cwnd$ size to two packets (fast retransmit/fast recovery). The $cwnd$ will be expanded gradually after the sender exits the fast recovery phase. When the sender receives the third duplicate ACK, at time 12.96 seconds, it does nothing because it is in the fast recovery phase. At the instant 12.97 seconds, the sender gets the acknowledgment for $packet323$, retransmits this packet and then exits the fast recovery procedure. $Packet323$ fills in the gap at the receiver's buffer, which triggers the ACK of $packet326$ due to the cumulative property of TCP acknowledgment strategy.

At instant 13.01 seconds, the sender receives the acknowledgment for $packet326$, and so transmits two new packets (327, 328). These two packets cause the receiver to send one ACK only as its $dwin$ is set to two packets at this point. After that, $dwin$ increases and, as a consequence, the number of delayed ACKs increase toward 4. Fig. 7.5(a) shows two spurious retransmissions by timeout at the receiver. $Packet339$ and $packet338$ are unnecessarily acknowledged at the instants 13.62 s and 13.79 s, respectively. This means that the timeout interval computation may still be improved, but that is left for future work.

Fig. 7.5(b) shows the response of LDA to a packet loss. In this simulation run, $packet241$ is lost at about 10.55 seconds. Differently from our technique in which the amount of packets in flight are limited to four packets, the proposed LDA works with a large limit for the $cwnd$ (10 packets), so it has more packets in flight than TCP-DAA. One can notice in Fig. 7.5(b) that although only one packet has been dropped, various acknowledgments triggered the transmission of less than the optimal four packets by the



(a) TCP-DAA



(b) LDA

Figure 7.5: Delayed acknowledgment strategies

receiver. This shows that the retransmission timer expired in several situations unnecessarily.

Additionally, the sender waits for the default three duplicate ACKs for retransmitting the dropped packet, which incurs in a longer time for it to take action. By comparing Fig. 7.5(a) with Fig. 7.5(b), one can clearly see that TCP-DAA provides more stability regarding the number of delayed ACKs. As a result, less packet delay variation is perceived by the sender, which in turn tends to minimize the inaccuracy in the timeout interval computation at the sender.

7.3.6 An Alternative Delaying Window Strategy

The basic delaying window strategy in 7.3.3 may be inefficient in scenarios facing considerable loss rates. In this section, we investigate improvements to such scenarios. We first observe that if the channel is facing constant losses, then it seems to be more appropriate to reduce the delaying window ($dwin$) to one in order to avoid timeout at the receiver. Additionally, the $dwin$ should be enlarged by less than one for every data packet received. This is more conservative than the initial strategy above, which is needed to ensure robustness for the mentioned scenarios. Hence, we propose to adjust the receiver side as illustrated in Fig. 7.6.

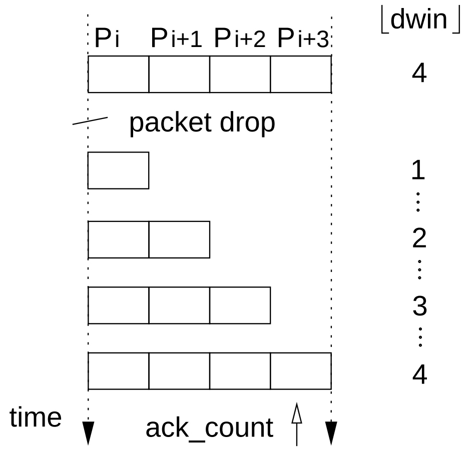


Figure 7.6: An alternative delaying window strategy for robustness against losses

Upon loss detection by either timeout or out-of-order packets, the receiver transmits an acknowledgment immediately and shrinks $dwin$ to one. By receiving new, in-order packets the receiver gradually expands $dwin$ by steps smaller than one. The operator $\lfloor x \rfloor$ represents the mathematical floor function which is defined as follows: for a real number x, $\lfloor x \rfloor$ results in the largest integer less than or equal to x. In other words, $\lfloor dwin \rfloor$ represents the integral part of $dwin$.

Fig. 7.6 illustrates that only the integral part of $dwin$ is needed in the comparison with *ack_count*. This establishes three ranges in which $dwin$ increases without causing any impact on the amount of packets to be delayed. These ranges are between the successive $\lfloor dwin \rfloor$ values in Fig. 7.6, i.e., between 1-2, 2-3, and 3-4. It is obvious that the smaller the steps by which $dwin$ increases the more points in each of these ranges, and consequently the longer the interval to $dwin$ fully enlarge until four.

It is not simple to determine the exact amount by which $dwin$ should be increased when an ACK is transmitted as many factors influence $dwin$ growth. For example, when the wireless channel is unconstrained, $dwin$ should increase as fast as possible, and under high loss rates it should grow more carefully. We estimate here the worst case scenario as an upper bound only rather than a rigorous specification.

A TCP receiver should provide enough ACKs to its corresponding sender in order to prevent timeout at the sender and also to trigger the sender $cwnd$ growth properly until its limit. Assume that the sender has just timed out while in steady state. Its $cwnd$ is reset to one and the slow start threshold $ssthresh$ is set to one half the current $cwnd = 4$, i.e., $ssthresh$ is set to two. In this case the sender increases its $cwnd$ by one when the next ACK arrives because it is in slow start phase ($cwnd < ssthresh$), and then it enters the congestion avoidance phase. As presented in section 2.4, the $cwnd$ increase (in packets) for the i_{th} received ACK during congestion avoidance is given by (7.4), where $cwnd_{i-1}$ refers to the previous value of $cwnd$.

$$cwnd_i = cwnd_{i-1} + \frac{1}{cwnd_{i-1}} \tag{7.4}$$

Although $cwnd$ grows exponentially in slow start and linearly in congestion avoidance, we can use the equation above for both phases because of our small window limit of four packets. In fact, since $ssthresh$ is set to two upon loss detection, only one ACK is enough to lead the sender to congestion avoidance. Moreover, replacing $cwnd_{i-1}$ in (7.4) with one (the reset value) the left-hand results in two, which is exactly the same that is obtained with slow start. Hence, assuming that $cwnd$ increases continuously from one to four governed by (7.4), the accumulated window increase is given by (7.5), where $cwnd_0$ is the value to which $cwnd$ is set just after a slowdown and $cwnd_i$ is the value of $cwnd$ at the i_{th} increase step, which ranges from one to n and is given by (7.4). If a loss is detected by timeout, $cwnd_0$ is reset to one. A loss detected by the fast retransmit mechanism causes $cwnd_0$ to be reset to a value between 1 and 2, depending on the current $cwnd$ value. For simplicity we assume $cwnd_0 = 1$ in the modeling below.

$$W = cwnd_0 + \sum_{i=1}^{n} \left(\frac{1}{cwnd_i} \right) \tag{7.5}$$

Solving (7.5) for $W = 4$, the *cwnd* limit in our mechanism, results in $n = 7$. This means that the window expansion process takes seven steps to reach the maximum size of four packets. Therefore, the receiver should take into consideration this value when enlarging its $dwin$. Recall that this variable defines the number of ACKs to be delayed. Fig. 7.7 illustrates how many steps the $dwin$ should follow to satisfy the sender demand for ACKs. While $dwin$ is less than two (first range), each data packet received triggers the transmission of one ACK and $dwin$ increases by $1/m$. So the receiver transmits m ACKs in response to m data packets received. When $dwin$ is between 2-3 (second range), every other data packet generates an ACK, which results in approximately $m/2$ ACKs being transmitted in this range. Likewise, for $dwin$ between 3-4 (third range) an ACK is sent for every 3 data packets and so about $m/3$ ACKs are transmitted in this range.

To meet the sender needs in terms of acknowledgments, the number of ACKs should be equal to the amount of expected $cwnd$ increases n necessary to expand this window to the limit, i.e., seven ACKs. Thus, the summation of the ACKs generated in each range of Fig. 7.7 should result in seven. In other words, $m + \frac{m}{2} + \frac{m}{3} = 7$, which specifies a $m = 3.8$ as the number of increases for each ack_count range shown in Fig. 7.7. The inverse of m gives us the $\mu' = 0.26$ parameter which determines how much $dwin$ should increase per correct data packet received. Thus, the equation governing $dwin$ growth is changed from (7.1) above to (7.6).

$$dwin = \begin{cases} dwin + \mu, & \text{if } maxdwin = false \\ dwin + \mu', & \text{otherwise} \end{cases} \tag{7.6}$$

Section 7.4.7 shows the performance evaluation of this improved algorithm which we call TCP-DAAp (TCP-DAA plus). As addressed in that section, with TCP-DAAp the algorithm at the sender side should react more promptly to losses. The reason is that the number of retransmissions by timeout are assumed to be significantly higher in such cases. Hence, TCP-DAAp uses a regular RTO increased twofold instead of fivefold.

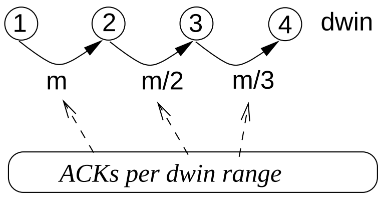


Figure 7.7: Optimal amount of ACKs for high loss rates

7.4 Performance Evaluations

Here we evaluate and compare the performance of TCP-DAA (including TCP-DAAp) with the other TCP flavors and with the LDA proposal presented in [EA03]. We compare our work with LDA because it also investigates a delayed acknowledgments strategy similar to ours for improving TCP performance in multihop networks. We also compare our results with other TCP flavors in their theoretical best conditions, so as to make sure that our proposal is indeed efficient among a wide range of options. Hence, we simulate the other TCP flavors including two potential improvements: the standard delayed acknowledgment (DA), and a low limit for their $cwnd$ (3 packets), for the sake of efficient bandwidth utilization, as explained in section 5.3.3.

7.4.1 Simulation Scenario

We used the ns2 [$\text{EFF}^+00, \text{EHH}^+00$] simulator in our evaluations of the two scenarios depicted in Fig. 7.8 in which we have a single chain topology and a grid topology with 9 (maximum) and 25 nodes, respectively. In both topologies, each node is 200 meters away from its closest neighbors. When applicable, the throughput r is calculated as $r = \frac{seq*8}{stime}$, where seq is the maximum sequence number (in bytes) transmitted and acknowledged and $stime$ is the simulated time. Unless otherwise stated, the other parameter settings are the ones shown in table 7.2.

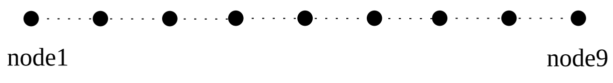
Table 7.2: General simulation parameters

Parameter	**Value**
Channel bandwidth	2 Mbps
Transmission range	250 meters
Interference range	550 meters
Packet size	1460 bytes
Window limit (WL)	3 packets
Regular TCP	NewReno
Routing protocol	AODV
MAC protocol	IEEE 802.11
Traffic type	FTP
TCP-DAA α	0.75
TCP-DAA κ	0.2
TCP-DAA μ	0.3
Confidence interval	95%
Number of runs averaged	5
Initial TCP-DAA rec. timeout	200 ms
Simulation time	300 seconds

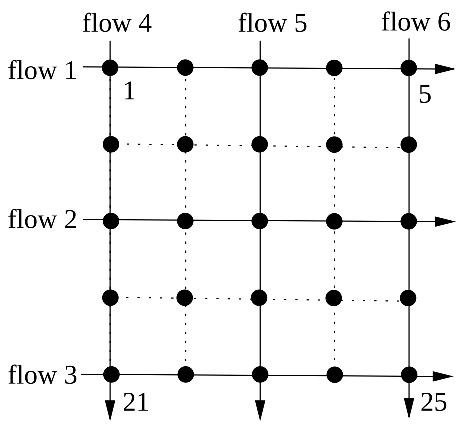
7.4.2 Throughput in the Chain Topology

In this section, we investigate the bandwidth utilization over a wide range of situations. While most related work present results over either a single flow or a fixed number of hops, we present here our evaluation not only for different number of hops but also for different number of concurrent flows in the chain topology of Fig. 7.8(a). In addition, we compare our results with the main existing TCP flavors. It would be reasonable only to compare our algorithm with the other flavors, including the regular TCP, with DA enabled. However, as the vast majority of related work present results over the regular TCP without DA, we also evaluate this configuration here to allow better comparison with related work.

Fig. 7.9 exhibits a remarkable achievement of TCP-DAA. These results are obtained by taking the average of 5 runs, as shown in table 7.2. TCP-DAA outperforms all the other algorithms in most situations. Only TCP Vegas and regular TCP, both with optimal setting (DA+WL), outperform slightly our mechanism in the scenario with 7 hops and 2



(a) chain topology



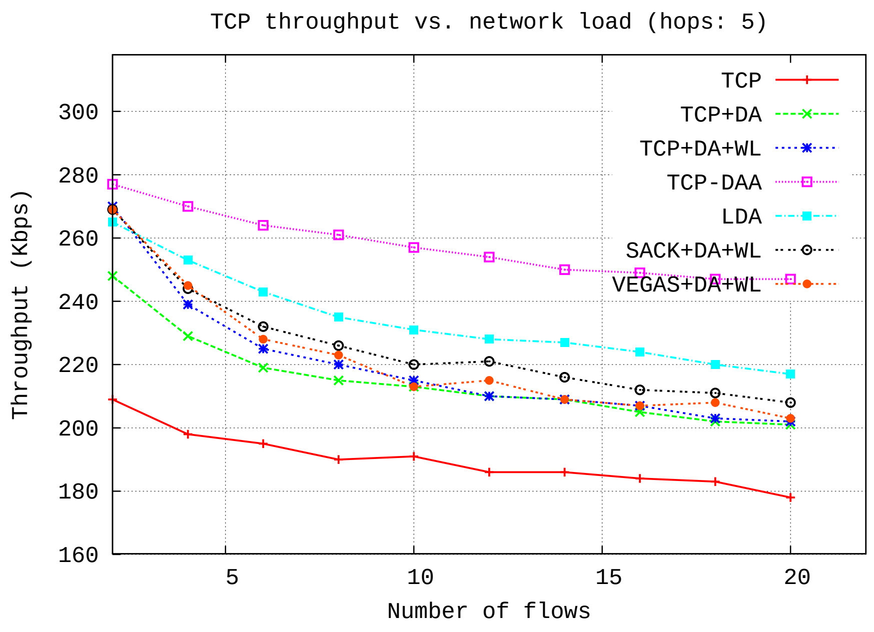
(b) grid topology

Figure 7.8: Simulated scenarios

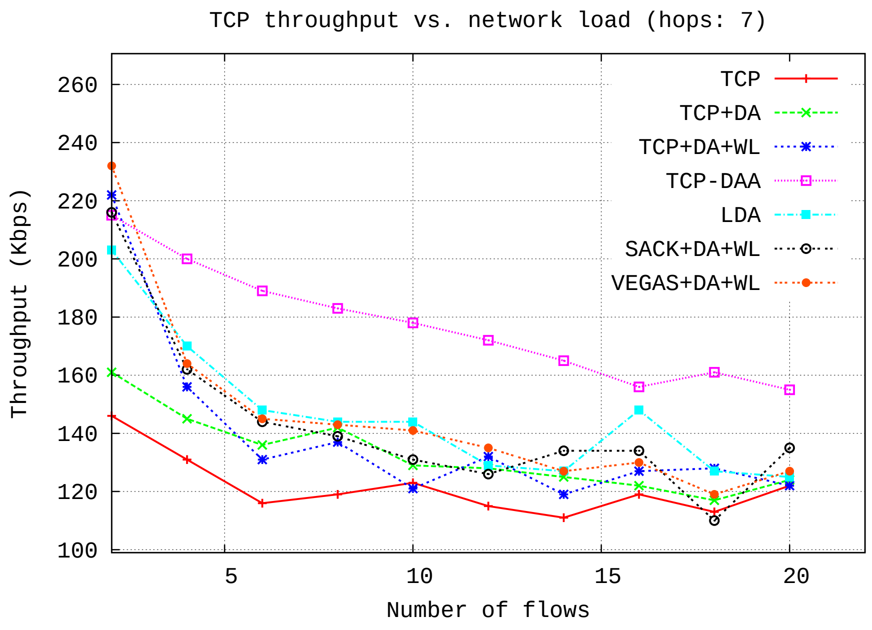
concurrent flows. Nevertheless, as the number of flows increase, the performance of both strategies degrades significantly, while it is not the case in our scheme. We believe that TCP-DAA will be further improved if the default sender's RTO calculation is fine tuned to its strategy.

It is interesting to note that, in general, the more flows the better the improvement of our algorithm over the other protocols. One reason for that is the high level of transmission delays due to the higher number of flows in the network. Under such high delays, the packet delay variance becomes less significant in the RTO calculation, and so less interference of the delayed ACKs is perceived by the sender. Another reason lies in the sender's high tolerance to invoke the timeout procedure, which renders the TCP-DAA sender less aggressive than a regular sender. As shown in section 7.4.4, this behavior is advantageous with regard to spurious retransmissions, resulting in more bandwidth to the concurrent flows. In case there is no concurrent flow to use the left bandwidth, while the sender is waiting for the timeout, then that bandwidth is simply wasted.

Overall, the observed improvements are up to about 50% over regular TCP, and over LDA improvements of up to 30% are obtained. We also performed simulations for 1, 2, 4 and 6-hop scenarios and the results are similar, in some cases less improvement are observed, but in most cases our algorithm performs significantly better than all the others.



(a) 5-hop



(b) 7-hop

Figure 7.9: Aggregate throughput in the chain topology

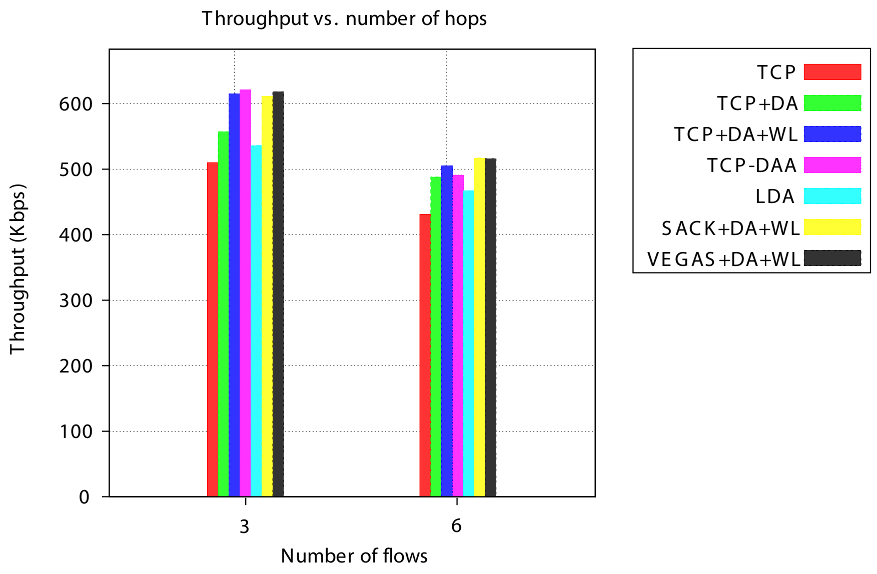


Figure 7.10: Aggregate throughput in the grid topology with cross traffic

7.4.3 Throughput in the Grid Topology

Here we describe the investigations carried out in a more complex scenario, the grid topology illustrated in Fig. 7.8(b). In these evaluations, we have first only 3 flows crossing the topology horizontally (flows 1, 2 and 3 in Fig. 7.8(b)). In the next step, 6 flows (3 horizontal and 3 vertical) are injected into the network concurrently. The averaged results are depicted in Fig. 7.10.

This is a critical scenario, given the various interactions among the nodes in place. The level of dropped packets is high, and so is the degradation of our mechanism. As the scheduling strategy of 802.11 is inherently unfair, it may happen that in some circumstances TCP-DAA outperforms the other implementations [OB04b], but its overall performance is expected to be similar to that of a regular TCP, as illustrated in Fig. 7.10.

In these simulations, our mechanism performs roughly the same as the other implementations for the run with only horizontal flows (3 flows). Its efficiency deteriorates for the 6 flows case, going down to the level of the regular TCP with DA. Note that the window limit (4 packets) of TCP-DAA is slightly higher than the one of the regular TCP (3 packets), which may explain why TCP-DAA does not reach the performance of the TCP+DA+WL configuration. We have not used the same window limit in both algorithms because the limit of 3 packets was expected to perform better for the regular TCP, as shown in section 7.2.1. TCP SACK and Vegas perform best in these evaluations. Our algorithm would most likely follow SACK and Vegas's performance closely if it had been implemented over these flavors, but it was implemented over TCP NewReno which performs well in a variety of scenarios. We believe that a more robust MAC protocol, concerning fairness, may favor our mechanism in such environments.

7.4.4 Retransmissions

Since TCP-DAA is more tolerant to packet delay variations by having the regular RTO (retransmission timeout) at the sender multiplied by a factor of 5 and a dynamic acknowledgment strategy at the receiver, it is expected that it minimizes spurious retransmissions by timeout.

Fig. 7.11 shows the result of a simulation run in which 10 flows share the medium in the chain topology of Fig. 7.8(a), under different number of hops. The figure exhibits the aggregate number of retransmissions including retransmissions by both timeout and the fast retransmit mechanism.

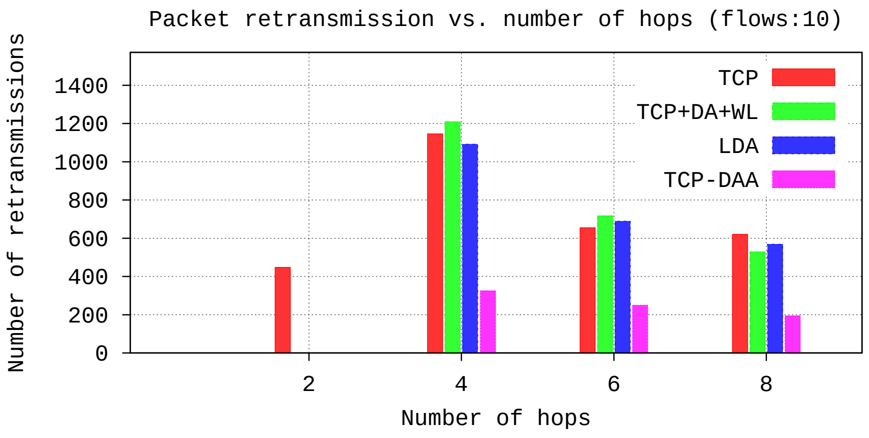


Figure 7.11: Aggregate number of retransmissions

It can be seen that for the 2-hop scenario, our protocol does not retransmit any packet at all, while the regular TCP retransmits over 400 times. For 4, 6 and 8 hops, our mechanism provides significant enhancement by retransmitting much less than the other algorithms. This is the result of less collisions between data and ACK packets, given the fewer amount of ACKs generated by our mechanism. The conservative mechanism for the retransmissions by timeout in the TCP-DAA sender contributes to such achievements as well. In these evaluations, TCP-DAA retransmits 72 to 100% less than the regular TCP and 65 to 70% less than the LDA scheme. This outcome is doubtlessly expressive in terms of energy consumption benefits, as shown in the next section.

7.4.5 Energy Efficiency

TCP-DAA is expected to be energy saving as it minimizes spurious retransmissions. In this section we evaluate the performance benefits of TCP-DAA in terms of energy consumption, as depicted in Fig. 7.12. We used the simple energy model implemented in the ns-2 simulator that has been presented in [XHE00]. By this model, a node starts with an initial energy level that is reduced whenever the node transmits, receives or overhears a packet. Thus, the total amount of energy, E(n_i), consumed at a given node n_i is given by (7.7).

$$E(n_i) = E_{tx}(n_i) + E_{rx}(n_i) + (N-1) * E_0(n_i) \tag{7.7}$$

Where E_{tx}, E_{rx}, and E_0 denote the amount of energy expenditure by transmission, reception, and overhearing of a packet, respectively. N represents the average number of neighbors nodes affected by a transmission from node n_i [KGLAO$^+$02].

In order to account only for the reception and transmission expenditure, we have discarded the energy spent by overhearing (E_0). This is appropriate to highlight the energy due to TCP transmissions and receptions. Fig. 7.12(a) shows the result of a simulation run in which 10 flows share the medium in the chain topology of Fig. 7.8(a) for different number of hops. The figure exhibits the energy consumption per bit at the TCP sender. This is computed as $e = \frac{pkt*pkt\_size*8}{e\_spent}$, where e is the energy/bit ratio, pkt is the amount of packet transmitted by the sender, pkt_size is the packet size in bytes and e_spent is the energy in Joule spent by the sending node.

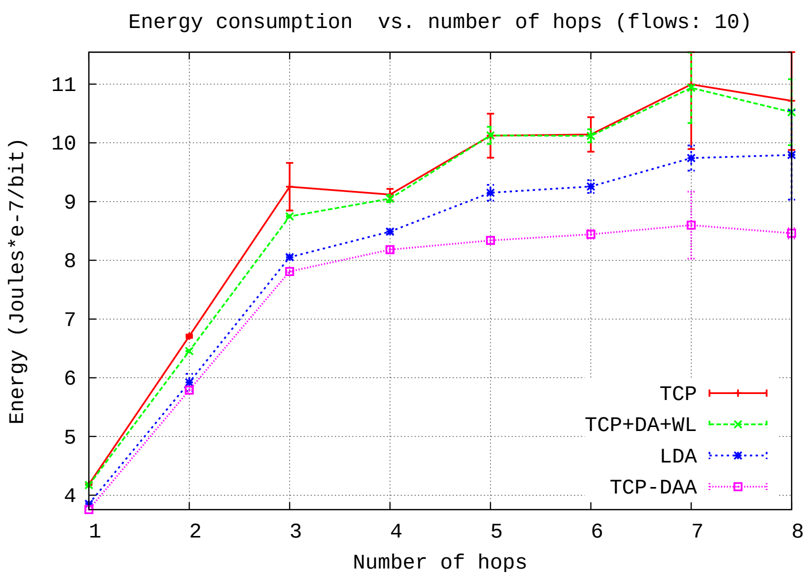
One can see in Fig. 7.12(a) that TCP-DAA provides the best result over all situations. The performance enhancement is more noticeable at large number of hops, where the probability of collisions is higher. This happens because our algorithm minimizes the number of packets in transit. As a result, less collisions occur leading to fewer retransmissions and consequently higher energy saving. In these simulations, TCP-DAA provided an improvement of up to 18% over the regular TCP. Alternatively, one can say that the regular TCP spent about 26% more energy than our scheme.

We also evaluated the impact of the packet size on TCP energy consumption as shown in Fig. 7.12(b). In this simulation, four different packet sizes are evaluated, namely packets of 256, 512, 1000, and 1460 bytes long. Sender and receiver are connected through 4 hops. The results show that the smaller the packet the higher the energy consumption. This is intuitive because with small packets TCP needs to process more packets to transmit the same amount of data than it does when using larger packet sizes. Fig. 7.12(b) also shows that in most cases, but for packet size of 256 bytes, TCP-DAA spent less energy than all the other configurations. The difference was not very significant though. Thus, in this scenario packet size does not seem to impact energy consumption significantly.

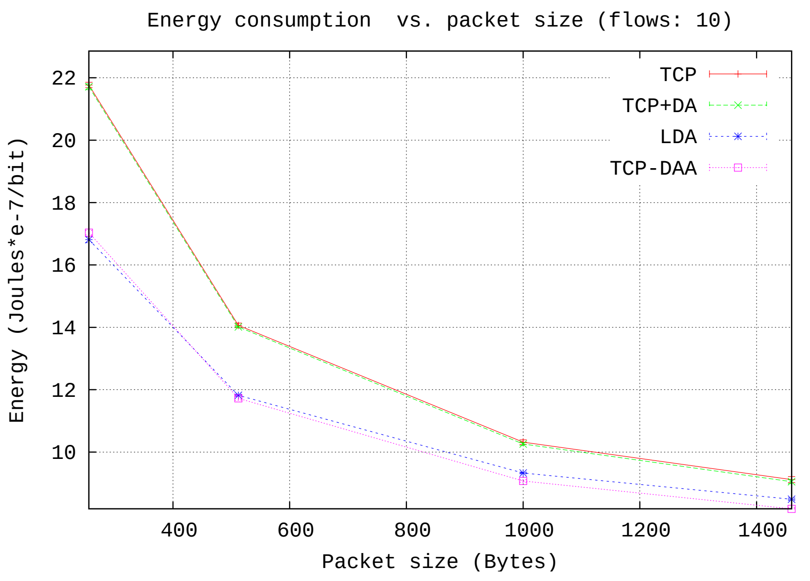
7.4.6 Short-lived Flows

An important aspect of approaches that defer acknowledgments at the receiver is their response to short-lived flows. The duration of such flows are usually not sufficient to lead the congestion control mechanism rapidly to an equilibrium. The problem is that TCP is an ACK-clocked mechanism as its sender needs ACKs to clock out data. TCP-DAA addresses this issue by using the startup speed factor μ which defines how aggressive the sender may be during its startup. In this section, we evaluate the performance of our mechanism for short-lived flows.

For the sake of clarity, we consider a scenario without the hidden node problem. Fig. 7.13 illustrates the bandwidth achieved for a run over the chain topology of Fig. 7.8(a). In this run, sender and receiver communicate over 3 hops, and different values of μ are simulated for a duration of 40 seconds. The curves plotted in the figure represent the



(a) Effect of number of hops



(b) Effect of packet size

Figure 7.12: Energy consumption at the TCP sender

bandwidth achieved by a single flow. The graphs are computed with a granularity of 0.3 seconds, and accumulated over the whole simulated interval, i.e., gradually accumulated from 0 to 40 seconds by steps of 0.3 seconds.

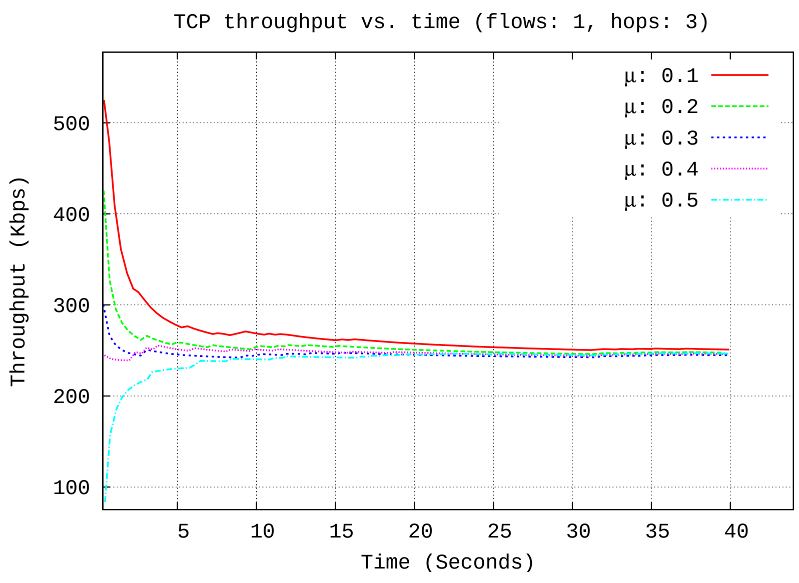
The results point out that smaller values of μ render better performance at startup than large values. As the session duration gets longer, the difference among the distinct values of μ become negligible, as shown clearly in Fig. 7.13(a). The same convergence would be noticed in Fig. 7.13(b), where 10 flows share the medium, if a longer simulation duration had been performed. The main outcome here is that short-lived connections should use low values for μ toward faster transfers.

In order to guarantee that the receiver does not lead the sender to miss ACKs at startup, the value of μ should then be in the range (0,0.5), i.e., $0 < \mu < 0.5$. Having μ=0.5 causes the receiver to increase its $dwin$ too quickly as follows. The sender first transmits two data packets (sender's initial window is set to 2). The receiver defers the acknowledgment of the first data packet because its $dwin$ (initialized with 1) is first increased by 0.5 (μ value) and so becomes 1.5 which is greater than the current ack_count that is equal to 1. By receiving the next data packet, $dwin$ reaches size 2, and so does ack_count. As ack_count is then as large as $dwin$, the receiver acknowledges the second data packet. When the sender gets such an ACK, it expands its $cwnd$ to 3, and transmits three new packets. These packets are going to be delayed at the receiver, but they are not enough to make ack_count as large as $dwin$. The three packets drive ack_count to 3, and $dwin$ to 3.5.

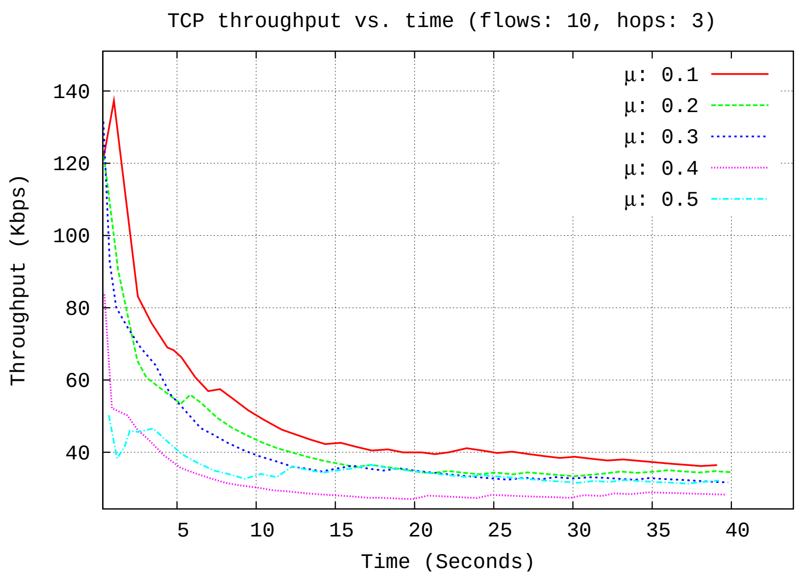
As a result, the receiver has to wait for its timer expiration before sending the buffered ACK, and reducing $dwin$ to 2. We use a conservative initial timeout interval of 200 ms for avoiding spurious timeout during startup, primarily for congested scenarios. Hence, if TCP-DAA increases its $dwin$ too fast, it may not wait long enough for the transmission channel to be full of packets for delaying the ACKs. This may, in the worst case, trigger a retransmission by timeout at the sender. Fig. 7.13(a) illustrates how the curve for μ=0.5 performs much poorly than the others at the initial instants of the simulation time. Higher values of μ experience the same problem.

Since the μ value is not very significant for long-lived flows, we simply used a value of 0.3 in the previous evaluations for providing some tolerance in case of dropped packets at the very beginning of the simulation. Smaller values may not be appropriate for scenarios with many concurrent flows, because the smaller the μ the more packet exchanges, which may induce excessive drops during startup.

To further illustrate the performance of our algorithm for short-lived flows, we conducted simulations in which only short files are transferred over the chain topology. The results are depicted in Fig. 7.14. In the evaluation shown in Fig. 7.14(b), a single flow crosses seven hops, the file size ranges from 10 to 500 Kbytes, and the startup speed factor μ is set to 0.1. Fig. 7.14(a) highlights that except for the regular TCP (with or without DA) and the LDA scheme, all the other implementations perform roughly the same. Short-lived flows indeed do not seem to benefit from our proposed mechanism. If the scenario is too much congested by many parallel flows, TCP-DAA may experience worst performance than the other flavors, given the lack of time it faces to reach equilibrium. This is shown in Fig. 7.14(b) for five concurrent flows.

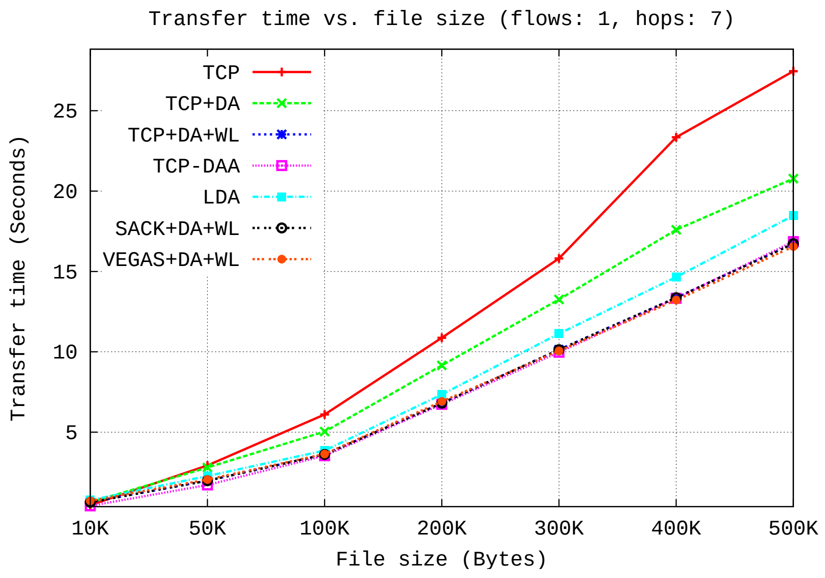


(a) 1 Flow

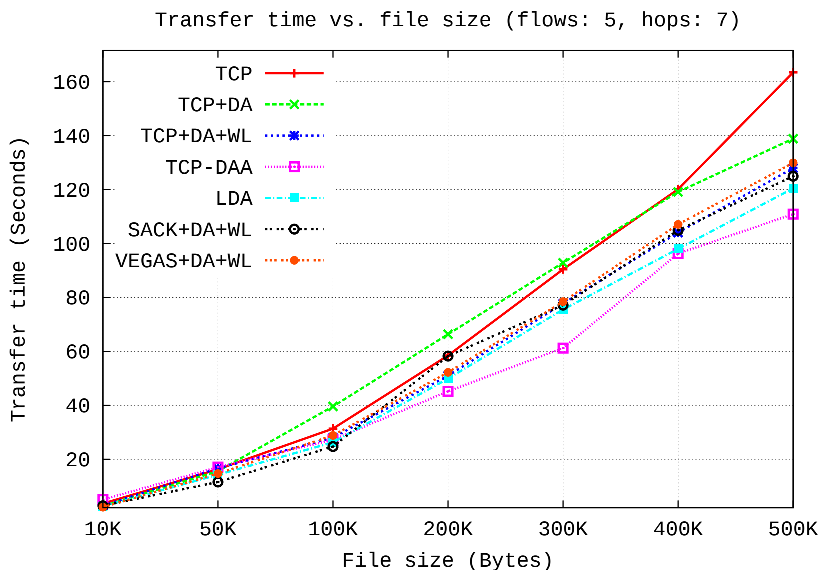


(b) 10 Concurrent flows

Figure 7.13: Accumulated throughput of 1 flow for short-lived flows



(a) 1 Flow



(b) 5 Concurrent flows

Figure 7.14: Transfer time for short-lived flows

The results confirm that by being too conservative, TCP-DAA does not perform well for very short file sizes as it does not speed up their transfer time. Such a time, in Fig. 7.14(b), refers to the time needed to fully transfer the slowest flow among the five concurrent ones. In these simulations, TCP-DAA does not provide any measurable benefit for file sizes of 10, 50 and 100 Kbytes. In fact, it performs slightly worse than the other flavors for the very short file size of 10 Kbytes. On the other side, for file of size bigger than 100 Kbytes, it may provide some benefit. The enhancements are not very significant, though. In short, to be optimal for short-lived flows, our mechanism needs a more aggressive strategy at the sender. However, there will be always a tradeoff between short and long-lived flows optimizations.

7.4.7 Optimization:TCP-DAAp

We investigate in this section the optimization proposed in section 7.3.6 (TCP-DAAp) in which the $dwin$ strategy of the sender is supposed to be more robust to environments facing non negligible losses. Upon losses the receiver reduces $dwin$ to one and slowly increases it again to prevent the receiver timer from expiring by lack of data packets. The analytical evaluation in section 7.3.6 showed that following a very conservative procedure, $dwin$ should increase by about 0.28 for each in-order data packet received. The simulation results illustrated in Fig. 7.15 conforms closely to the analytical prediction.

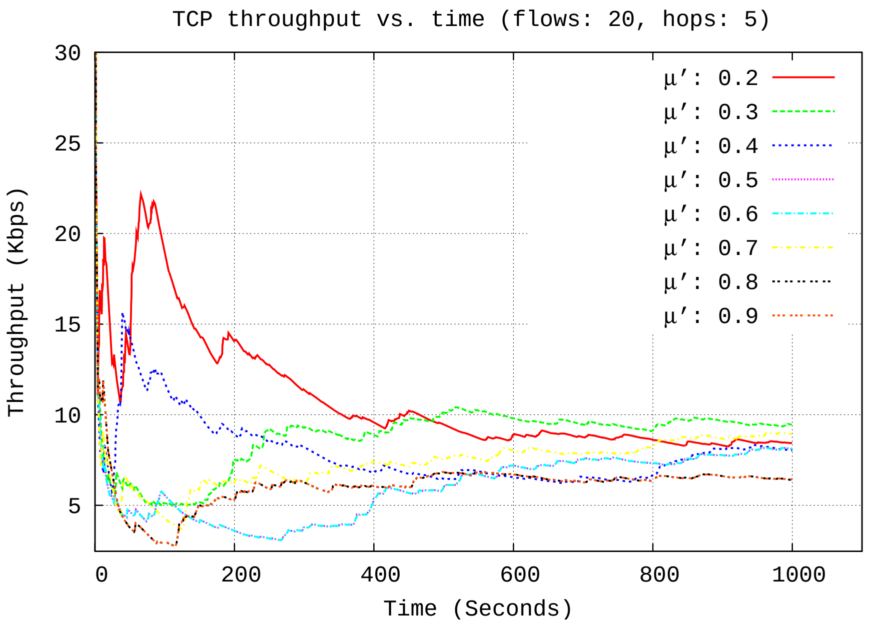


Figure 7.15: Optimal μ' parameter for TCP-DAAp

These evaluations were conducted over the chain topology and each run lasted one thousand seconds. Each curve indicates the throughput of a single flow out of twenty competing flows in a 5-hop scenario. Since the scenario is quite constrained in this case, the sender retransmission timeout (RTO) must not be too tolerant as in the previous case. Hence, its tolerance was decreased from fivefold to twofold to conform with the concept in section 7.3.6. Various values for the μ' parameter were simulated. Despite the varying behavior of the curves, one can see that $\mu' = 0.2$ and $\mu' = 0.3$ tend to provide optimal performance.

It is expected that TCP-DAAp does not provide the same improvements shown in Fig. 7.9. This happens because of two reasons: TCP-DAAp transmits more ACKs than the basic version, and its sender is more aggressive concerning retransmissions. Fig. 7.16 exhibits the comparisons between the two TCP versions for the same conditions in section 7.4.2. The figure shows that the robustness to losses comes at the cost of throughput under moderate conditions. Nevertheless, the changed algorithm performs as effective as the regular TCP in Fig. 7.9(a).

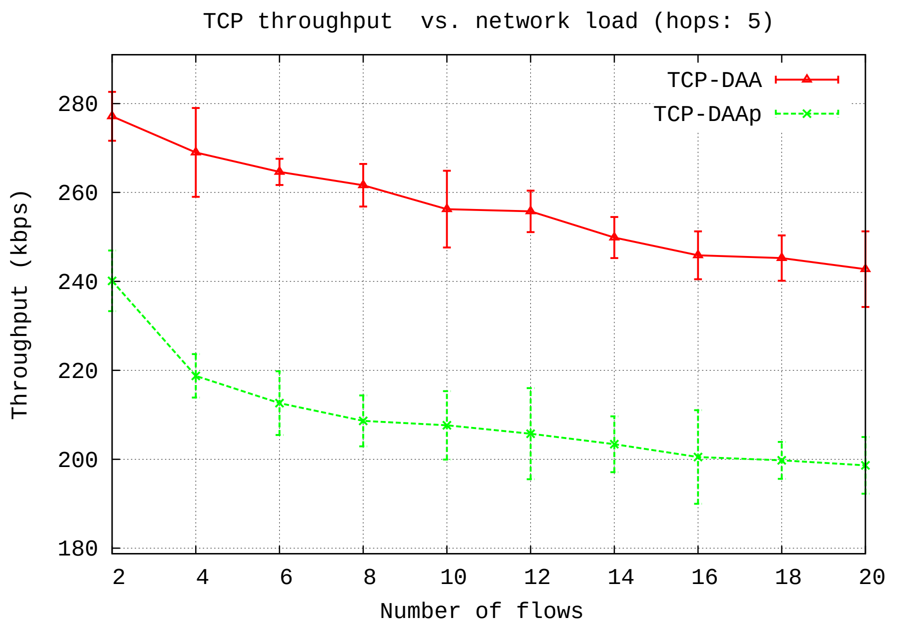


Figure 7.16: Comparison between the two TCP-DAA versions under moderate loss rate

The justification for the TCP-DAAp strategy is to render our strategy as robust as the regular TCP mechanism under heavily constrained environments. TCP-DAA is not optimized to such environments, and because of that it degrades substantially under high loss rates. Fig. 7.17 highlights the importance of TCP-DAAp in a scenario where just a single flow crosses a 5-hop chain of nodes under varying packet error rates. This is a very noisy scenario where not only losses due to MAC collisions are in place but also losses induced by a permanent external disturbance. The error model used follows a uniform distribution function. The results in Fig. 7.17 shows that indeed our strategy can handle losses in an effective way since it performs as effective as the TCP+DA+WL version. Although it is not shown here, we affirm that these results are even better over the regular TCP without any further adjustment.

The discussions in the two paragraphs above suggest that it is helpful to have an additional monitoring mechanism at the receiver to adjust TCP-DAA strategy on the basis of the channel condition. This procedure along with an improved TCP sender, regarding the RTO computation, can surely render our proposal very robust in a wide range of scenarios. Using such a mechanism, TCP-DAA would be invoked under moderate loss rate and TCP-DAAp would take over when the channel condition deteriorated. This is left for potential future work.

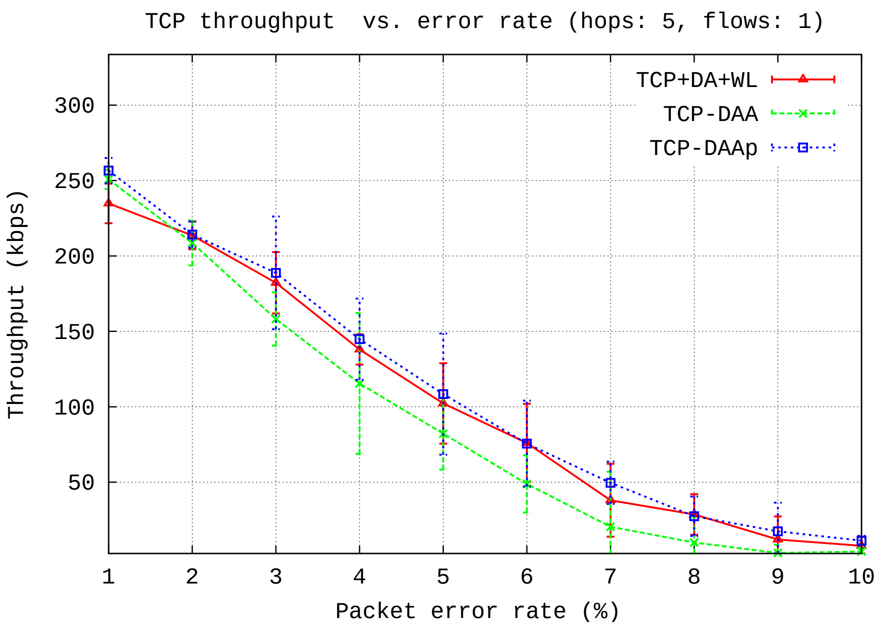


Figure 7.17: TCP-DAAp provides robustness for highly noisy scenarios

7.4.8 TCP Friendliness

Gradual deployment requires acceptable friendliness behavior when TCP-DAA is sharing the medium with other flows. This means that our mechanism should not suppress regular flows but allow them to achieve at least the same throughput they would obtain without any improved flow in parallel. We show here how friendly TCP-DAA can be when competing with regular flows in a multihop channel facing moderate loss rate.

Fig. 7.18 depicts the result of a simulation run in which two flows of distinct versions share the medium. Namely, a TCP-DAA flow competes with a regular TCP that uses DA and window limit (WL). It is clear that our mechanism outperforms the adjusted regular TCP in the span of 8 hops. The difference between both protocols is noticeable for 1 to 3 hops, where the hidden node problem does not happen. After that, more collisions take place and both mechanisms performs quite the same. One can say that TCP-DAA performs very aggressively against the optimal regular flow for the cases of 1, 2 and 3 hops, being upmost for the 1-hop case.

To measure the degradation imposed by our mechanism over the other flow, we include the “Reference” curve in Fig. 7.18. This curve refers to the adjusted regular flow (TCP+DA+WL) performance when no modified flow is in place. In other words, this reference curve is obtained when a regular flow is sharing the medium with another regular flow. Let this be the reference curve hereafter. The throughput of the regular flow for the 1-hop run in Fig. 7.18 is 469 Kbps. The corresponding throughput for the reference curve is 681 Kbps. This shows an unfairness of our mechanism for this scenario, which leads the regular flow to a decrease in throughput of up to 31%.

Another experiment is shown in Fig. 7.19 where the number of hops are fixed at three and distinct amount of flows are simulated. Since this is a scenario without the hidden node problem, the number of collisions are not very high. Note that as the number of flows rise our mechanism degrades performance leaving more bandwidth to the regular flow. Like the previous case, TCP-DAA induces performance degradation to the regular flow. In this case, the regular flow would achieve about 223 Kbps of throughput if only regular flows were being transmitted, but it obtains only 175 Kbps. This means a reduction in throughput of approximately 23%.

The results above suggest that TCP-DAA needs a sort of pacing for controlling its sending rate in mixed scenarios involving non-TCP-DAA flows. A possible mechanism for that is proposed in [All98] in which the authors propose to limit the amount of packets sent at once by the sender to two packets. This comes at the cost of the end-to-end bandwidth utilization, though.

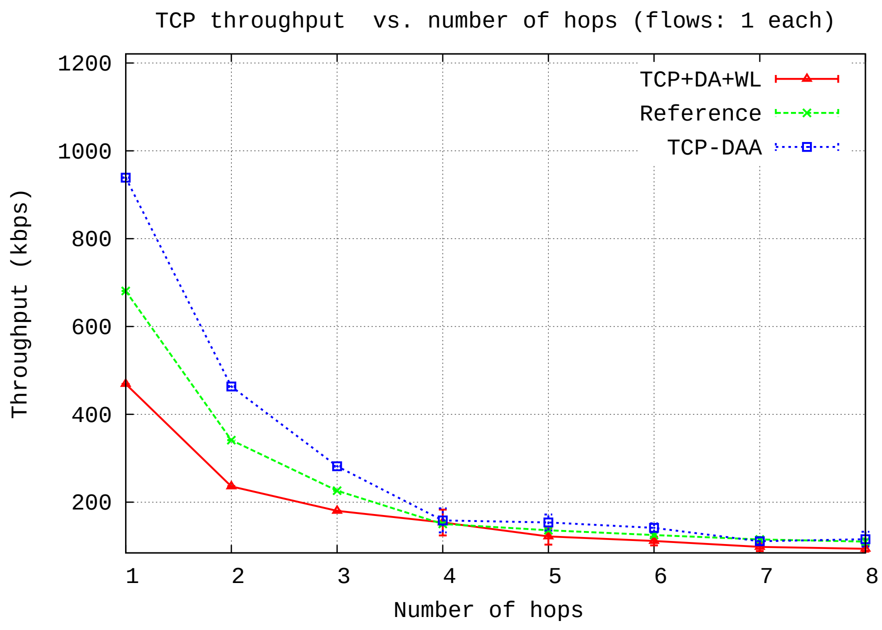


Figure 7.18: TCP-DAA friendliness

To extend the discussion above to a more lossy scenario facing the hidden node problem effects, we conducted simulations for a 5-hop chain topology. The results are de-

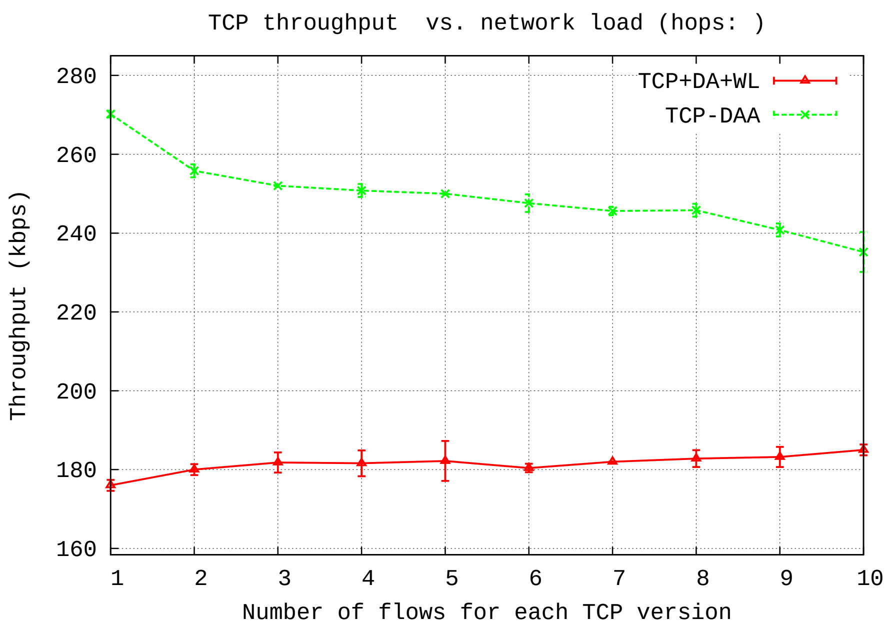


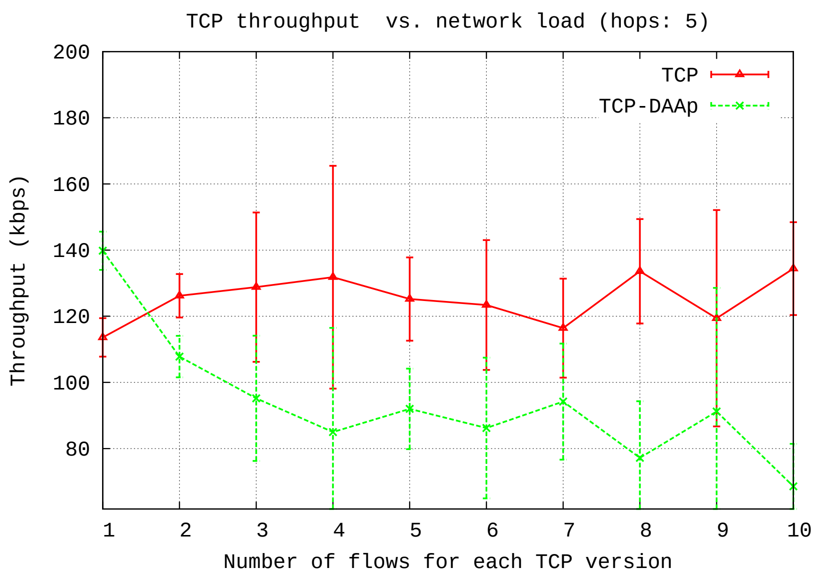
Figure 7.19: Aggregate throughput for two distinct versions in parallel

picted in Fig. 7.20. The results in Fig. 7.20(a) are obtained with the optimization described in section 7.3.6 for the receiver side only. The sender side was kept as in the basic mechanism where the timeout interval RTO is increased fivefold. The purpose of this experiment is to highlight that although the receiver is optimized to deal with losses more effectively, if the sender is too tolerant to invoke its retransmission mechanism its performance degrades. Fig. 7.20(a) shows that the degradation increases as a function of the number of hops.

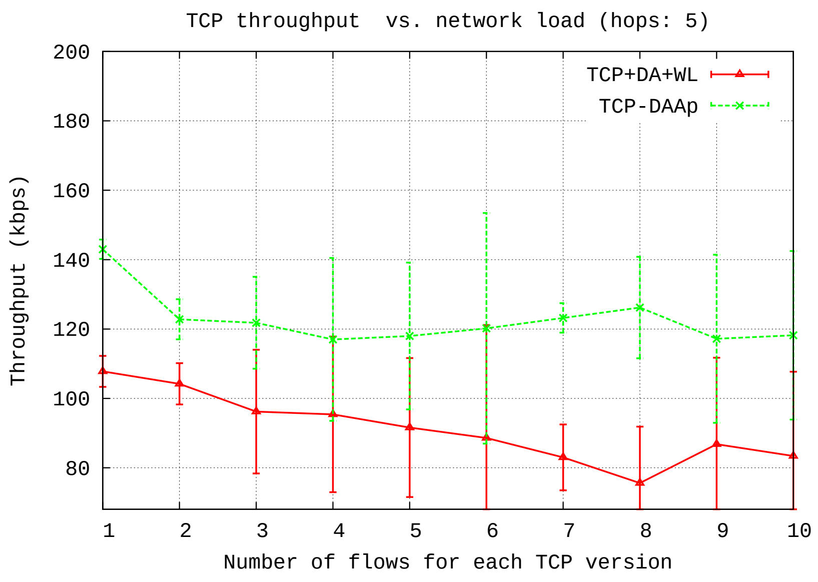
A more aggressive response by the sender is thus fundamental as shown in Fig. 7.20(b). Here both sender and receiver are optimized to deal with frequent packet losses. The sender uses an RTO limit increased by twofold only and the receiver increases its $dwin$ in a slower pace. As a result, TCP-DAAp may compete more efficiently with the regular flow. In this case, our mechanism achieves higher performance than the regular flow. A more balanced bandwidth distribution can surely be achieved if the involved parameter are fined tuned. The key point here is that the degradation of the regular flow is not very pronounced. So our mechanism can indeed be adjusted in such cases toward friendly behavior.

7.4.9 Discussions

The general perception is that our mechanism is indeed valuable to multihop networks. The results presented here support our claim that a dynamic and adaptive mechanisms is effective in such complex environments. The results in section 7.4.7 indicate that the mechanism can be refined to handle highly constrained conditions. Distinct parameter



(a) TCP-DAAp with high delay tolerance at the sender



(b) TCP-DAAp with low delay tolerance at the sender

Figure 7.20: Aggregate throughput for two distinct versions in parallel

settings for moderate and elevated constraints are needed, though. The key remark here is that our mechanism automatically uses available bandwidth and does not perform worse than a regular TCP when traffic conditions deteriorate.

As far as friendly behavior is concerned, our basic mechanism is not very friendly in scenarios without the hidden node problem. This is a weakness of our technique, which calls for further investigation. At first, a rate limitation at the sender may mitigate such a problem. On the other side, TCP-DAA parameters may be optimized under hidden node problem effects to render the protocol friendly. This optimization includes the sender side, which has not yet been fully investigated.

Short-lived flows seem not to benefit from our mechanism. These sort of flows last too shortly, so TCP-DAA does not reach an equilibrium. Nevertheless, our changes do not impair the performance of these flows. As we expected, by reducing unnecessary retransmissions at the sender, our mechanism minimizes energy consumption as well. Throughput improvements are very significant in the chain of nodes topology, but not really effective in the grid topology. The grid topology considered experiences elevated loss rate and so TCP-DAA benefits is not achieved. Using 802.11 as the MAC protocol, we believe that it is very difficult to improve performance for such kind of scenarios where many flows interact with each other. Changes at the MAC layer are indeed necessary.

7.5 Summary

This chapter introduced and evaluated our dynamic adaptive acknowledgment strategy to improve TCP performance over short multihop wireless networks on an end-to-end basis. Our technique mimics the sender congestion control in that it uses the available channel bandwidth dynamically as soon as it becomes available. When the channel is in good shape, our mechanism transmits minimal amount of ACKs for improving bandwidth utilization. Under constraints, the proposed mechanism promptly reacts by transmitting more acknowledgments in order to mitigate disruptions at the sender side.

The outcome of the performance evaluation is encouraging, especially for scenarios facing moderate loss rates. Considerable gains in terms of throughput, retransmissions, and energy consumption were observed in a variety of scenarios. Further improvements seem to be possible by fine tuning the parameters of the presented algorithm, as well as by enhancing the sender side to handle the effects of delayed ACKs smoothly.

We certainly do not claim that our technique is the optimal acknowledgment strategy for a TCP implementation in multihop networks. Rather, we believe that various improvements may be added to the mechanisms presented in this chapter. The most important ones, from our standpoint, are presented in the next chapter as proposal for future work. The contributions included in this chapter are above all expected to be a sound foundation for further developments on TCP implementations tailored to multihop wireless networks.

Chapter 8

Conclusions and Outlook

Wireless technologies are foreseen as the predominant means in future network communications. These wireless networks approximate the the user to an old mankind's dream of communication anywhere, anyhow, and at anytime. The success of wireless communications depends, however, on their ability to use the TCP/IP Suite protocol smoothly. This is a very important research topic nowadays since many issues remain to be addressed. This thesis has identified key TCP problems in multihop networks and proposed solutions. This chapter summarizes the problems addressed in the thesis, revises the proposed solutions, draws conclusions, and gives directions for future work.

Challenges and Solutions

Reliable data transfer over multihop wireless networks is one of the most difficult tasks to be accomplished in this promising framework. Traditional transport protocols like TCP face severe performance degradation over multihop networks given the noisy nature of wireless media as well as unstable connectivity conditions in place. The research community has been seeking ways to improve TCP over such networks largely because of the various applications that have been developed to this protocol over the years. Interoperability with the Internet is another aspect that motivates extending TCP to multihop networks.

As discussed in chapter 5, TCP was initially designed to work in wired networks. These networks rely on communication channels that experience generally very low bit error rates, typically much less than 1%. The communicating nodes in a wired network are normally fixed, i.e., these nodes do not change location often. Moreover, these traditional networks count on reasonable bandwidth resource to delivery data. In contrast, wireless networks do not encompass any of these characteristics fully. As a consequence, TCP experiences substantial performance degradations in multihop wireless networks. This thesis proposes solutions to two key problems faced by TCP in such networks: association of losses to congestion, and lack of a proper acknowledgment management.

TCP associates dropped packets to network congestion. As a result, whenever a lost packet is perceived TCP reduces its transmission rate to alleviate the presumed congestion inside the network. This is effective in wired networks, which has rendered TCP the *de facto* standard protocol for reliable data transport in the Internet. On the other hand, congestion is not the only reason for packet loss in multihop networks. Rather, these wireless networks are prone to much higher bit error rates due to the medium nature. Hence, a conventional TCP may waste precious bandwidth by reducing its transmission rate when reacting to a single drop caused by a random noise rather than by congestion. This problem calls for a mechanism at the sending node to determine the actual reason of a dropped packet. Being aware of the nature of the loss allows the sender to react properly. This thesis has proposed a mechanism to perform packet loss discrimination at the sender, which is addressed below.

The other aspect covered in this thesis is proactive instead of reactive as is the case in the discussion above. The acknowledgment strategy used by TCP poses a high burden on shared wireless media using the IEEE 802.11 standard, as shown in chapter 7. In addition to the random backoff mechanism that imposes random backoff intervals to sequentially unsuccessful transmission attempts, 802.11 uses the short RTS/CTS control frames for any transmission request. This leads a TCP acknowledgment to cause similar MAC overhead of a data packet, despite the much smaller ACK size. Furthermore, the data packets of a TCP connection have to contend with the returning ACKs from the receiver, increasing the probability of collisions. As wireless media rely on very scarce bandwidth, the amount of ACK injected into the network should be managed carefully. Thank to the cumulative acknowledgment scheme used in TCP, it is possible to reduce the amount of ACKs transmitted by a TCP connection. However, this has to be carried out in an adaptive way because ACKs are fundamental to the correct operation of TCP. Our proposed solution to this problem is a dynamic adaptive acknowledgment approach that is presented below.

Packet Loss Discrimination

Chapter 6 has introduced and evaluated our proposed mechanism to distinguish between packet loss caused by congestion and due to the medium. The general concepts of a TCP error detection mechanism based on fuzzy logic has been introduced as well. Our mechanism is tailored to short multihop networks, which is the only one feasible today given the limitation of the IEEE 802.11 standard.

The proposed technique uses fuzzy logic to perform packet loss discrimination. The designed fuzzy engine makes inference on RTT measurements to decide whether a given loss is error or congestion induced. Fuzzy logic allows intelligent systems to be implemented in a very lightweight fashion. As a result, low processing overhead at the sender is possible, extending the lifetime of the battery powered devices. As only passive measurements are needed for the inference process, it is a completely end-to-end solution. This is advantageous to leverage deployment since no explicit intermediate node cooperation is necessary.

Simulation evaluations show that the proposed fuzzy engine is promising in the target scenarios. Provided that the input data, gathered from the measurements, follow a given pattern the fuzzy engine can provide efficient discrimination. RTT mean and variance are useful to match the network condition to a well-defined pattern. In steady state scenarios, where abrupt changes does not occur too often, the proposed engine is quite robust. This makes this mechanism quite appropriate to scenarios facing consistent loss rates.

A Dynamic Adaptive Acknowledgment Strategy

In chapter 7, our proposed mechanism to reduce ACK induced overhead and consequently collisions has been introduced and evaluated. The central idea behind this mechanism lies in the observation that while the TCP sender actively interacts with the network to use its changing bandwidth efficiently, the receiver is merely a responsive mechanism to the sender transmissions. The receiver plays no significant role in the end-to-end connection other than the straight acknowledgment of incoming data packets to the sender. Our proposal adds functionalities to the TCP receiver, so it can also interact with the network to alleviate losses and consequently improve the end-to-end performance adaptively.

Our dynamic adaptive acknowledgment approach minimizes the number of ACKs transmitted by a TCP receiver by actively monitoring the channel condition and adapting to it. The sender keeps track of the packet inter-arrival interval to adjust its timeout timer properly on the basis of the ongoing network status. The packets sequence number are also monitored to speed up loss detection dynamically. The receiver combines from one to four ACKs into a single ACK saving precious bandwidth. Thus, the receiver delays ACKs in a fully dynamic and adaptive fashion similarly to the sender congestion control. A key feature of our scheme is to work independently of the channel data rate. This is also an end-to-end solution that dispenses with changes in the intermediate nodes.

Simulation evaluations using the ns2 network simulation show very good results in a variety of scenarios. The proposed mechanism improves not only bandwidth utilization but also energy consumption. Less retransmissions at the sender and less overall ACK processing are the reasons for energy saving at the sender. The basic mechanism of our scheme targets scenarios facing moderate loss rates. Simulation results indicate that improvements are needed for highly lossy environments. Because of that, an alternative response to lossy channels were developed and the evaluations show that it is efficient. The proposed mechanisms can be combined to make the changed receiver feasible in both unconstrained and constrained situations.

Lessons Learned

This section summarizes a few general remarks that might be useful for future researches in the field of this thesis.

The complexity involved in the modeling of protocols for today's networks is extremely high. As a result, most work based on analytical analysis have to make strong assumptions so that the modeling becomes possible. The problem is that this practice generally renders the results unrealistic and so these approaches are mostly not deployed. Heuristic approaches are easier to design and are usually based on intuition and observable real life events. However, a pure heuristic approach is usually not optimized since it lacks full knowledge of the system which it represents. Hence, the most prominent solutions combine heuristics with basic analytical models. The traditional TCP is a typical example of such approaches. Some researchers have been trying to model this protocol precisely, but no such modeling has yet been achieved or deployed.

Simulation reliability is another important subject that has been discussed in the research community. Some researchers do not trust simulation evaluations. From our experience, we believe that simulation can be as useful as real testbeds. An important point to be noticed here is that the simulators parameters have to be set realistically. This ensures that simulation results represent real life systems' behaviors very closely.

The last observation we want to comment on here concerns the degree of accuracy in typical evaluation results. Many related work conduct experiments either by simulation or real measurements and draw conclusions based on a single run. From our experience, we indeed know that most measurements have to be averaged over several runs for reliability purposes. Only a single run can really mask the results toward a completely wrong trend compromising the conclusions.

Outlook

The proposals in this thesis improve performance of a traditional TCP in a variety of scenarios. However, there are some limitations as well as pending evaluations that can be pursued for future work. We outline next some obvious extensions of the present work.

Fine tuning the Fuzzy Engine: The fuzzy engine proposed in chapter 6 was designed with a very simple set of membership functions. More accuracy appears to be possible if not only more fuzzy sets are used but also other membership functions are taken. We used a uniform distribution for the error pattern, which we believe ensures a worst case scenario. However, different pattern distributions could be evaluated to determine to what extend they may influence the engine efficiency. Moreover, since the amount of ACKs tends to decrease substantially as the number of hops increase, it seems that a way to shrink the computing window is needed in these cases.

Automatic Setting of the Fuzzy Engine: The fuzzy engine was evaluated for a fixed number of hops only. Different number of hops impose distinct RTT ranges, which implies in specific parameters for each number of hops. In addition, the RTT characteristic varies with error pattern. To accelerate deployment, the fuzzy engine settings have to be adaptive to the network conditions dynamically. A genetic algorithm can be used to monitor the RTT parameters and adjust the engine parameters on the fly.

A Customized Sender for the Smart Acknowledgment: Our proposed dynamic adaptive acknowledgment strategy is currently focused on the receiver side. It is well-known that the sender relies on ACKs for computing its timeout interval and transmit new data packets. Since our mechanism at the receiver delays ACKs dynamically, it can disturb the sender performance by inducing spurious retransmissions degrading the end-to-end performance. In order to evaluate our changes at the receiver, we simply increased the sender

RTO fivefold and decreased the threshold to trigger the fast retransmit mechanism from three to two packets. This is appropriate for moderate loss rate but too conservative for high loss rates. Therefore, a comprehensive investigation on the sender algorithm to deal with the side effects of the dynamically delayed ACKs is needed.

Interoperability of the Smart Acknowledgment with the Internet: Our proposed strategy is completely customized to multihop wireless networks. Users might find it useful to participate in such networks and at the same time communicate with the Internet. In such cases, it is important to a TCP implementation in multihop networks to be able to establish connections to the Internet. A gateway between the two networks is necessary to establish such interoperation. We believe that some TCP functionalities in these gateways are needed to connect these two networks such as the such as the ones proposed in I-TCP [BB95]. To investigate such functionalities, as well as the changes required in our technique represent indeed an interesting extension of our work.

Combination of the Smart Acknowledgment with the Fuzzy Engine: The smart acknowledgment improves performance by being proactive in minimizing collisions, while the fuzzy engine is reactive because it detects loss. It seems to be promising the combination of both mechanisms. This is, however, not a simple task since the changes in the number of acknowledgments generated by the receiver disturb the RTT patterns perceived by the fuzzy engine. Hence, full investigation starting by detecting the resulting patterns is suggested for future work.

Evaluation of the Dynamic Adaptive Acknowledgment in a Testbed: Even though we have evaluated our proposal extensively through simulations, its implementation in a testbed is surely of interest. To evaluate our mechanism with real traffic is definitely a good contribution to be performed. We believe that the configuration parameters of our scheme might be fine tuned by considering real traffic features.

List of Figures

List of Tables

List of Abbreviations and Acronyms

AP	Access Point
ACK	Acknowledgment
AODV	Ad-hoc On-demand Distributed Vector
AQM	Active Queue Management
ATCP	Ad hoc TCP
BER	Bit Error Rate
CF	Contention-Free
CFP	Contention-Free Period
CP	Contention Period
CRC	Cyclic Redundancy Check
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CTS	Clear to Send
CW	Contention Window
CWR	Congestion Window Reduced (CWR)
cwnd	Congestion Window
DAA	Dynamic Adaptive Acknowledgment
DAAp	Dynamic Adaptive Acknowledgment plus
DCF	Distributed Coordination Function
DNS	Domain Name Server
DS	Distributed System
DSR	Dynamic Source Routing
ECN	Explicit Congestion Notification
ELFN	Explicit Link Failure Notification
FCS	Frame Check Sequences
FEDM	Fuzzy-based Error Detection Mechanism
FILO	First In Last Out
FTP	File Transfer Protocol
HMM	Hidden Markov Model
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
IDD	Inter-packet Delay Difference
IED	Improved Error Detection
IEEE	Institute of Electrical and Electronics Engineers

IETF	Internet Engineering Task Force
IFS	Interframe Space
IP	Internet Protocol
ISN	Initial Sequence Number
LAN	Local Area Network
LDA	Large Delayed Acknowledgment
LRED	Link RED
MAC	Medium Access Control
MSS	Maximum Segment Size
ndup	Number of duplicate ACKs (threshold)
NH	Number of Hops
OSI	Open Systems Interconnect
PC	Point Coordinator
PCF	Point Coordination Function
PER	Packet Error Rate
PLR	Packet Loss Rate
POR	Packet Out-of-order delivery Ratio
RED	Random Early Detection
RFN	Route Failure Notification
RR	RTT increase Rate
RRN	Route Re-establishment Notification
RREP	Route Reply
RREQ	Route Request
RTO	Retransmit Timeout
RTS	Request to Send
RTT	Round Trip Time
RTTVAR	Round Trip Time Variation
rwin	Receiver Window
rx	Reception
SMTP	Mail Transfer Protocol
SRTT	Smoothed Round Trip Time
STT	Short Term Throughput
SWS	Silly Window Syndrome
TCP	Transport Control Protocol
TTL	Time to Live
tx	Transmission
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Network
WWW	World Wide Web

Bibliography

[ABF01] M. Allman, H. Balakrishman, and S. Floyd. Enhancing TCP's Loss Recovery Using Limited Transmit. RFC 3042, IETF Network Working Group, January 2001.

[Abr85] N. Abramso. Development of the ALOHANET. *IEEE Transactions on Information Theory*, IT-31:pages 119–123, March 1985.

[All98] M. Allman. On the generation and use of TCP acknowledgements. *ACM Computer Communication Review*, 28:pages 1114–1118, 1998.

[APS99] M. Allman, V. Paxson, and W. Stevens. Transmission Control Protocol. RFC 2581, IETF Network Working Group, April 1999.

[ASSC02] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A Survey on Senser Networks. *IEEE Communications Magazine*, 40:pages 102–114, August 2002.

[AWSC02] I. F. Akyildiz, Y. S. W. Su, and E. Cyirci. Wireless Sensor Networks: A Survey. *Computer Networks (Elsevier)*, 38(4):pages 393–422, March 2002.

[BB95] A. Bakre and B. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. In *Proceedings of IEEE ICDCS'95*, pages 136–143. Vancouver, Canada, May 1995.

[BCC$^+$98] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, G. Minshal, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Z. and. Recommendations on Queue Management and Congestion Avoidance in the Internet. RFC 2209, IETF Network Working Group, April 1998.

[BMJ$^+$98] J. Broch, D. Maltz, D. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom '98)*. Dallas, October 1998.

[BMP94] L. S. Brakmo, S. W. O. Malley, and L. Peterson. New Techniques for Congestion Detection and Avoidance. In *Proceedings of ACM SIGCOMM*, pages 24–35. London, UK, October 1994.

[BPS$^+$97] H. Balakrishnan, V. Padmanabhan, S. Seshan, M. Stemm, and R. Katz. TCP Behavior of a Busy Web Server: Analysis and Improvements. Technical report, UCB/CSD-97-966, August 1997.

[Bra89] R. Braden. Requirements for Internet Hosts – Communication Layers. RFC 1122, IETF Network Working Group, October 1989.

[BS97] K. Brown and S. Singh. M-TCP: TCP for Mobile cellular Networks. *ACM Computer Communications Review*, 27:pages 19–43, 1997.

[BSAK95] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz. Improving TCP/IP Performance over Wireless Networks. In *Proceedings of 1st ACM Mobicom*. Vancouver, Canada, November 1995.

[BV98] S. Biaz and N. H. Vaidya. Distinguishing Congestion Losses from Wireless Transmission Losses:A Negative Result. In *Proceedings of IEEE 7th Int. Conf. on Computer Communications and Networks*. New Orleans, LA, USA, October 1998.

[CGL00] A. Chandra, V. Gummalla, and J. Limb. Wireless Medium Access Control Protocols. *IEEE Communications Surveys*, 39(1):pages 02–15, Second Quarter 2000.

[CJ89] D. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17(1):pages 1–14, June 1989.

[CJ03] T. Clausen and P. Jacquet. ptimized Link State Routing Protocol (OLSR). RFC 3626, IETF Network Working Group, October 2003.

[CK03] C. Chong and S. Kumar. Sensor Networks: Evolution, Opportunities, and Challenges. *IEEE*, 91(8):pages 1247–1256, August 2003.

[Cla82] J. P. Clark. Window and Acknowledgment Strategy in TCP. RFC 813, IETF Network Working Group, july 1982.

[CM99] S. Corson and J. Macker. Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations. RFC 2501, IETF Network Working Group, January 1999.

[CM01] L. Cheng and I. Marsic. Fuzzy Reasoning for Wireless Awareness. *International Journal of Wireless Information Networks*, 8(1):pages 15–26, January 2001.

[Cox94] E. Cox. *The Fuzzy Systems Handbook: A Practitioner's Guide to Building, Using, and Maintaining Fuzzy Systems*. Academic Press, 1994.

[CRVP98] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash. A Feedback Based Scheme For Improving TCP Performance In Ad-Hoc Wireless Networks. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS'98)*. Amsterdam, The Netherlands, June 1998.

[CSH$^+$01] P. M. Chan, R. Sheriff, Y. F. Hu, P. Confort, and A. Spazio. Mobility Management Incorporating Fuzzy Logic for a Heterogeneous IP Environment. *IEEE Communications Magazine*, 39(12):pages 42–51, December 2001.

[CWKS97] B. Crown, I. Widjaja, J. Kim, and P. Sakai. IEEE 802.11 Wireless Local Networks. *IEEE Communications Magazine*, 39(1):pages 116–126, September 1997.

[CXN03] K. Chen, Y. Xue, and K. Nahrstedt. On Setting TCP's Congestion Window Limit in Mobile Ad Hoc Networks. In *Proceedings of IEEE International Conference on Communications (ICC 2003)*. Anchorage, Alaska, May 2003.

[Dar81] Darpa. Transmission Control Protocol. RFC 793, IETF Network Working Group, April 1981.

[DB01] T. D. Dyer and R. Boppana. A Comparison of TCP Performance over Three Routing Protocols for Mobile Ad Hoc Networks. In *ACM Symposium on Mobile Ad Hoc Networking and Computing - Mobihoc*. Long Beach, USA, October 2001.

[DLJL00] D. Dumitrescu, B. Lazzerini, L. C. Jain, and B. Lazzerini. *Fuzzy Sets and their Application to Clustering and Training*. CRC Press, 2000.

[DVRA04] A. Dunkels, T. Voigt, H. Ritter, and J. Alonso. Distributed TCP Caching for Wireless Sensor Networks. In *Proceedings of The Third Annual Mediterranean Ad Hoc Networking Workshop (MedHocNet04)*. Bodrum, Turkey, June 2004.

[EA03] T. J. E. Altman. Novel delayed ACK techniques for improving TCP performance in multihop wireless networks. In *Personal Wireless Communications (PWC'03)*. Venice, Italy, September 2003.

[EFF$^+$00] D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Ya, and H. Yu. Advances in Network Simulation. *Computer*, 33(5):pages 59–67, May 2000.

[EGHK99] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *Proceedings of 5th ACM/IEEE Mobicom Conference (MobiCom)*, pages 263–270. Seattle, USA, August 1999.

[EHH$^+$00] D. Estrin, M. Handley, J. Heidemann, S. S. McCanne, X. Ya, and H. Yu. Network Visualization with Nam, the VINT Network Animator. *Computer*, 33(11):pages 63–68, November 2000.

[FF96] K. Fall and S. Floyd. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. *Computer Communication Review*, 26(3), July 1996.

[FGML02] Z. Fu, B. Greenstein, X. Meng, and S. Lu. Design and Implementation of a TCP-Friendly Transport Protocol for Ad Hoc Wireless Networks. In *Proceedings of 10th IEEE International Conference on Network Protocosls (ICNP'02)*. November 2002.

[FH99] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 3782, IETF Network Working Group, April 1999.

[FHAG04] S. Floyd, T. Henderson, and E. A. Gurtov. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 3782, IETF Network Working Group, April 2004.

[FZL$^+$03] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla. The Impact of Multihop Wireless Channel on TCP Throughput and Loss. In *Proceedings of Infocom'03*. San Francisco, USA, April 2003.

[GK04] N. Gupta and P. R. Kumar. A Performance Analysis of the 802.11 Wireless Lan Medium Access Control. *Communications in Information and Systems*, 3(4):pages 279–304, September 2004.

[GTB99] M. Gerla, K. Tang, and R. Bagrodia. Evaluation for TCP with Delayed ACK Option in Wireless multi-hop Networks. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*. New Orleans, Louisiana, USA, February 1999.

[Haa02] Z. J. Haas. Wireless Ad Hoc Networks. In J. Proakis, editor, *Encyclopedia of Telecommunications*. John Wiley, 2002.

[HBG00] U. Hengartner, J. Bolliger, and T. Gross. TCP Vegas Revisited. In *Proceedings of IEEE INFOCOM 2000*. Tel Aviv, Israel, March 2000.

[HCB00] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy Efficient Routing Protocols for Wireless Microsensor Networks. In *Proceedings of 33rd Hawaii International Conference on System Sciences (HICSS)*. Hawaii, January 2000.

[HFPW03] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 3448, IETF Network Working Group, January 2003.

[HKB99] W. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. In *Proceedings of 5th ACM/IEEE Mobicom Conference (MobiCom)*. Seattle, USA, August 1999.

[HMM99a] G. Hasegawa, M. Murata, and H. Miyahara. Fairness and Stability of Congestion Control Mechanisms of TCP. In *Proceedings of IEEE INFOCOM*, pages 1329–1336. New York, March 1999.

[HMM99b] G. Hasegawa, M. Murata, and H. Miyahara. Fairness and Stability of the congestion control mechanism of TCP. In *in Proceedings of IEEE INFOCOM'99*. New York, USA, March 1999.

[HPS02] Z. J. Haas, M. R. Pearlman, and P. Samar. The Zone Routing Protocol (ZRP) for Ad Hoc Networks. Internet-draft, IETF MANET Working Group, July 2002. Draft-ietf-manet-zone-zrp-04.txt.

[HV99a] G. Holland and N. Vaidya. Analysis of tcp performance over mobile ad hoc networks. In *Proceedings of ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99)*. Seattle, USA, August 1999.

[HV99b] G. Holland and N. Vaidya. Impact of Routing and Link Layers on TCP Performance in Mobile AD Hoc Networks. In *Proceedings of ACM/IEEE Wireless Communication Networks Conference (IEEE WCNC 1999)*. New Orleans, USA, September 1999.

[IEE99] IEEE. Standard 802.11-1999, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. In *The Institute of Electrical and Electronics Engineers*. 1999.

[Jac88] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of ACM SIGCOMM*, pages 314–329. Stanford, CA,, August 1988.

[Jac90] V. Jacobson. Modified TCP Congestion Avoidance Algorithm, 1990.

[Jan93] J. Jang. ANFIS: Adaptive-Network-based Fuzzy Inference Systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(3):pages 665–685, May 1993.

[Jan01] J. Jantzen. Tutorial On Fuzzy Logic. Technical report, Technical University of Denmark, August 2001.

[JBB92] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. RFC 1323, IETF Network Working Group, May 1992.

[JM96] D. Johnson and D. Maltz. Dynamic source routing in ad hoc wireless networks. In T. Imielinski and H. Korth, editors, *Mobile Computing*, chapter 5, pages 153–181. Kluwer Academic, 1996.

[JMB01] D. Johnson, D. Maltz, and J. Broch. DSR The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks. In C. Perkins, editor, *Ad Hoc Networking*, chapter 5, pages 139–172. Addison-Wesley, 2001.

[JMH04] D. B. Johnson, D. A. Maltz, and Y. Hu. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR). Internet-draft, IETF MANET Working Group, July 2004. Draft-ietf-manet-dsr-10.txt.

[Joh95] S. R. Johnson. *Increasing TCP Throughput by Using an Extended Acknowledgment Interval*. Master's thesis, Ohio University, USA, June 1995.

[JT87] J. Jubin and J. Tornow. The DARPA Packet Radio Network Protocols. *Proceedings of IEEE*, 75:pages 21–32, January 1987.

[KGLAO$^+$02] D. Kim, J. Garcia-Luna-Aceves, K. Obraczka, J. Cano, and P. Manzoni. Power-Aware Routing Based on The Energy Drain Rate for Mobile Ad Hoc Networks. In *IEEE International Conference on Computer Communication and Networks(ICCCN2002)*. Miami, USA, October 2002.

[KKP00] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Emerging Challenges: Mobile Networking for Smart Dust. *Journal of Communications and Networks*, 2(3):pages 188–196, September 2000.

[KP87] P. Karn and C. Partridge. Improving Round-Trip Time Estimates in Reliable Transport Protocols. *Computer Communication Review*, 17(5):pages 2–7, August 1987.

[Kul01] A. Kulkarni. *Computer Vision and Fuzzy-Neural Systems*. Prentice Hall PTR, May 2001.

[LAS01] A. Lindgren, A. Almquist, and O. Schelén. Quality of Service Schemes for IEEE 802.11 - A Simulation Study. In *Proceedings of Ninth International Workshop on Quality of Service (IWQoS 2001)*. Karksruhe, Germany, June 2001.

[LBC$^+$01] J. Li, C. Blake, D. S. J. D. Couto, H. I. Lee, and R. Morris. Capacity of Ad Hoc Wireless Network. In *Proceedings of ACM MOBICOM'01*. Rome, Italy, July 2001.

[LS01] J. Liu and S. Singh. ATCP: TCP for Mobile Ad Hoc Networks. *IEEE Journal on Selected Areas in Communications*, 19(7):pages 1300–1315, july 2001.

[Mat02] MathWorks. Fuzzy Logic Toolbox User's Guide, July 2002.

[MCGW01] S. Mascolo, C. Casetti, M. Gerla, and M. S. andR. Wang. TCP Westwood: End-to-End Bandwidth Estimation for Efficient Transport over Wired and Wireless Networks. In *Proceedings of MobiCom*. Rome, Italy, July 2001.

[MMR96] M. Mathi, J. Mahdavi, and S. F. A. Romanow. TCP Selective Acknowledgement Options. RFC 2018, IETF Network Working Group, October 1996.

[MPC01] J. P. Macher, V. D. Park, and S. Corson. Putting the Pieces Together. *IEEE Communication Magazine*, 39(6):pages 148–155, June 2001.

[MSK02] J. Merwe, S. Sen, and C. Kalmanek. Streaming Video Traffic: Characterization and Network Impact. In *Proceedings of Seventh International Web Content Caching and Distribution Workshop*. Boulder, CO, USA, August 2002.

[MSM$^+$97] M. Mathis, J. Semke, J. Mahdavi, , and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM Computer Communication Review*, 27(3):pages 67–82, July 1997.

[OB02] R. Oliveira and T. Braun. TCP in Wireless Mobile Ad Hoc Networks,. Technical report, TR-002-003, University of Bern, July 2002.

[OB03] R. Oliveira and T. Braun. An Edge-based Approach for Improving TCP in Wireless Mobile Ad Hoc Networks. In *Proceedings of Design, Analysis and Simulation of Distributed Systems 2003 (DASD03), Part of the 2003 Advanced Simulation Technologies Conference (ASTC 2003)*. Orlando, USA, March 2003.

[OB04a] R. Oliveira and T. Braun. A Delay-based Approach Using Fuzzy Logic to Improve TCP Error Detection in Ad Hoc Networks. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC04)*. Atlanta, USA, March 2004.

[OB04b] R. Oliveira and T. Braun. A Dynamic Adaptive Acknowledgment Strategy for TCP over Multihop Wireless Networks. Technical report, TR-004-005, University of Bern, July 2004.

[OB04c] R. Oliveira and T. Braun. A Fuzzy Logic Engine to Assist TCP Error Detection in Wireless Mobile Ad Hoc Networks. In *Proceedings of Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN'04)*. St.Petersburg, Russia, February 2004.

[OB05] R. Oliveira and T. Braun. A Dynamic Adaptive Acknowledgment Strategy for TCP over Multihop Wireless Networks. In *Proceedings of IEEE INFOCOM 2005*. Miami, USA, March 2005.

[OG01] R. Oliveira and P. Guardieiro. A comparative Study of TCP Reno and TCP Vegas in a Differentiated Services Network. In *Proceedings of 3rd Conference on Telecommunication (Conftele2001)*. Figueira da Foz, Portugal, April 2001.

[OTL04] R. Ogier, F. Templi, and M. Lewis. Topology Dissemination Based on Reverse-Path Forwarding (TBRPF). RFC 3684, IETF Network Working Group, February 2004.

[PA00] V. Praxon and M. Allman. Computing TCP's Retransmission Timer. RFC 2988, IETF Network Working Group, November 2000.

[PB94] C. Perkins and P. Bhagwat. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of ACM SIGCOMM Conference on Communications Architectures, Protocols and Applications*, pages 234–244. August 1994.

[PB01] C. Perkins and P. Bhagwat. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of IEEE International Conference on Communications (ICC)*, pages 278–283. Helsinki, Finland, June 2001.

[PBRD03] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561, IETF Network Working Group, December 2003.

[PC97] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of IEEE INFOCOM*. April 1997.

[PFTK98] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. A simple Model and its Empirical Validation. In *Proceedings of ACM SIGCOMM'98*. Vancouver, Canada, September 1998.

[PH02] D. Perkins and H. Hughes. Investigating the performance of TCP in Mobile Ad Hoc Networks. *International Journal of Computer Communications*, 25(11-22):pages 1132–1139, July 2002.

[PK00] G. Pottie and W. Kaiser. Wireless Integrated Network Sensors. *Communications of the ACM*, 43(5):pages 51–58, MAY 2000.

[Pos81] J. Postel. Internet Protocol. RFC 791, IETF Network Working Group, September 1981.

[PR99] C. Perkins and E. Royer. Ad-hoc On-Demand Distance Vector routing. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*. New Orleans, USA, February 1999.

[RFB01] K. Ramakrisgnan, S. Floyd, and D. Balck. The addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, IETF Network Working Group, September 2001.

[SAA03] Y. Sankarasubramaniam, O. B. Akan, and I. F. Akyildiz. ESRT: Event-to-Sink Reliable Transport in Wireless Sensor Networks. In *Proceedings of ACM MobiHoc03*. Maryland, USA, June 2003.

[SGAP00] K. Sohrabi, J. Gao, V. Ailawadhi, and G. Pottie. Protocols for Self-organization of a Wireless Sensor Network. *IEEE Personal Communications*, 7(5):pages 16–27, October 2000.

[SH03] F. Stann and J. Heidemann. RMST: Reliable Data Transport in Sensor Networks. In *Proceedings of 1st IEEE International Workshop on Sensor Net Protocols and Applications (SNPA)*. Alaska, USA, May 2003.

[Ste94] W. R. Stevens. *TCP/IP Illustrated Volume 1*. Addison Wesley, 1994.

[Ste97] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001, IETF Network Working Group, January 1997.

[TAGH02] S. Tilak, N. Abu-Ghazaleh, and W. Heinzelman. A Taxonomy of Wireless Micro-Sensor Network Models. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(2):pages 28–36, April 2002.

[TB00] V. Tsaoussidis and H. Badr. TCP-Probing: Towards an Error Control Schema with Energy and Throughput Performance Gains. In *Proceedings of 8th IEEE Conference on Network Protocols*. Japan, November 2000.

[TCG01] K. Tang, M. Correa, and M. Gerla. Effects of Ad Hoc MAC Layer Medium Access Mechanisms Under TCP. *ACM/Bal MONET*, 6(4):pages 317–329, 2001.

[TG99] K. Tang and M. Gerla. Fair sharing of MAC under TCP in wireless ad hoc networks. In *Proceedings of IEEE Multiaccess, Mobility and Teletrac for Personal Communications (MMT'99)*. Venice, Italy, October 1999.

[TM02] V. Tsaoussidis and I. Matta. Open issues on TCP for Mobile Computing". *The Journal of Wireless Communications and Mobile Computing*, 2(1):pages 1–14, February 2002.

[TMW97] K. Thompson, G. Miller, and R. Wilder. Wide-Area Internet Traffic Patterns and Characteristics. *IEEE Network*, 11(6):pages 10–23, November 1997.

[TO04] C. Tschudin and E. Osipov. Estimating the Ad Hoc Horizon for TCP over IEEE 802.11 Networks. In *3rd Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net 2004)*. Bodrum, Turkey, June 2004.

[WATC02] C. Wan and L. K. A. T. Campbell. PSFQ: A Reliable Transport Protocol For Wireless Sensor Networks. In *Proceedings of First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002)*. Atlanta, USA, September 2002.

[WKST04] B. Wang, J. Kurose, P. Shenoy, and D. Towsley. Multimedia streaming via TCP: An analytic performance study. In *Proceedings of ACM Multimedia*. October 2004.

[WPL$^+$02] H. Wu, Y. Peng, K. Long, S. Cheng, and J. Ma. Performance of Reliable Transport Protocol over IEEE 802.11 Wireless LAN: Analysis and Enhancement. In *Proceedings of INFOCOM 02*. New York, USA, June 2002.

[WZ02] F. Wang and Y. Zhang. Improving TCP Performance over Mobile Ad-Hoc Networks with Out-of-Order Detection and Response. In *ACM Mobihoc*. Lausanne, Switzerland, June 2002.

[XHE00] Y. Xu, J. Heidemann, and D. Estrin. Adaptive EnergyConserving Routing for Multihop Ad Hoc Networks. In *USC/Information Sciences Institute*. Research Report 527, October 2000.

[XS01a] S. Xu and T. Saadawi. Does the IEEE 802.11 MAC protocol work well in multihop wireless ad hoc networks? *IEEE Communications Magazine*, 39(6):pages 130–137, june 2001.

[XS01b] S. Xu and T. Saadawi. Does the IEEE 802.11 MAC protocol work well in multihop wireless ad hoc networks? *IEEE Communications Magazine*, 39:pages 130–137, june 2001.

[XS01c] S. Xu and T. Saadawi. Evaluation for TCP with Delayed ACK Option in Wireless multi-hop Networks. In *Proceedings of IEEE MILCOM2001*. Vienna, Virginia, USA, October 2001.

[YYS$^+$04] T. Yuki, T. Yamamoto, M. Sugano, M. Murata, H. Miyahara, and T. Hatauchi. Performance Improvement of TCP over an Ad Hoc Network by Combining of Data and ACK packets. *to apper in IEICE Transactions on Communications*, 2004.

[Zad65] L. Zadeh. Fuzzy Sets. *Journal of Information and Control*, 8:pages 338–353, 1965.

[Zad96] L. Zadeh. Fuzzy Logic = Computing with Words. *IEEE Transactions on Fuzzy Systems*, 4(2):pages 103–111, May 1996.

[ZT01] C. Zhang and V. Tsaoussidis. TCP-Probing: Towards an Error Control Schema with Energy and Throughput Performance Gains. In *Proceedings of 11th IEEE/ACM NOSSDAV*. New York, June 2001.

Curriculum Vitae

1967	Born on May 14, in Cuiabá-MT, Brazil
1975-1978	Primary School, Cuiabá-MT, Brazil
1979-1982	Secondary School, Cuiabá-MT, Brazil
1983-1986	Technician in Electronics at the Technical Vocational High School (ETFMT), Brazil
1988-1992	BSc in Electrical Electronics Engineering at the Federal Engineering School of Itajubá (EFEI), Brazil
1993	Safety Engineering at the Federal University of Mato Grosso, Brazil
1999-2001	MSc in Computer Science at the University of Uberlândia, Brazil
2001-2005	Research and lecture assistant and Ph.D. student at the University of Berne.