Department of Information Engineering, University of Pisa, Italy
Ph.D. Thesis - XIX Cycle

MANETs: Internet Connectivity and Transport Protocols

Ph.D. Candidate
Emilio Ancillotti

Advisor
Prof. Giuseppe Anastasi

April 2007

.

List of Publications

Journals

- E. Ancillotti and R. Bruno and M. Conti and E. Gregori and A. Pinizzotto. *A Layer-2 Framework for Interconnecting Ad Hoc Networks to Fixed Internet: Test-bed Implementation and Experimental Evaluation.* Computer Journal, To appear.
- E. Ancillotti, G. Anastasi, M. Conti and A. Passarella. A Comprehensive Study of TPA: a Transport Protocol for Ad hoc Networks, *submitted to The Computer Journal, Special Issue on Performance Evaluation of Wireless Networks.*

Conferences

- E. Ancillotti, G. Anastasi,M. Conti and A. Passarella, Experimental analysis of a transport protocol for ad hoc networks (TPA), *Proceedings of the ACM International Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN 2006)*, October 2 2006, Torremolinos (Spain), ACM Press
- E. Ancillotti, R. Bruno, M. Conti, E. Gregori and A. Pinizzotto. A Layer-2 Architecture for Interconnecting Multi-hop Hybrid Ad Hoc Networks to the Internet. *In Proceedings of WONS 2006.* January, 18–20 2006. Les Menuires, France. pp. 87–96.
- G. Anastasi, E. Ancillotti, M. Conti and A. Passarella. TPA: A Transport Protocol for Ad Hoc Networks. *ISCC '05: Proceedings of the 10th IEEE Symposium on Computers and Communications (ISCC'05).* Cartagena (Spain). June 27-30 2005. pp. 51–56. IEEE Computer Society.
- E. Ancillotti and R. Bruno and M. Conti and E. Gregori and A. Pinizzotto. Experimenting a Layer 2-based Approach to Internet Connectivity for Ad Hoc Networks. *IEEE ICPS Workshop on Multi-hop Ad hoc Networks (REALMAN 2005).* July 14, 2005. Santorini (Greece).

Book Chapters

- E. Ancillotti, G. Anastasi, M. Conti and A. Passarella, Experimental Analysis of TCP Performance in Static Multi-hop Ad Hoc Networks.

Chapter 6 in Mobile Ad Hoc Networks: from Theory to Reality, Nova Science Publisher.
Also available at http://www2.ing.unipi.it/ o1653499/papers.htm.

- E. Ancillotti and R. Bruno and M. Conti and E. Gregori and A. Pinizzotto. Implementation and Experimentation of a Layer-2 Architecture for Interconnecting Heterogeneous Ad Hoc Network to the Internet. *chapter in Mobile Ad Hoc Networks: from Theory to Reality, (M. Conti, J. Crowcroft, A. Passarella, Editors).* Nova Science Publisher.

- E. Ancillotti, G. Anastasi, M. Conti and A. Passarella. Design, Implementation and Measurements of a Transport Protocol for Ad Hoc Networks. *Chapter in MobileMAN (M. Conti, Editor), Sprinter.* To appear.
  Also available at http://www2.ing.unipi.it/ o1653499/papers.htm.

- E. Ancillotti and R. Bruno and M. Conti and E. Gregori and A. Pinizzotto. A MobileMAN Approach for the Interconnection of Heterogeneous Ad Hoc Networks to the Internet. *chapter in MobileMAN (M. Conti, Editor)*, Springer. To appear.

Acknowledgments

This thesis is the outcome of about three years (since January 2004) of research work at the Department of Information Engineering at the University of Pisa and at the Institute for Informatics and Telematics of the National Research Council of Pisa. 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 me.

First I want to express my gratitude to my advisor, Prof. Giuseppe Anastasi. His trust in my work even when things seemed not to be on track was fundamental for my success. His advices always helped me to overcame the difficulty that I encountered during my research work. Professor Giuseppe Anastasi also gave me the opportunity to being part of a team of people working at the Institute for Informatics and Telematics of the National Research Council of Pisa. This team includes Dr. Marco Conti, Ing. Enrico Gregori, Ing. Raffaele Bruno, Ing. Andrea Passarella, Ing. Franca DelMastro, Ing. Eleonora Borgia, Ing. Luciana Pelusi, and Ing. Antonio Pinizzotto. I express a special appreciation to Dr. Marco Conti, Ing. Andrea Passarella and Ing. Raffaele Bruno, that gave me a valuable contribution during my research work.

The research discussed in this dissertion contains materials previously published in papers [13, 16, 17, 18, 19, 24, 23, 21, 20, 22]. I express a special appreciation to all the co-authors for their valuable contribution that have made the success of this doctorate possible.

At the end of this page, I wish to thank the people who made it all possible: my mother, Loretta, my father, Augusto, my brother Andrea, and my girlfriend Ilaria.

Contents

Chapter 1

Introduction

1.1 Mobile Ad hoc Networks

A Mobile Ad hoc Network (MANET) is a collection of mobile nodes connected together over a wireless medium, which self-organize into an autonomous multi-hop wireless network. Nodes in ad hoc networks 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 have data to send to another node that is not in its transmission range, it uses one of its neighbors to forward the data toward the destination. This process may involve multiple intermediate nodes, and it may produce the establishing of a multihop connection (multi-hop ad hoc network) between sender and receiver, as depicted in Figure 1.1. 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. In addition, in the last few years MANETs are also emerged as a flexible and low-cost extension of wired infrastructure networks.

MANETs inherit the traditional problems of wireless communication and wireless networking, like high bit error rate, high sensitivity of wireless channel from outside signals, the possibility of path asymmetry, and so on. In addition, the multihop nature of connections, the lack of a fixed infrastructure, and nodes mobility add new problems, such as network partitions, route failures, and the hidden (or exposed) terminal. These new problems pose a number of design constraints that are specific to ad hoc networking.

Research on MANETs spanned a large number of issues. Transport protocols, roting protocols and MAC protocols are active fields of research. In the first place, this thesis examines the problem of interconnect Ad hoc network to Internet, and presents a practical architecture to logically extend traditional wired LANs using multi-hop ad hoc networking technologies. In the second place, this thesis presents the problems encountered by TCP when it operates over multi-hop ad hoc networks, and proposes a novel transport protocol for

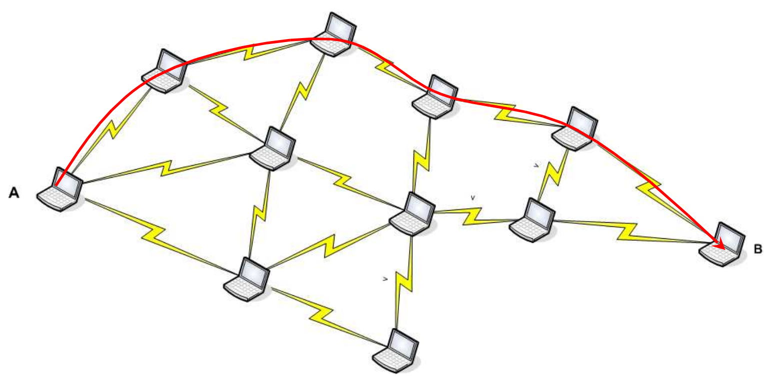


Figure 1.1: Multi Hop path between two communicating nodes.

ad hoc networks, named TPA, that is specifically tailored to the characteristics of the MANET environment. All the proposed solutions have been prototyped and evaluated in a *real* ad hoc testbed, as well as using the ns-2 simulator [3]. In the following, I briefly present the problems and the solutions that I studied during my Ph.D.

1.2 Problems Statement and Proposed Solutions

The recent advances in mobile and ubiquitous computing, and the development of inexpensive, portable devices are extending the application fields of ad hoc networking. Mobile users are looking for multi-purpose networking platforms in which cost is an issue and Internet access is a must. As a consequence, nowadays, multi-hop ad hoc networks do not appear only as isolate self-configured networks, but also emerge as a flexible and low-cost extension of wired infrastructure networks, coexisting with them. A new class of networks is emerging from this view, in which a mix of fixed and mobile nodes interconnected via heterogeneous (wireless and wired) links forms a multihop heterogeneous ad hoc network integrated into classical wired/wireless infrastructure-based networks.

Three different categories of solutions have been proposed for enabling interconnection between ad hoc networks and the Internet. One approach utilizes the Network Address Translation (NAT) mechanism, implemented on each gateway that interconnect the ad hoc network with the wired infrastructure network. With this approach, the mobile nodes do not need a globally routable IP address because the NAT gateway translates the source private IP address of outgoing traffic with a public IP address, which is routable on the fixed Internet. An alternative approach relies on the design of techniques capable of automatically configuring a unique, topology-dependent and globally routable IP address for each mobile node visiting an ad hoc network (IPv6 based solutions). Finally, a third category of solutions assumes that a Mobile IP Foreign Agent is implemented in the ad hoc nodes that act as Internet gateways. With this approach, the mobile nodes need a *permanent* and unique globally routable IP address (i.e., their home address), which is used during the registration procedures with the foreign agents of the visiting ad hoc network.

However, all the solutions that have been proposed in literature have a number of disadvantages. For example, the Mobile IP based solutions have several drawbacks. The first is that, in order to allow Mobile IP and ad hoc networking to cooperate, it is needed to introduce further complexities and sub-optimal operations in the implementations of both Mobile IP and ad hoc routing protocols. In addition, Mobile IP was designed to handle mobility of devices in case of relatively infrequent mobility. Thus, the overheads introduced to manage roaming and handoffs between foreign agents are a relevant issue in MANETs. Finally, when the technique of default routes is used to route the traffic from the mobile nodes to the closest gateway, the use of Mobile IP can easily lead to triangle routing. In fact, when the mobile node moves, it can get closer to a gateway different from the one to which it is currently registered. As a consequence, the forward traffic leaves the ad hoc network through one gateway, while the return traffic enters the ad hoc network through the new gateway to which the mobile node is registered. To solve this problem, some authors propose not to use default routes. They instead propose to use Mobile IP reverse tunnelling [69], or explicit tunnelling to one of the gateway used to leave ah hoc network. However, the tunnelling mechanism introduces a non negligible overhead in the communication between nodes.

Also the NAT based solutions have some drawbacks. For example, they encounter problems in multi-homed networks, i.e., when multiple gateways are present in the ad hoc network. Indeed, to avoid transport-layer session breaks, it is necessary to ensure that each packet from the same session is routed over a specific gateway, since a NAT router translates both outgoing and incoming packets. To solve this problem, some NAT based solutions adopts the IP-in-IP encapsulation mechanism to tunnel the packets towards the desired gateway. Employing explicit tunnelling ensures that each packet of the same transport-layer session is consistently routed through the same gateway, even if the source node moves. However, it introduces a non negligible overhead in every packet sent. In addition, NAT is not very suitable for incoming connections and this fact causes significant difficulties for peer-to-peer applications.

These observations motivated me to study an alternative, and more efficient and lightweight solution to provide Internet connectivity for ad hoc networks. Specifically, in the first part of my Ph.D. activity, I investigated how MANETs can be used to extend the range of traditional Wireless Local Area Networks (WLANs) [4] over multiple radio hops, in order to provide seamless and untethered mobility support for mobile/portables devices in the local area environment. In my work, I envisaged an heterogeneous network environment in which wired and multi-hop wireless technologies transparently coexist and interoperate (see Figure 1.2). In this network, separated groups of nodes without a direct access to the networking infrastructure form ad hoc islands, establishing multi-hop wireless links. Special nodes, hereafter indicated as gateways, having both wired and wireless interfaces, are used to build a wired backbone inter-

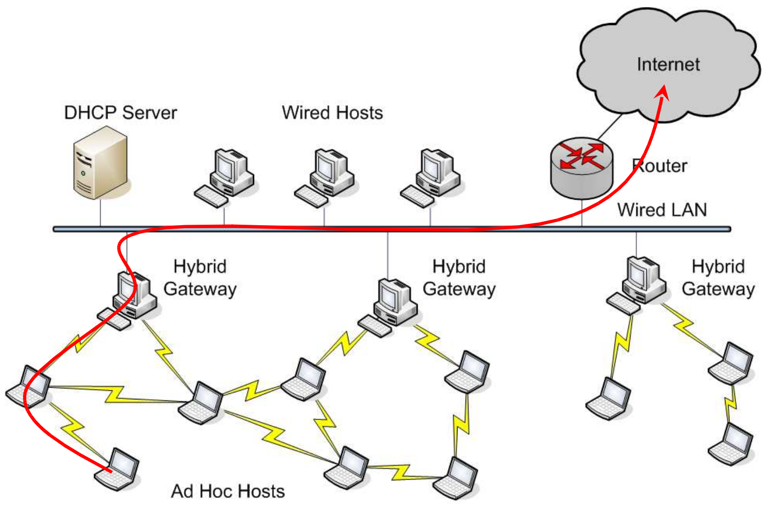


Figure 1.2: Envisaged an heterogeneous network environment.

connecting separated ad hoc components. In addition, the gateways use their wired interfaces also to communicate with static hosts belonging to a wired LAN. The network resulting from the integration of the ad hoc network with the wired LAN is an extended LAN, in which static and mobile hosts transparently communicate using traditional wired technologies, or ad hoc networking technologies.

The solution I presented during my Ph.D. [24, 23, 21, 20, 22] is a simple yet practical approach that relies only on basic ARP capabilities [103] and standard IP routing rules to logically extend a wired LAN. In addition, my solution includes a distributed protocol for the address autoconfiguration of ad hoc nodes. This protocol relies on DHCP servers located in the wired part of the network, and it does not require that new ad hoc nodes have direct access to the DHCP servers. Using my scheme, mobile nodes can dynamically obtain a unique IP address that is topologically correct within the extended LAN. During my Ph.D., I have also prototyped the main components of my architecture in a general and realistic test-bed. Using this test-bed, I have conducted a large variety of experiments, comparing the throughput performance of Internet access provided by my proposed scheme and an alternative well-known NAT-based solution [48]. The shown experimental results demonstrate that: *i*) my scheme ensures higher per-connection throughputs than the NAT-based solution, *ii*) node mobility does not cause permanent transport-layer session breaks, *iii*) node mobility induces drastic throughput degradations when using the NAT-based solution, while my proposed technique performs more efficient gateway handoffs, and *iv*) the network performances can be significantly improved by properly setting the routing protocol parameters such as to increase route stability.

In this thesis I also analyse the problems encountered by TCP when it operates over MANETs. Research on efficient transport protocols for ad hoc networks is one of the most active topics in the MANET community. Such a great interest is basically motivated by numerous observations showing that, in general, TCP is not able to efficiently deal with the unstable and very dynamic environment provided by multi-hop ad hoc networks. This is because some assumptions, in TCP design, are clearly inspired by the characteristics of wired networks dominant at the time when it was conceived. More specifically, TCP implicitly assumes that packet loss is almost always due to congestion phenomena causing buffer overflows at intermediate routers. Furthermore, it also assumes that nodes are static (i.e., they do not change their position over time). Unfortunately, these assumptions do not hold in MANETs, since in this kind of networks packet losses due to interference and link-layer contentions are largely predominant, and nodes may be mobile.

Many papers have pointed out that the drastic differences between MANETs and the legacy Internet may lead to poor performance of TCP over MANETs. However, almost all these studies rely on simulation, and many of them do not consider some important details (e.g., the routing protocol is often omitted). To the best of my knowledge, very few experimental analyses have been carried out so far [57, 71]. On the other side, previous experimental studies have shown that certain aspects of real MANETs are often not effectively captured in simulation tools [14]. Furthermore, available software and hardware products often use parameters settings different from those commonly assumed in simulation tools. Finally, real operating conditions are often different from those modeled in simulation experiments. For example, interferences caused by WiFi hotspots or other devices in the proximity are inevitable in practice. For all the above reasons, I spent some time to analyse TCP performance in a real testbed, using a network having a chain topology with variable length, and using different routing protocols [19]. My experimental outcomes are normally aligned with simulation results, and they show that TCP performance in multi hop ad hoc networks is sub-optimal and strong depends on the link quality and on the routing protocol parameters. In addition, I also found some results contrasting with simulation. Specifically, I discovered that in a real world the TCP optimal operating point moves with respect to that measured by simulation.

To address the problems experienced by TCP in MANETs, a number of proposals have been presented. The vast majority of these proposals are TCP modifications that address some particular TCP inefficiency. The main design requirement is indeed to keep the improved transport protocol backward compatible with the legacy TCP, so that “improved” and “legacy” users may be able to communicate with each other. While I acknowledge the importance of TCP compatibility, int his thesis I advocate a different approach. The differences between MANETs and traditional wired networks are so many, that TCP would need a large number of modifications to work in this environment. Consequently, it might be worth to design a transport protocol from scratch, without worrying too much – at the design stage – about backward compatibility with the legacy TCP. Interoperability with TCP could be implemented at a later stage, as a single and coherent patch to the new protocol. Following this approach, the last activity of my Ph.D. consisted in the design of a new transport protocol named Transport Protocol for Ad hoc networks (TPA) [13], specifically tailored to the MANETs characteristics. TPA is a lightweight transport protocol that provides a connection-oriented, reliable type of service. It differs from TCP in a number of ways. Specifically, the data transfer and the congestion control algorithms have been re-designed. Furthermore, TPA explicitly detects and deals with both route failures and route changes. TPA can leverage cross-layer interactions with the routing protocol, when available. For example, it is able to intercept and interpret route failure and route re-establishment messages. However, TPA works also with routing protocols that do not provide this type of information.

This thesis reports the complete description of TPA protocol. In addition, it reports the results of a performance analysis of TPA [17, 18, 16]. Differently from the most of the researches on ad hoc networks, that realize only on simulation analysis, this thesis reports the analysis of TCP and TPA performance in a real, as well as, in a simulation environment. Specifically, I used real word experiments to compare TCP and TPA performance over a wide range of reproducible network topology, like the string and the cross topology. I also used real word experiments to analyse TCP and TPA performance in a very simple but realistic mobile scenario, like the roaming node scenario. The obtained results show that TPA always outperforms TCP. Specifically, TPA throughput is between 5% and 19% greater than the TCP throughput, and, furthermore, TPA retransmits between 64% and 94% less data segments. I instead used simulative analysis to study TPA performance over simple network topologies, like the cross and the paralles topologies, and over more complex network topologies, like the grid topology and a random topology (50 nodes randomly distributed in a area A=$1000m \times 1000m$). I also analysed TPA performance over a highly dinamic environment, where 50 nodes move over a 1000 x 1000 area. The simulative show that TPA is able to achieve an increment in throughput up to 6%, and an increment in fairness up to 34% respect to TCP.

To conclude TPA study, I used a simulative analysis to study the well-known unfairness problems among concurrent connections that affects TCP as well as TPA. I integrated in TPA the Adaptive Pacing mechanism, a popular proposal for improving TCP fairness [46], and I showed that TPA with Adaptive Pacing outperforms TCP with Adaptive Pacing. In all cases I have investigated, TPA is able to improve the performance of TCP. Specifically, TPA delivers greater throughput with respect to TCP (up to 39% increase), while granting an increment in fairness up to 5.6% respect to TCP. In addiction, TPA is able to reduce the number of retransmitted segment up to 78.7%.

1.3 Structure of the document

This thesis is structured into three parts. In the first part I present background information for each of the areas I have explored. Part I includes an overview of the Transmission Control Protocol (TCP), and some of its main variants and extentions. It also includes a review of the main features of MANETs, introducing the IEEE 802.11 architecture and protocols, and providing background information on routing protocols for MANETs, with a special attention on the Ad hoc On-demand Distance Vector (AODV) protocol and on the Optimised Link State Routing (OLSR) protocol. Finally, Part I briefly describes some mechanisms and protocols used in Part II, like the Mobile IP Protocol, the Address Resolution Protocol (ARP), and the Network Address Translation (NAT) mechanism.

In the second Part, I describe the activity about the interconnection between ad hoc networks and Internet. Part II first discusses the variety of architectural issues and design options that need to be considered to interconnect ad hoc networks to fixed IP networks. Then it introduces existing approaches to tackle the internetworking of MANETs with the fixed Internet and reviews the most well known solutions. Finally, Part II describes the design principles and the protocol details of my proposed solution, and shows experimental results on the network performance in various test-bed configurations.

In the third Part, I describe my activity about the study of TCP performance over MANETs. Part III first describes the main specificity of MANETs that condition TCP behaviour and also discusses the major proposal aimed to improve TCP's performance in such environment. Then, it reports the complete description of TPA features, and also reports the description of TPA prototype implementation. Finally, Part III reports the results of the analysis of TCP and TPA performance in a real, as well as, in a simulation environment.

Part I

Preliminary Information

Chapter 2

Introduction

This Part presents preliminary information and concepts that will be used in the rest on the thesis. It is organized as follow. Chapter 5 presents an overview of the Transmission Control Protocol (TCP), and also presents some of its variants and extentions. Chapter 3 reviews the main features of MANETs, introducing the IEEE 802.11 architecture and protocols. Chapter 4 provides background information on routing protocols for MANETs, focusing special attention on the Ad hoc On-demand Distance Vector (AODV) protocol and on the Optimized Link State Routing (OLSR) protocol, that was been used in the thesis activity. Finally, Chapter 6 briefly describes some mechanisms and protocols used in the first part of this thesis, like the Mobile IP Protocol, the Address Resolution Protocol (ARP), and the Network Address Translation (NAT) mechanism.

Chapter 3

IEEE 802.11 Architecture and Protocols

This section focuses on the IEEE 802.11 architecture and protocols as defined in the original standard [2], with a particular attention to the MAC layer. The IEEE 802.11 [2] is the standard for Ad hoc networks, and it specifies both the MAC and the Physical layer. The MAC layer offers two different types of service: a contention free service provided by the *Distributed Coordination Function* (DCF), and a contention-free service implemented by the *Point Coordination Function* (PCF). These service types are made available on top of a variety of physical layers. Specifically, three different technologies have been specified in the standard: Infrared (IF), Frequency Hopping Spread Spectrum (FHSS) and Direct Sequence Spread Spectrum (DSSS).

The DCF provides the basic access method of the 802.11 MAC protocol and is based on a Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) scheme. The PCF is implemented on top of the DCF and is based on a polling scheme. It uses a Point Coordinator that cyclically polls stations, giving them the opportunity to transmit. Since the PCF can not be adopted in ad hoc mode, it will not be considered hereafter.

3.1 Distributed Coordination Function (DCF)

According to the DCF, before transmitting a data frame, a station must sense the channel to determine whether any other station is transmitting. If the medium is found to be idle for an interval longer than the *Distributed Inter-Frame Space* (DIFS), the station continues with its transmission (see Figure 3.1). To guarantee fair access to the shared medium, a station that has just transmitted a packet and has another packet ready for transmission must perform the backoff procedure before initiating the second transmission. On the other hand, if the medium is busy the transmission is deferred until the end of the ongoing transmission. A random interval, henceforth referred to as the backoff time, is then selected, which is used to initialize the backoff timer. The backoff timer is decreased for as long as the channel is sensed as idle, stopped when a transmission is detected on the channel, and reactivated when the channel is sensed as idle again for more than a DIFS. The station is enabled to transmit its frame when the backoff timer reaches zero. For example, the backoff timer of Station 3 in Figure 3.1 is disabled while Station 2 is transmitting its frame; the timer is reactivated a DIFS after Station 2 has completed its transmission. The backoff time is slotted. The backoff time is an integer number of slots uniformly chosen in the interval (0, *CW-1*). *CW* is defined as the *Backoff Window*, also referred to as *Contention Window*. At the first transmission attempt $CW = CW_{min}$, and it is doubled at each retransmission up to CW_{max}. In the standard CW_{min} and CW_{max} values depend on the physical layer adopted. For example, for the FHSS Physical Layer CW_{min} and CW_{max} values are 16 and 1024, respectively [2].

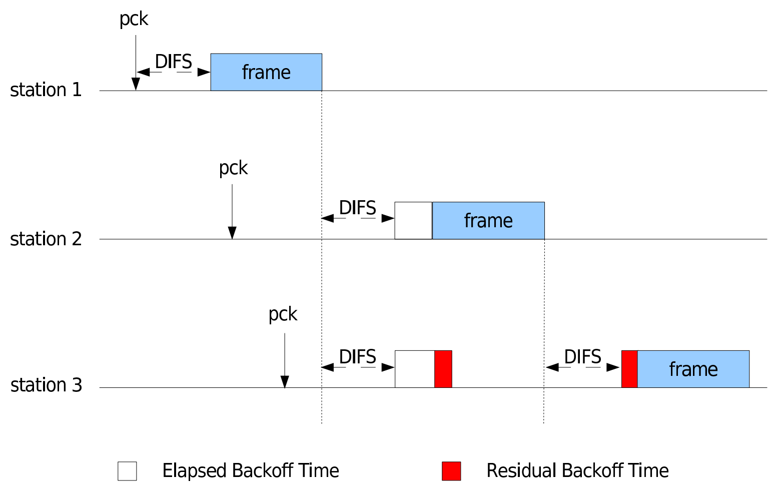


Figure 3.1: Basic Access Method.

Obviously, it may happen that two or more stations start transmitting simultaneously and a collision occurs. In the CSMA/CA scheme, stations are not able to detect a collision by hearing their own transmissions (as in the CSMA/CD protocol used in wired LANs). Therefore, an immediate positive acknowledgement scheme is employed to check the successful reception of a frame. Specifically, upon reception of a data frame, the destination station initiates the transmission of an acknowledgement frame (ACK) after a time interval called *Short InterFrame Space* (SIFS). The SIFS is shorter than the DIFS (see Figure 3.2) in order to give priority to the receiving station over other possible stations waiting for transmission. If the ACK is not received by the source station, the data frame is presumed to have been lost, and a retransmission is scheduled. The ACK is not transmitted if the received packet is corrupted. A *Cyclic Redundancy Check* (CRC) algorithm is used for error detection.

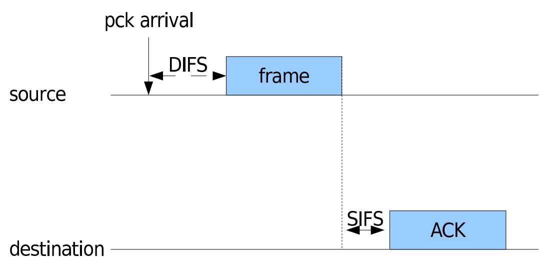


Figure 3.2: ACK generation.

After an erroneous frame is detected (due to collisions or transmission errors), a station must remain idle for at least an *Extended InterFrame Space* (EIFS) interval before it reactivates the backoff algorithm. Specifically, the EIFS shall be used by the DCF whenever the physical layer has indicated to the MAC that a frame transmission was begun that did not result in the correct reception of a complete MAC frame with a correct FCS value. Reception of an error-free frame during the EIFS re-synchronizes the station to the actual busy/idle state of the medium, so the EIFS is terminated and normal medium access (using DIFS and, if necessary, backoff) continues following reception of that frame.

3.2 Common Problems in Wireless Ad Hoc Networks

This section discusses some problems that can arise in wireless networks, mainly in the ad hoc mode. The characteristics of the wireless medium make wireless networks fundamentally different from wired networks. Specifically, as indicated in [2]:

- the wireless medium has neither absolute nor readily observable boundaries outside of which stations are known to be unable to receive network frames;
- the channel is unprotected from outside signals;
- the wireless medium is significantly less reliable than wired media;
- the channel has time-varying and asymmetric propagation properties.

In wireless (ad hoc) network that relies upon a carrier-sensing random access protocol, like the IEEE 802.11 DCF protocol, the wireless medium characteristics generate complex phenomena such as the *hidden-station* and *exposed-station* problems. Figure 3.3 shows a typical *hidden station* scenario. Let us assume that station B is in the transmitting range of both A and C, but A and

C cannot hear each other. Let us also assume that A is transmitting to B. If C has a frame to be transmitted to B, according to the DFC protocol, it senses the medium and finds it free because it is not able to hear A's transmissions. Therefore, it starts transmitting the frame but this transmission will results in a collision at the destination Station B.

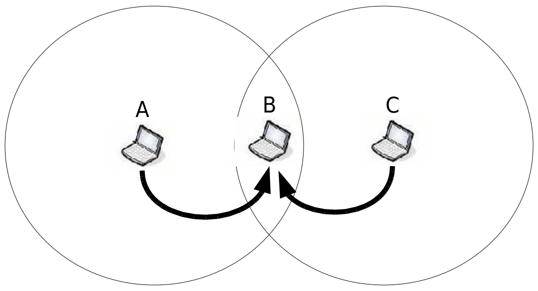


Figure 3.3: Hidden Station.

The hidden station problem can be alleviated by extending the DCF basic mechanism by a *virtual carrier sensing mechanism* that is based on two control frames: *Request To Send* (RTS) and *Clear To Send* (CTS), respectively. According to this mechanism, before transmitting a data frame, the station sends a short control frame, named RTS, to the receiving station announcing the upcoming frame transmission (see Figure 3.4). Upon receiving the RTS frame, the destination station replies by a CTS frame to indicate that it is ready to receive the data frame. Both the RTS and CTS frames contain the total duration of the transmission, i.e., the overall time interval needed to transmit the data frame and the related ACK. This information can be read by any listening station that uses this information to set up a timer called *Network Allocation Vector* (NAV). While the NAV timer is greater than zero the station must refrain from accessing the wireless medium. By using the RTS/CTS mechanism, stations may become aware of transmissions from *hidden station* and on how long the channel will be used for these transmissions.

Figure 3.5 depicts a typical scenario where the *exposed station* problem may occur. Let us assume that both Station A and Station C can hear transmissions from B, but Station A can not hear transmissions from C. Let us also assume that Station B is transmitting to Station A and Station C receives a frame to be transmitted to D. According to the DCF protocol, C senses the medium and finds it busy because of B's transmission. Therefore, it refrains from transmitting to C although this transmission would not cause a collision at A. The *exposed station* problem may thus result in a throughput reduction.

However, even with the RTS/CTS mechanism enabled, the hidden terminal problems still exist [55, 128]. Figure 3.6 depicts a typical scenario where the hidden and exposed terminal problems may occur in the presence of the RTS/CTS mechanism. Let us assume that node C is transmitting to station D and node A want to transmit a frame to station B. Node A, according to

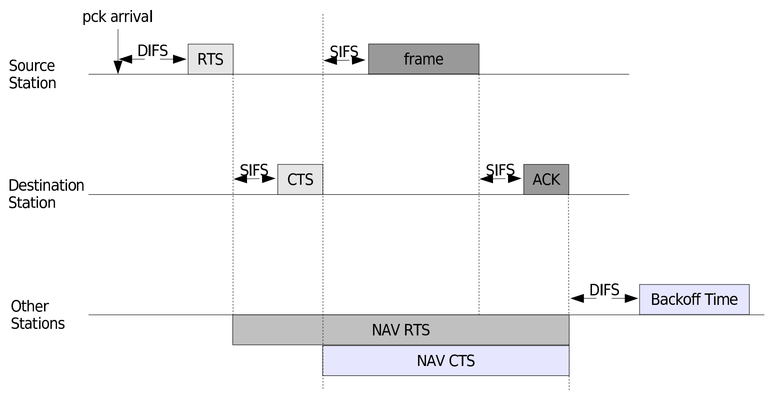


Figure 3.4: Virtual Carrier Sensing Mechanism.

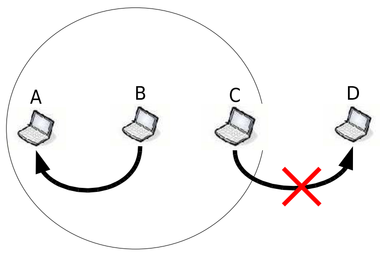


Figure 3.5: Exposed Station.

the DFC protocol, senses the medium and finds it free because it is not able to hear C's transmissions. Therefore, it starts transmitting the RTS frame but this transmission will results in a collision at the destination Station B. Even if the collision doesn't happen, node B cannot reply to node A, since it has the NAV set. Also node F cannot communicate to node E, since node E has its NAV set.

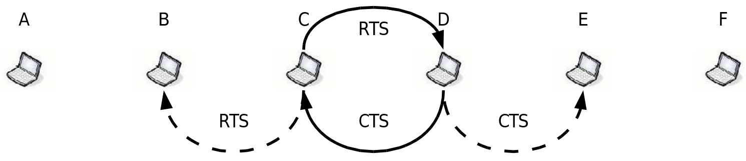


Figure 3.6: Hidden and Exposed Station in the presence of the RTS/CTS mechanism.

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 three communication ranges: the *Transmission Range*, the *Interference Range*, and the Physical Carrier Sensing Range. The *Transmission Range* is the range (with respect to the transmitting station) within which a transmitted packet can be successfully received. It is mainly determined by the transmission power and the radio propagation properties. The *Physical Carrier Sensing Range* is the range (with respect to the transmitting station) within which the other stations detect a transmission. It mainly depends on the sensitivity of the receiver (the receive threshold) and the radio propagation properties. The *Interference Range* is the range within which stations in receive mode will be "interfered with" by a transmitter, and thus suffer a loss. The interference range is usually larger than the transmission range, and smaller then the Physicall Carrier Sensing Range. It is a function of the distance between the sender and receiver, and of the path loss model. It is very difficult to predict the interference range as it strongly depends on the ratio between power of the received "correct" signal and the power of the received "interfering" signal. Both these quantities heavily depend on several factors (i.e., distance, path, etc.) and hence to estimate the interference is needed a detailed snapshot of the current transmission and relative station position. In simulation studies the following relationship has been generally assumed:

$$TX\_range \leq IF\_range \leq PCS\_range$$

For example, in the ns-2 simulation tool [3] the following values are used to model the characteristics of the physical layer:

$$TX\_range = 250m, IF\_range = PCS\_range = 550m$$

Chapter 4

Routing Protocols for MANETs

Development of routing protocols for ad hoc networks has been one of the hottest topics within this area in recent years. As a consequence, a large number of routing protocols have been designed, either by modifying Internet routing protocols, or proposing new routing approaches [101, 89, 100, 68, 58, 37, 38, 78, 75, 81, 134, 123, 94]. In the following, I present a high-level classification of MANET routing protocols, and then I describe the two routing protocol used in my thesis.

MANET routing protocols are typically subdivided into two main categories [25]: *proactive routing protocols* [101, 89, 100] and *reactive on-demand routing protocols* [100, 68]. Proactive routing are derived from the traditional *distance vector* and *link state* protocols developed for Internet. The primary characteristic of proactive approaches is that each node in the network maintains a route to every other node in the network at all times. Route creation and maintenance is accomplished through some combination of periodic and event-triggered routing updates. This approach have the advantage that routes are available at the moment they are needed. A source can simply check its routing table, when it has data packets to send to some destination, and begin packet transmission. However, the primary disavantage of these protocols is that the control overhead can be significant in large. Further, the amount of routing state maintained at each node scales as $O(n)$, where n is the number of nodes in the network.

Reactive on demand routing protocols take a very different approach than proactive protocols, since they do not maintain a route between all pairs of network nodes. Instead, reactive protocols discover the route to a destination only when there is a demand for it. Specifically, when a source node needs to send date packets to some destination, it checks its routing table to determine whether it has a route. If no route exists, it performs a *route discovery* procedure to find a path to the destination. Hence, route discovery becomes on-demand. With this approach, if two nodes never need to talk to each other, then nodes in the network do not need to utilize their resources maintaining a path between each other. The benefit of this approach is that signalling overhead is likely to be reduced compared to proactive approaches, particularly in networks with low to moderate traffic load. When the number of data sessions in the network becomes high, then the overhead generated by the reactive routing protocols may even surpass that of the proactive approaches. The drawback of reactive approaches is the introduction of a *route acquisition latency*. That is, when a route is needed by a source node, there is some finite latency while the route is discovered. In contrast, with a proactive approach, routes are typically available at the moment they are needed.

4.1 Ad hoc On-demand Distance Vector (AODV)

The Ad hoc On-Demand Distance Vector (AODV) [100] is a well know reactive routing protocol. With this protocol, each node maintains a routing table in which next hop routing information for destination nodes is stored. Each routing table entry has an associated lifetime value to expire the route if it is not utilized within the lifetime period. In addition, every node under AODV keeps two counters: a node sequence number and a broadcast ID. These counters are used for loop freedom, duplicates detection, and for ensuring selection of the most recent routing path.

When a node has data packets to send to some destination, it first checks its routing table to determine whether it already has a route to the destination. If such a route exists, it can use that route for packet transmissions. Otherwise, it must initiate a route discovery procedure to find a route. 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 packet reaches either the destination itself or an intermediate node that contains a fresh enough table entry pointing to the destination.

When a neighboring node receives a RREQ, it first creates a *reverse route* to the source node. Then it checks whether it has an unexpired route to the destination. If it does not have a valid route to the destination, it simply rebroadcasts the RREQ to it neighbors. In this manner, the RREQ floods the network in search of a route to the destination. Instead, if the node has a valid route to the destination, then it checks if its routing table entry is at least as recent as the source node's last known route to the destination (see [100] for details). This condition also guarantees loop freedom. If this condition is met, the node can create a *route replay* (RREP) message. After creating the reply, the node unicasts the message to its next hop towards the source node. Thus, the reverse route that was created by the RREQ is utilized to forward the RREP back to the source node.

When the next hop receives the RREP, it first creates a *forward route* entry for the destination node, using the node from which it received the RREP as the next hop toward the destination. Once the node has created the forward route entry, it forwards the RREP to the source node. The RREP is thus forwarded hop by hop to the source node. Once the source node receives the RREP, it can utilize the created path for the transmission of data packets.

Once a route is established, it must be maintained as long as it is needed (a route that has been recently utilized for the transmission of data packets is called an *active* route). Node mobility and link layer contention may produce link break along the path followed by the route. Breaks on links that are not being utilized for the transmission of data packets do not require any repair. Breaks in active routes, instead, must be quickly repaired. When a link break along an active route occurs, the node upstream of the break invalidates the routes to each of those destinations in its routing table passing through the broken link. It then creates a *route error* (RERR) message. This message lists all of the destinations that are now unreachable due to the loss of the link. After creating the RRER message, the node sends the RERR message to its upstream neighbors that were also utilizing the broken link. These nodes, in turn, invalidate the broken routes and send their own RRER messages to their upstream neighbors that were utilizing the link. The RERR message thus traverses the reverse path to the source node. Once the source node receives the RERR, it can perform a new RREQ to repair the route, if it is still needed. An optimization of AODV is the the *local repair* of link breaks in active routes. When a link break occurs, instead of sending a RERR message back to the source, the node upstream of the break can try to repair the link locally itself. If successful, fewer data packets are dropped because the route is repaired more quickly. If the local repair attempt fails, a RERR message is sent to the source node as previously described.

An important feature of AODV is that it can use two different mechanisms for neighbour discovery and local connectivity maintenance, i.e., *link layer feedback* information provided by the underlying MAC protocol, or HELLO messages. HELLO messages are periodic broadcast messages that are utilized by each node to announce its presence in the one-hop neighbourhood. In AODV, HELLO messages and broadcast messages can server the same function. For example, RREQ messages are broadcast IP packets; therefore, reception of a RREQ indicates the presence of a link. Hence, the term HELLO message is used to loosely refer to all broadcast control messages. In AODV, reception of a HELLO message indicates bidirectional connectivity to the sender. Once a link is established, failure to receive several HELLO messages from a neighbour indicates a loss of connectivity. When HELLO messages are used, each node broadcasts HELLO messages at least once every HELLO_INTERVAL seconds. Failure to receive a HELLO message for $ALLOWED\_HELLO\_LOSS * HELLO\_INTERVAL$ seconds indicates a loss of connectivity to that neighbour.

In contrast to HELLO messages, *link layer feedback* is able to quickly identify link failures during transmission of a data packet to another node. This feedback must be provided by the underlying MAC protocol, i.e. the IEEE802.11 MAC protocol. In IEEE 802.11, a unicast packet is first queued for transmission at the MAC layer. If the packet cannot be transmitted after multiple

MAC layer retries, an indication is given to the higher layers, that a failure has occurred. This results in immediate notification of a broken link as soon as a packet fails to be transmitted.

4.2 Optimized Link State Routing (OLSR)

The OLSR protocol [37] is an optimization of the classical link state algorithm tailored to mobile ad hoc networks. More precisely, being a proactive routing protocol, OLSR periodically floods the network with route information, so that each node can locally build a routing table containing the complete information of routes to all the nodes in the ad hoc network running the OLSR protocol. The OLSR routing algorithm employs an efficient dissemination of the network topology information by selecting special nodes, the *multipoint relays* (MPRs), to forward broadcast messages during the flooding process. More precisely, each node independently selects its multipoint relays among its one-hop neighbours such as to ensure that all its two-hop neighbours receive the broadcast messages retransmitted by these selected relays. The link state reports, which are generated periodically by MPRs, are called TOPOLOGY CONTROL (TC) messages. These TC messages are flooded to all the nodes in the network, but only the MPRs are allowed to forward the control messages received from other nodes, in order to reduce the number of retransmissions needed to cover the entire network.

OLSR employs a neighbour discovery procedure based on HELLO messages. The HELLO packets contain the list of neighbours known to the node and their link statuses. Thus, HELLO messages allow each node to discover its one-hop neighbours, as well as its two-hop neighbours, which are needed during the MPR selection procedure. The neighbourhood information and the topology information are updated periodically, and they enable each node to locally compute the least-cost routes to any possible destination in the ad hoc network, by using the Dijkstra's shortest path algorithm. This routing table is recomputed whenever there is a change in either the neighbourhood information or the topology information.

In order to enable the injection of external routing information into the ad hoc network, the OLSR protocol defines the HOST AND NETWORK ASSOCIATION (HNA) message. The HNA message binds a set of network prefixes to the IP address of the node attached to the external networks, i.e., the gateway node. Consequently, each ad hoc node is informed about the network address and netmask of the network that is reachable through each gateway. In other words, the OLSR protocol exploits the mechanism of *default routes* to advertise Internet connectivity. For instance, a gateway that advertises the conventional default route 0.0.0.0/0, will receive each packet destined to IP addresses without a known route on the local ad hoc network.

The periodic exchange of OLSR control packets is regulated by a set of parameters that establish the timing for the OLSR operations. These parameters define the generation period of each control packet and the validity time related to the information provided with the control packet. The default constant values for these parameters are defined in the OLSR RFC [37]. For example, HELLO messages are generated by each node with period equal to *HELLO_Interval*, and the information provided in HELLO messages is considered valid for a *NEIGHB_HOLD_TIME*. The TC messages, instead, are generated by the each MPRs every *TC_Interval*, and their validity time is *TOP_HOLD_TIME*. Finally, each gateway being connected to external networks, generates HNA messages every *HNA_Interval*, and their information is valid for *HNA_HOLD_TIME*.

Chapter 5

Transmission Control Protocol

Transmission Control Protocol (TCP) is the *de facto* standard for reliable connection-oriented transport protocols, and is normally used over IP (Internet Protocol) to provide end-to-end reliable communications to Internet applications. TCP provides a reliable, connection-oriented, and full duplex type of service. In addition, TCP implements both flow control and congestion control mechanisms. The former prevents the TCP receiver's buffer from being overflowed. The second is an end-to-end congestion control mechanism, that prevents process to inject into the network an excessive traffic load.

This chapter introduces TCP and discusses the main concepts and mechanisms associated with it. In addition it describes some TCP variants, like TCP Reno, TCP NewReno, TCP Sack, and TCP Vegas. Moreover, it introduces some TCP extentions, like Delayed Acknowledgments (DA), Explicit Congestion Notification (ECN) and Limited Transmit.

5.1 TCP Segment Structure

The TCP segment consists of an *header field* and a *data field*. The *data field* contains a chunk of application data. The MSS (Maximum Segment Size) limits the maximum size of a segment's data field. When TCP sends a large file it typically breaks the file into chunks of size MSS. However, the size of the data field can be smaller that MSS. For example with application like Telnet, the data field in the TCP segment is often only one byte. 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.

Figure 5.1 shows the structure of the TCP segment. The header includes the following fields:

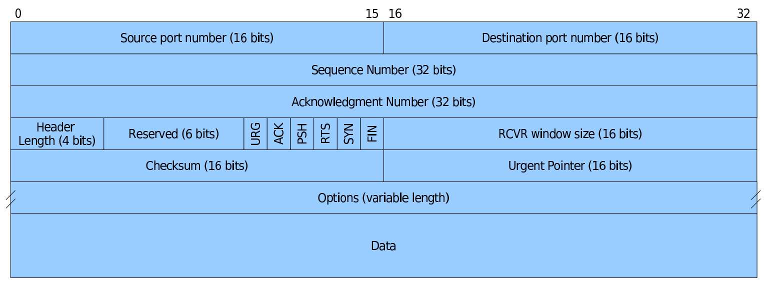

0
15
16
32
Source port number (16 bits)
Destination port number (16 bits)
Sequence Number (32 bits)
Acknowledgment Number (32 bits)
Header Length (4 bits)
Reserved (6 bits)
URG
ACK
PSH
RTS
SYN
FIN
RCVR window size (16 bits)
Checksum (16 bits)
Urgent Pointer (16 bits)
Options (variable length)
Data

Figure 5.1: TCP header.

- *Source and Destination port number fields*: these fields are used for multiplexing/demultiplexing data from/to upper layer applications. These two values combined with the source and destination fields in the IP header, uniquely identify each connection.

- *Sequence Number field*: TCP views data as an unstructured, but ordered stream of bytes. The *sequence number* field is the byte-stream number of the first byte in the segment. This 32-bit fields is used by TCP to implement a reliable data transfer service, as discussed in Section 5.3.

- *Acknowledgment Number field*: TCP is full-duplex. This means that an host A may be receiving data from host B while it sends data to host B (as part of the same TCP connection). Each of the segments sent by host B have a sequence number for the data flowing from B to A. The *acknowledgment number* that host A puts in its segment is the sequence number of the next byte host A is expecting from host B. This 32-bit fields is used by TCP to implement a reliable data transfer service, as discussed in Section 5.3.

- *Header Length field*: This field specifies the length of the TCP header in 32-bits words. The TCP header can be of variable length due to the TCP *option field.* By having 4 bits, this field limits the header size to 60 bytes. The length of a TCP header with no options set is 20 bytes.

- *Reserved field*: This fields was reserved for future use. For example, some bits of this field are used by the Explicit Congestion Notification mechanism [104, 108].

- *Flag field*: This fields contains 6 bits. The *URG* bit specifies that the Urgent Pointer filed is valid (there is data in the segment marked as "urgent"). The *ACK* bit indicates that the value carried out in the acknowledgment number field is valid. The *PSH* bit indicates to the receiver that it should pass the data to the upper layer immediately. The *RST* bit resets the connection. The *SYN* and *FIN* bits are used for connection setup and teardown (see Section 5.2).

- *rcvr window size field*: This field contains the size of the receiver window, which defines the number of bytes the TCP receiver is willing to accept from the sender (see Section 5.4).
- *Checksum field*: This field is used for error detection. 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 field*: This field is valid only if the URG flag is set. It specifies a part of the data filed that must be sent quickly to the receiver.
- *Options Field*: This field is used by a sender and receiver pair to negotiate TCP options, such as the maximum segment size (MSS), timestamps, Window Scale Option, etc.

5.2 TCP Connection Management

TCP is a connection oriented transport protocol. This means that before one application process can being to send data to another, the two processes must first perform an handshake to open a TCP connection with each other. During the TCP connection establishment, both sides of the connection will initialize many TCP "state variables" associated with the TCP connection. The connection state resides entirely in the two end systems. The intermediate network elements do no maintain TCP connection state.

A TCP connection provides for *full duplex* data transfer. If there is a TCP connection between process A and B, the application-level data can flow from A to B and from B to A at the *same time.* A TCP connection is also always *point-to-point*, that is, between a single sender and a single receiver. Multicasting is not possible with TCP.

Now, I will briefly describe the procedures used by TCP to setup and teardown a connection.

5.2.1 Connection Setup and teardown

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*" since 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 packets used exclusively for connection setup reasons. Figure 5.2 illustrates the three-way handshake's exchanges [41].

The client host first sends a special segment to the server host requesting a connection setup. This segment is generally named SYN segment and does not contain any data. It only contains the header with the SYN flag set to one and the desired initial sequence number X. Provided that the server host is able to accept the connection, it allocates the TCP buffers and variables associated

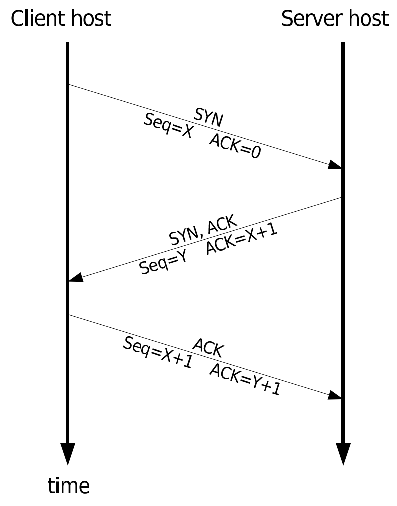


Figure 5.2: TCP three-way handshake.

with the connection, and sends back an acknowledgment to the client host. This acknowledgment also does not contain any data, has the SYN flag set to one, and has the sequence number field set with the server host's desired value Y. Additionally, this packet has the acknowledgment field in the TCP header set to sequence number of the received SYN 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 associated with 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 plus one, i.e. X+1, 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. After the connection termination, both end nodes have their resources freed. As an example, suppose that client decides to close the connection, as show in Figure 5.3. The client TCP sends a special TCP segment to the server. This segment is generally named FIN segment since it has the FIN flag set to 1. When the server receives this segment, it sends the client an acknowledgment in return. The server, when the application decides it want to close the TCP connection, sends its own shutdown segment, which has the FIN bit set to 1. Finally, the client acknowledges the server's FIN segment. At this point, all the resources in the two hosts are de-allocated.

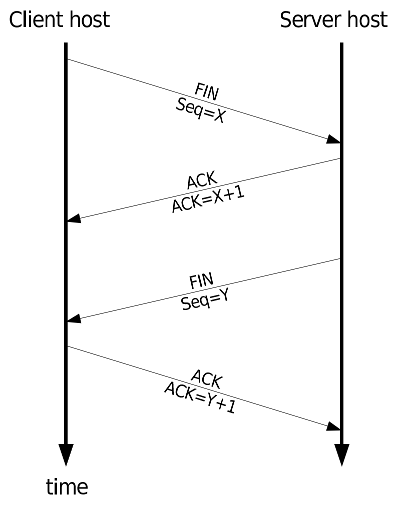


Figure 5.3: TCP connections teardown.

5.3 Reliable Data Transfer

TCP implements a reliable data-transfer service. This service ensures that the byte stream received by the application is exactly the same byte stream that was sent by the end system on the other side of the connection (no gaps, no corrupted packets, no out of order packets). TCP provides a reliable data transfer service by using positive acknowledgments and timers (retransmission timer). Each time TCP passes a data segment to IP, it starts a timer for that segment. If this timer expires, TCP reacts retransmitting the segment in timeout.

On ACK reception, the sender must determine whether the ACK is a new ACK for a segment for which it has not yet received an acknowledgment, or a *duplicate ACK* that re-acknowledges a segment for which it has already received an acknowledgment. In the case of a new ACK, the sender discovers that all data up to the byte being acknowledged has been received correctly at the receiver. Consequently, it updates its state variable that tracks the sequence number of the last byte that is know to have been received correctly and in order by the receiver. If the TCP sender receives three duplicate ACKs for the same data, it assumes that the segment following the segment that has been ACKed three times has been lost. In this case, TCP performs a Fast Retransmission [10], retransmitting the missing segment before that segment's timer expires. The *Fast Retransmit* saves the time the sender would waste by waiting for the retransmission timer expiration.

On the receiver side, TCP generates ACKs on reception of an in-order segment with expected sequence number, and if there are no gaps in the received data. In this case TCP sends a cumulative ACK, acknowledging all received in-order segments. The TCP receiver, instead, generates a *duplicate ACK* when it detects a gap in the data stream (there are missing segments). This happen if it receives a segment with a sequence number that is larger than the next, expected, in-order sequence number. In this case, TCP re-acknowledges (generates a duplicate ACK) the last acknowledged segment.

If the Delayed ACK mechanism [28] is used by the TCP receiver, TCP generates an ACK only if it receives an in-order segment with expected sequence number, there is one other in-order segment waiting for ACK transmission, and there are no gaps in the received data. If, after a predefined interval on the reception of the first ACK, the next in-order segments does not arrive, TCP sends an ACK.

5.4 Flow Control

The sending rate of a TCP connection is regulated by two distinct mechanisms, the flow control and the congestion control. Both these mechanisms, use a *sliding window* mechanism to manage data transmission. Thus, the TCP sender contains a variable, denoted window, determining the amount of segments it can send into the network before receiving an ACK. This variable changes dynamically over time to properly limit the connection's sending rate. This section describes the flow control mechanism. Section 5.6 instead, describe the congestion control mechanism.

On each side of a TCP connection there is a receiver buffer. This is the buffer where TCP places data for application. Specifically, when TCP connection receives bytes that are correct and in sequence, it places these in the receiver buffer. However, the application process will read data from this buffer not necessarily at the instant the data arrives. Consequently, if the application is relatively slow, the TCP sender can overflow the connection's receiver buffer. Flow control is implemented to avoid that a TCP sender overflows the receiver's buffer. With this mechanism, the receiver uses the *rcvr window size* field of the TCP header of every ACK transmitted, to advertise 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. The flow control mechanism throttles the rate at which TCP is sending to the rate at which the receiving application is reading data, avoiding thus buffer overflow at the receiver.

To implement flow-control, the receiver side of a TCP connection maintains two variables:

- *LastByteRead*: the number of the last byte in the data stream read from the receiver buffer by the application.
- *LastByteRcvd*: the number of the last byte in the data stream that has arrived from the network and has been placed in the receiver buffer.

Not to overflow the receiver buffer, we must have:

$$LastByteRcvd - LastByteRead <= RcvBuffer$$

where $RcvBuffer$ is the size of the receiver buffer. Thus, the TCP receiver must set the *rwin window size* field of each segment sent to the following value:

$$RcvWindow = RcvBuffer - [LastByteRcvd - LastByteRead]$$

where $RcvWindow$ is the receive window (i.e., the available space of the receiver buffer). On the sender side, instead, TCP maintains the following two variables:

- *LastByteSent*: the number of the last byte in the data stream sent to the destination.
- *LastByteAcked*: the number of the last byte in the data stream received by the destination.

The difference between these two variables is the amount of unacknowledged data. By keeping this value less than the value of the received $RcvWindow$, there is no overflow at the receiver buffer.

5.5 Round Trip Time and Timeout

AS described in Section 5.3, TCP uses positive acknowledgments and timers to ensure a reliable service. Specifically, when TCP sends a segment, it starts a timer. If the timer expires before the reception of an acknowledgment for the data in the segment sent, TCP retransmits the segment. The duration of the timer is called retransmission timeout (RTO). RFC 2988 [96] is the most up-to-date specification for computing the RTO value. This RFC is a refinement of the algorithm proposed by Jacobson in [63]. The algorithm specified in RFC 2988 is describe below.

A TCP sender maintains two state variables for computing RTO, the average of the measured round-trip time ($ERTT_{rtt}$) of the TCP connection, and the round-trip time variation (DEV_{rtt}). Additionally, a clock granularity of G seconds is assumed in the computation. The rules governing the computation of $ERTT_{rtt}$, DEV_{rtt} and RTO are as follows.

1. Until a 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:

$ERTT_{rtt} = R$

$DEV_{rtt} = R/2$

$RTO = ERTT_{rtt}(n) + max(G, K \times DEV_{rtt})$, where $K = 4$

3. When a subsequent RTT measurement $R(n)$ is made, the sender must update the variables as follows:

$$DEV_{rtt}(n) = (1 - \beta) \times DEV_{rtt}(n-1) + \beta \times |ERTT_{rtt}(n) - R(n)|$$

$$ERTT_{rtt}(n) = (1 - \alpha) \times ERTT_{rtt}(n-1) + \alpha \times R(n)$$

$$RTO = ERTT_{rtt}(n) + max(G, K \times DEV_{rtt}(n)), \text{ where } K = 4$$

   where: i) $ERTT_{rtt}(n)$ and $DEV_{rtt}(n)$ are the average value and the standard deviation of the RTT estimated at the n^{th} step. α and β are normally set to 1/8 and 1/4, respectively

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 [64], RTT samples must not be taken for packets that were retransmitted, as specified in the Karn's algorithm [70]. 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:

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

2. TCP takes a timer expiration as a sign of network congestion. Under this hypothesis, using the same RTO for the new segment to send would not make sense, as it could easily lead to new expiration of the timer. Consequently, TCP adopts a binary exponential backoff mechanism to retransmits the segment. Specifically, after consecutive timeout expiration, the sender doubles the RTO up to a given limit.

3. Restart the retransmission timer, such that it expires after the current RTO.

5.6 Congestion 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 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 can 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 in such a way that the amount of unacknowledged data within the TCP connection may not exceed the minimum of *cwnd* and *rwin*, that is:

$$LastByteSent - LastByteAcked <= min(cwnd, rwin)$$

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.

To implement congestion control, each side of a TCP connection keep track of the *cwnd* variable and also uses another variable named *threshold.* This variable affects how *cwnd* grows. In the following I will assume that the TCP receive buffer is so large that the receiver window constraint can be ignored. In this case, the size of the transmission window is solely limited by *cwnd.* Further, I will assume that the sender has a very large amount of data to send to the receiver.

At the start of TCP connection, the congestion window is equal to one MSS. After this, TCP increases its *cwnd* using the *Slow Start* algorithm. The idea behind Slow Start is to make the connection rate to start slowly and then rapidly rises toward the communication channel capacity. To do so, it adopt an *exponential* enlargement of *cwnd.* Specifically, in Slow Start, for each ACK received the sender increases its *cwnd* by one MSS and so transmits two new data segments. This phase of the algorithm is called Slow Start because it begins with a small congestion window equal to one MSS.

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. Congestion Avoidance* imposes a *linear* increase to *cwnd.* Specifically, if W is the current value of the congestion window, after W acknowledgments have arrived, TCP replaces W with $W + 1$.

The *Congestion Avoidance* phase continues as long as the acknowledgments arrive before the corresponding tomeouts. But the rate at which the TCP sender can send, cannot increase forever. Eventually, TCP will saturate the links along the path, at which point loss (and a resulting timeout at the sender) will occur. When a timeout occurs, the value of *ssthresh* is set to half the value of the current congestion window, and the congestion window is reset to one MSS. The sender then again grows the congestion window using Slow Start until the congestion window hits the threshold, and then using *Congestion Avoidance.*

In summary, using packets instead of bytes to denote the congestion window size, the growth experienced by TCP window during Slow Start and Congestion Avoidance is generally performed as follows:

$$cwnd = \begin{cases} cwnd + 1 & \text{if } cwnd < ssthresh \quad \text{Slow Start} \\ cwnd + \dfrac{1}{cwnd} & \text{if } cwnd \geq ssthresh \quad \text{Congestion Avoidance} \end{cases}$$

The TCP congestion control algorithm just described is often referred to as *Tahoe.* One problem of this TCP version is that, when a segment is lost, the sender side of the application may have to wait a long period of time for the timeout. For this reason, a variant of Tahoe, called *Reno*, was proposed by Jacobson [63]. This variant introduces the *Fast Recovery* mechanism. 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, the only difference between TCP Reno and TCP Tahoe is the use of the *Fast Recovery* mechanism.

This variant of congestion control is often referred to as an *Additive Increase Multiplicative Decrease* (AIMD) algorithm [35]. The name AIMD comes from the behaviour 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 above. This continues until either a dropped segment 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 follow:

$$cwnd = \begin{cases} \uparrow cwnd + \dfrac{1}{cwnd}, & \text{Additive Increase} \\ \downarrow \dfrac{cwnd}{2}, & \text{Multiplicative Decrease} \end{cases}$$

5.7 TCP Variants

This section presents the main TCP variants 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. Slight refinements to these implementation have been described in the RFCs 2581, 2582, and 3782 [10, 50, 51], but the general concepts remain unchanged.

5.7.1 TCP Tahoe

Tahoe represents the basic TCP version that was specified by Jacobson in [63]. It was the first TCP designed to solve the congestion collapse 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, 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 5.6. 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.

5.7.2 TCP Reno

TCP Reno [110] conserved the three essential mechanisms of the basic TCP Tahoe, namely *Slow Start*, *Congestion Avoidance* and *Fast Retransmit.* As explained in section 5.6, the novelty introduced into TCP Reno is the *Fast Recovery* mechanism. This mechanism prevents the communication path ("pipe") from going empty after Fast Retransmit, thereby avoiding the need to Slow Start to re-fill it after a single packet loss.

Fast Recovery is generally invoked when a TCP sender receives a predefined threshold of duplicate ACKs, just after the Fast Retransmit mechanism. This threshold, usually known as *tcprexmtthresh*, is generally set to three. Once the threshold of dup ACKs is received, the 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 outgoing data packets.

Fast Recovery assumes that each dup ACK received represents a single packet having left the pipe. Thus, during Fast Recovery the TCP sender is able to make intelligent estimates of the amount of outstanding data. Specifically, during Fast Recovery the usable TCP window is defined as $min(rwin, cwnd + ndup)$, where *rwin* refers to the receiver advertised window and *ndup* tracks the number of duplicate ACKs received. By using the *ndup* variable, the sender may estimate the amount of packets in flight. After entering Fast Recovery and retransmitting a single packet, the sender effectively waits until half a window of dup ACKs have been received, and then sends a new packet for each additional dup ACK that is received. Upon receipt of an ACK for new data (called a "recovery ACK"), the sender exits Fast Recovery by setting *ndup* to 0.

TCP Reno is optimized for the case when 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.

5.7.3 TCP NewReno

NewReno [50, 51] 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, the NewReno do not exit Fast Recovery when it receives partial ACKs. Instead, TCP NewReno treats partial ACKs received during Fast Recovery as an indication that the packet immediately following the acknowledged packet in the sequence space has been lost, and should be retransmitted. Thus, when multiple packets are lost from a single window of data, TCP NewReno can recover without a retransmission timeout, retransmitting one lost packet per round-trip time until all of the lost packets from that window have been retransmitted. TCP NewReno remains in Fast Recovery until all of the data outstanding when Fast Recovery was initiated has been acknowledged. In this way, TCP NewReno avoids multiple reductions in the cwnd or unnecessary retransmit timeout with Slow Start invocation, thereby improving the end-to-end performance.

5.7.4 TCP Sack

TCP Sack (Selective Acknowledgment) [82] preserves the basic principles of TCP Reno. In fact, it uses the same algorithms of Reno for increasing and decreasing its congestion window. The novelty in TCP Sack lies in its behaviour when multiple packets are dropped from one window of data, similarly to TCP NewReno. In Sack, the receiver uses the option fields of TCP header (Sack option) for notifying the status of data received and queued by the receiver.

The SACK option field contains a number of SACK blocks, where each SACK block reports the received and queued bytes of data that are contiguous and isolated (there are gaps in the data stream). The first block in a SACK option is required to report the most recently received segment, and the additional SACK 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 if 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. 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 terminates 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.

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.

5.7.5 TCP Vegas

Differently from the above examined TCP versions, TCP Vegas [29] is not an ACK-clocked congestion control. Specifically, TCP Vegas does not 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 as follow:

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

With,

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

Where *rtt* means the measured RTT, *base_rtt* is the smallest value of observed RTTs so far, and α and β are some constant values. This equation indicates that if RTTs of the segments are stable, TCP Vegas keeps the window size unchanged. This 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.

TCP Vegas has another feature in its congestion control algorithm. During Slow Start, TCP Vegas increments its *cwnd* slower than TCP Tahoe. Specifically, to be able to detect and avoid congestion during Slow Start, Vegas allows exponential growth only every other RTT. In between, the congestion window stays fixed so a valid comparison of the expected and actual rates can be made.

5.8 TCP Extensions

5.8.1 Delayed Acknowledgments (DA)

As stated in RFC 813 [36], 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 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 infers 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 [36] introduces a new mechanism that optimizes 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 [36].

The delay ACK mechanism optimizes transmission efficiency by reducing the number of acknowledgments generated by a TCP receiver. 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 [28] and refined in RFC 2581 [10]. The former specifies that a TCP receiver should acknowledge every other data packets 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 senders 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 receivers 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 receivers 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.

5.8.2 Limited Transmit

RFC 3042 [9] specifies the Limited Transmit as an enhancement for TCP loss recovery when a connections 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 receivers 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.

5.8.3 Explicit Congestion Notification (ECN)

The ECN scheme specified in RFC 3168 [104] 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 overow 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 signalling among sender, routers and receiver. The active queue management (AQM) [27, 104] 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.

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.

Chapter 6

Miscellaneous Protocols

6.1 Background on the Mobile IP Protocol

The Mobile IP protocol [99] was developed to enable the roaming in Internet of mobile nodes between different networks. Normal IP routing adopts a hierarchical addressing scheme, in which the IP address is location dependent and composed of two parts: the *network identifier* and the *host identifier.* If a mobile node moves from its home network, i.e., the network to which its IP address belong, to another network without changing address, the routers will not be able to correctly route packets addressed to it. To manage the IP mobility, the Mobile IP solution introduces two entities: the *home agent* (HA) and the *foreign agent* (FA). The HA is a host or router in the node's home network that is responsible to keep tracking the mobile node location and to tunnel packets to the node while it is away from its home network. The FA is a host or router in the mobile node's visited (foreign) network, which registers the entrance of mobile nodes, detunnels and forward traffic to the visiting node. To make mobility transparent to applications and higher layer protocols, the mobile node should continue to use its home address to receive data, even when it leaves its home network. However, when the mobile node is attached to a network other than its home network, specific procedures are executed to assign to the new node an IP address belonging to the visited network, called *care-of-address.* If the care of address is one of the addresses announced by the FA, it is known as *foreign agent care-of-address.* If the mobile node is able to acquire an IP address valid on the foreign network that it is visiting (e.g., using a DHCP server), such a care-of-address is called *co-located care-of-address.* To be able to receive packets while visiting a foreign network, the mobile nodes should register with the HA its care-of-address, which provides the information about the current mobile node's point of attachment to the Internet. To accomplish this task, the mobile node send a REGISTRATION REQUEST message directly (if it is using the co-located care-of address) or via the FA (if it using the FA care-of-address) to its HA, which in turn responds with a REGISTRATION REPLY message. Data packets sent to the mobile node's home address are intercepted by its HA, which tunnels those packets to the mobile node's care-of-address.

One of the key mechanisms in Mobile IP is the procedure for updating the location information of mobile nodes. Mobile IP uses a proactive approach since both HAs and FAs periodically broadcast AGENT ADVERTISEMENT messages to announce their presence. A mobile node uses these messages to determine whether it is attached to its home network or it is visiting a foreign network. In addition, the AGENT ADVERTISEMENT messages enable the visiting mobile nodes to discover the care-of-address of the advertised FA. When a mobile node receives AGENT ADVERTISEMENT messages from more than one FA, there are several algorithms to manage the FA selection and the handoff to a new FA, such as the Lazy Cell Switching (LCS) and the Eager Cell Switching (ECS) [102]. In addition, mobile nodes can proactively search for FAs by sending AGENT SOLICITATION messages. Upon receiving this type of message, the FA should reply with a unicast AGENT ADVERTISEMENT message to the mobile node that issued the solicitation.

6.2 Network Address Translation (NAT)

The process of Network Address Translation (NAT) is used to enable multiple hosts on a private network to access Internet using a public IP address associated to the gateway. More precisely, a router with a NAT module implemented on it, behaves as an address translator, dynamically mapping the set of private addresses used in the local domain to a set of globally valid network addresses. Figure 6.1 shows a typical scenario where the NAT is used. In this scenario there is a private network (192.168.0.1/24) that access Internet through a gateway implementing a NAT daemon. The gateway is connected to the private network using a private address (192.168.0.1) and is also connected to the Internet with a single "public" address or multiple "public" addresses. As traffic passes from the local network to the Internet, the NAT daemon translates the source address in each packet from the private addresses to one public address belonging to the set of globally valid network addresses associated with the gateway. When a reply returns to the router, the NAT daemon replaces the destination address with that of the internal host that must receive the reply.

A variant of basic NAT is the Network Address Port Translation, or NAPT [109]. It permits to translate multiple private IP addresses to a single globally routable IP addresses by using different transport layer ports. NAPT works as follow. When an host on the private network sends a packet to an host in Internet, the NAPT daemon replaces the private IP address in the packet header's source field (sender's address) with the NAPT device's public IP address. It then assigns the connection a port number from a pool of available ports, inserts this port number in the packet header source port field, and places the packet on the outside network. The NAPT device then makes an entry in its *translation table* containing the private IP address, private source port, and assigned port. Subsequent packets belonging to the same connection and sent

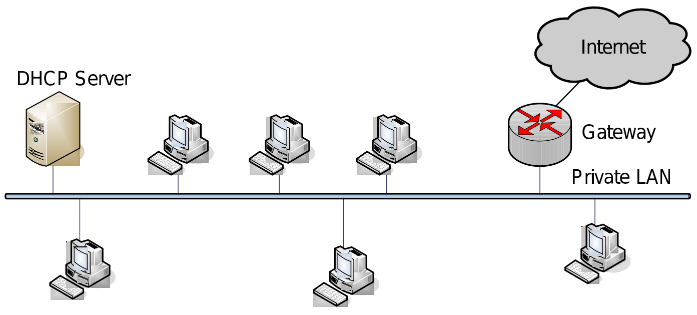


Figure 6.1: NAT example.

to Internet are translated using the same outside port number. The host receiving a data packet will move the source IP address and source port as the corresponding destination fields in any response it sends back. When NAPT receives packets from the Internet, it operates as follow. If NAPT doesn't find the destination port number of the incoming packet in the translation table, it drops the packet, because the NAPT device doesn't know where to send it. Otherwise, it replaces the destination IP address and the destination port in the incoming packet header, with that obtained from the translation table. The packet is then placed on the inside network. The NAPT device periodically deletes translations from its table when they no longer appear to be in use.

6.3 Address resolution Protocol (ARP)

IP-based applications address a destination host using its IP address. On the other hand, on a physical network individual hosts are known only by their physical address, i.e., MAC address. The ARP protocol [103] is then used to translate, inside a physical network, an IP address into the related MAC address.

The ARP protocol works as follow. As a packet is sent down through the network layers, routing determines the protocol address of the next hop for the packet. At this point, to forward the packet to the destination, the Address Resolution module must be consulted to convert the IP address of the next hop to a MAC address. The Address Resolution module at first tries to convert the IP address using a table. If it finds this, it gives the corresponding MAC address back to the caller, which then can transmits the packet. If it does not, the ARP module broadcasts the *ARP_Request* message to all hosts attached to the same physical network. This packet contains the IP address the sender is interested in communicating with. The target host, recognizing that the IP address in the *ARP_Request* packet matches its own, returns its MAC address to the requester using an unicast *ARP_Reply* message. The ARP protocol uses the *ARP table* to keep a cache of ARP responses received by the host, to avoid continuous ARP requests. An rare usage of ARP is *gratuitous ARP*. This mechanism is used by a host to announce the mapping between an IP address (generally its own IP address), and its MAC address on a physical network. A gratuitous ARP request is an *ARP_Reply* for which no request has been made.

In addition to these basic functionalities, the ARP protocol has been enhanced with more advanced features. For instance in [103] it has been proposed the *Proxy-ARP* mechanism, which allows constructing local subnets. Basically, the Proxy ARP technique allows one host to answer the ARP requests intended for another host. This mechanism is particularly useful when a router connects two different physical networks, say *NetA* and *NetB*, belonging to the same IP subnet (see Figure 6.2). By enabling the Proxy ARP on the router's interface attached to *NetB*, any host B in *NetB* sending an ARP request for a host A in *NetA*, will receive as response the router's MAC address (host C). In this way, when host B sends IP packets for host A, they arrive to the router C, which will forward such packets to host A.

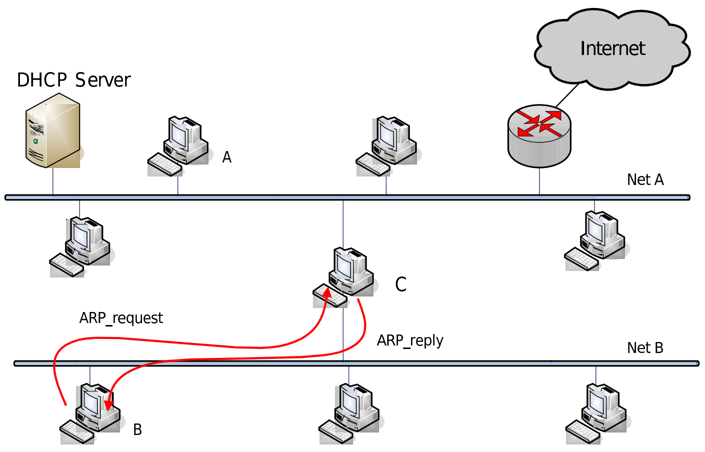


Figure 6.2: Subnetting with Proxy ARP.

Part II

Interconnecting MANETs and Internet

Chapter 7

Introduction

The recent advances in mobile and ubiquitous computing, and the development of inexpensive, portable devices are extending the application fields of ad hoc networking. Mobile users are looking for multi-purpose networking platforms in which cost is an issue and Internet access is a must. As a consequence, nowadays, multi-hop ad hoc networks do not appear only as isolate self-configured networks, but also emerge as a flexible and low-cost extension of wired infrastructure networks, coexisting with them. A new class of networks is emerging from this view, in which a mix of fixed and mobile nodes interconnected via heterogeneous (wireless and wired) links forms a multihop heterogeneous ad hoc network integrated into classical wired/wireless infrastructure-based networks.

Three different categories of solutions have been proposed for enabling interconnection between ad hoc networks and the Internet. One approach utilizes the Network Address Translation (NAT) mechanism, implemented on each gateway that interconnect the ad hoc network with the wired infrastructure network. With this approach, the mobile nodes do not need a globally routable IP address because the NAT gateway translates the source private IP address of outgoing traffic with a public IP address, which is routable on the fixed Internet. An alternative approach relies on the design of techniques capable of automatically configuring a unique, topology-dependent and globally routable IP address for each mobile node visiting an ad hoc network (IPv6 based solutions). Finally, a third category of solutions assumes that a Mobile IP Foreign Agent is implemented in the ad hoc nodes that act as Internet gateways. In this case, the mobile node needs a *permanent* and unique globally routable IP address (i.e., its home address), which is used during the registration procedures with the foreign agents of the visiting ad hoc network.

However, all the solution that have been proposed in literature have a number of disadvantage. For example, the Mobile IP based solutions have several drawbacks. The first is that in order to allow Mobile IP and ad hoc networking to cooperate it is needed to introduce further complexities and sub-optimal operations in the implementations of both Mobile IP and ad hoc routing protocols. In addition, Mobile IP was designed to handle mobility of devices in case of relatively infrequent mobility. Thus the overheads introduced to manage roaming and handoffs between foreign agents are a relevant issue in MANET environments. Finally, when the technique of default routes is used to route the traffic from the mobile node to the closest gateway, the use of Mobile IP can easily lead to triangle routing. In fact, when the mobile node moves it can get closer to a gateway different from the one to which it is currently registered. As a consequence, the forward traffic leaves the ad hoc network through one gateway while the return traffic enters the ad hoc network through the new MIP-FA gateway to which the mobile node is registered. To solve this problem the authors of [48] have proposed to use Mobile IP reverse tunnelling [69], or explicit tunnelling to one of the gateway instead of using default routes. However, the tunnelling mechanism introduces a non negligible overhead in the communication between nodes.

Also the NAT based solutions have some drawback. For example they encounter problem in multi-homed networks, i.e., when multiple gateways are present in the ad hoc network. Indeed, to avoid transport-layer session breaks it is necessary to ensure that each packet from the same session is routed over a specific gateway since a NAT router translates both outgoing and incoming packets. To solve this problem, some NAT based solutions adopts the IP-in-IP encapsulation mechanism to tunnel the packets towards the desired gateway. Employing explicit tunnelling ensures that each packet of the same transport-layer session is consistently routed through the same gateway, even if the source node moves. However, it introduces a non negligible overhead in every packet sent. In addition, NAT is not very suitable for incoming connections and this fact causes significant difficulties for peer-to-peer applications.

In this Part I will describe the solution that I proposed during my Ph.D. to provide Internet connectivity for ad hoc networks. This soltion is a simple yet practical approach to logically extend a wired LAN by employing proactive ad hoc networking technologies in a way that is transparent for the wired nodes. One of the main innovations of my solution is to rely only on basic ARP capabilities [103] and standard IP routing rules. By positioning my architecture at the data link layer, I may avoid undesired and complex interactions with the IP protocol and provide global Internet connectivity in a very straightforward manner. In addition, my solution includes a distributed protocol for the address autoconfiguration of ad hoc nodes, which relies on DHCP servers located in the wired part of the network and does not require that new ad hoc nodes have direct access to the DHCP servers. Using my scheme, mobile nodes can dynamically obtain a unique IP address that is topologically correct within the extended LAN. During my Ph.D., I have also prototyped the main components of my architecture in a general and realistic test-bed using the OLSR protocol [37] as the ad hoc routing protocol. In this test-bed, I have conducted a large variety of experiments, comparing the throughput performance of Internet access provided by my proposed scheme and an alternative well-known NAT-based solution [48]. The shown experimental results demonstrate that: *i*) my scheme ensures higher per-connection throughputs than the NAT-based solution, *ii*) node mobility does not cause permanent transport-layer session breaks, *iii*) node mobility induces drastic throughput degradations when using the NAT-based solution, while my proposed technique performs more efficient gateway handoffs, and *iv*) the network performances can be significantly improved by properly setting the OLSR protocol parameters such as to increase route stability.

This Part is organized as follow. Chapter 8 discusses the variety of architectural issues and design options that need to be considered to interconnect ad hoc networks to fixed IP networks. Chapter 9 introduces existing approaches to tackle the internetworking of MANETs with the fixed Internet and reviews the most well known solutions. Chapter 10 describes the design principles and the protocol details of my proposed solution. Chapter 11 shows experimental results on the network performance in various test-bed configurations. Finally, chapter 12 draws concluding remarks and discusses future work.

Chapter 8

Basic Design Challenges

To design my solution, I envisage a heterogeneous network environment in which wired and multi-hop wireless technologies transparently coexist and interoperate (see Figure 8.1). In this network, separated group of nodes without a direct access to the networking infrastructure form *ad hoc "islands"*, establishing multi-hop wireless links. Special nodes, hereafter indicated as *gateways*, having both wired and wireless interfaces, are used to build a wired backbone interconnecting separated ad hoc components. In addition, the gateways use their wired interfaces also to communicate with static hosts belonging to a wired LAN. The network resulting from the integration of the ad hoc network with the wired LAN is an *extended LAN, in which static and mobile hosts transparently communicate using traditional wired technologies or ad hoc networking technologies.*

The characteristics of the ad hoc networking differ substantially from the conventional IP architecture. Therefore, the interconnection of ad hoc networks to the fixed Internet brings up several issues regarding addressing, routing, mobility management and gateway selection. In this section, I discuss in particular the substantial conflicts between the routing and addressing architectures of MANETs and the fixed Internet.

One of the fundamental characteristics of the fixed Internet is the adoption of a hierarchical addressing architecture with the IP addresses divided into a network identifier and a host identifier. All the hosts in a certain network segment have to share the same network identifier (also called network prefix). Consequently, the IP address has a location-specific topological meaning and the conventional IP routing protocols can use one single route to address an entire IP subnet (i.e., a network consisting of hosts that share the same network prefix). Note that the use of structured addressing in the Internet has been a fundamental factor in enabling Internet scalability. On the contrary, in the traditional view of ad hoc networking structured addresses are not required and IP addresses have been seen as unique identifiers without a specific topological meaning. Consequently, ad hoc networks are usually autonomous systems without hierarchy and organized adopting a flat private address space. Thus,

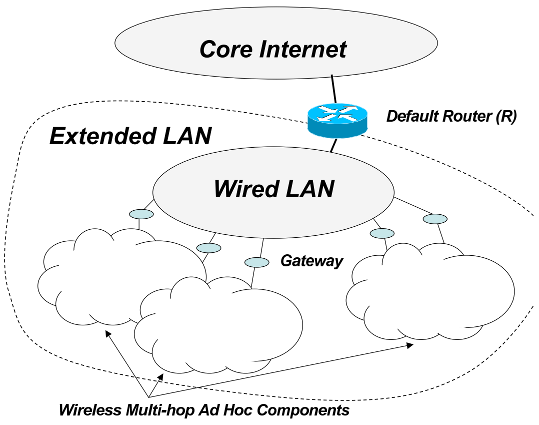


Figure 8.1: Reference network architecture.

routing in ad hoc networks is typically performed using host specific routes. It is evident that non-trivial issues arise to allow the interoperability between routing protocols adopting either network-specific routes or host-specific routes.

Another thing that distinguishes ad hoc routing and conventional IP routing is the use of multi-hop communications within the ad hoc network. This implies that hosts residing in the same network do not always share a common physical link. On the contrary, nodes in a fixed Internet sharing the same network prefix expect to have link-layer connectivity. Several basic mechanisms employed by conventional IP-based protocols (such as Mobile IP) heavily rely on this assumption. This means that these mechanisms should be modified and/or extended to be applied in multi-hop ad hoc networks.

Another problem that needs to be addressed to interconnect ad hoc networks to the fixed Internet is how to determine that an address is not present in the ad hoc domain. This is a trivial issue when using a structured addressing architecture, because it is always possible to decide whether a destination is located within same network by simply looking at the destination's network prefix. Furthermore, in this case default routes can be used when no route exists to that destination. On the other hand, these features cannot be considered as implicitly granted by the ad hoc routing protocol. For these reasons, several ad hoc routing protocols implement specific route discovery mechanisms that are executed before deciding whether the destination is within the ad hoc network or not.

When the ad hoc node has somehow determined that the destined node is not present on the MANET, it has to detect available gateways that it can use to reach that destination. Thus, specific gateway discovery procedures and gateway selection algorithms should be designed. In addition, gateway selection is particularly critical when managing mobility inside the ad hoc network. In fact, frequent and unnecessary changes of selected gateways can introduce a significant burden to the routing protocol and drastically degrade the performance of transport-layer connections.

An issue that is strictly related to the gateway selection is how to ensure that the incoming traffic returning from the Internet is correctly routed to the gateway and then back to the originator ad hoc node. To allow this, the ad hoc node needs an IP address that is routable from the rest of the Internet. A variety of solutions can be devised to accomplish this, such as either implicit address translation at the gateway or some kind of explicit signalling between the ad hoc node and the gateway. However, at least the gateway must have a globally routable IP address because it resides on the edge between the ad hoc domain and the external Internet.

Chapter 9

Existing solutions

The following subsections review the existing solutions for enabling the internetworking of ad hoc and fixed networks, pointing out how they cope with the main challenges introduced in Section 8. This survey is focused on the mechanisms that can be applied to IPv4-based MANETs because the mechanism proposed in this thesis is especially designed for IPv4 networks. For the sake of completeness, the end of this section also briefly discuss the approaches proposed for IPv6 networks, providing references to the most consolidated proposals.

9.1 Mobile IP-based approaches

One of the possible approaches to provide Internet connectivity for ad hoc networks is based on the integration of Mobile IP[1] [99] with the ad hoc routing protocols. The reader is referred to Section 6.1 for an overview of the Mobile IP protocol.

The key idea behind the integration of Mobile IP and ad hoc networking is to set up a Mobile IP Foreign Agent (MIP-FA) in the gateways that interconnect the ad hoc network with external networks, and to run an extended version of classical Mobile IP in the MANET. *This approach requires that each mobile node has a permanent and unique routable IP address* (i.e., an “home address”). When a mobile node visits an ad hoc network it adopts its home address as a unique identifier, and exploits conventional ad hoc routing protocols to establish a multi-hop route with one of the foreign agents present in the MANET. After registering with one of the available MIP-FA gateways, the visiting node will be globally routable on the Internet. Although the basic design principles are simple, several problems arise when adapting the Mobile IP to operate in multi-hop ad hoc networks. The first obstacle comes from the fact that standard Mobile IP protocol assumes that FAs and visiting nodes have link-layer connectivity to rely on link-local broadcast for signalling and control. On

[1]In this section I consider only Mobile IPv4 and the IP version is omitted for brevity.

the contrary, in multi-hop wireless networks broadcast transmissions consume a great amount of network resources (i.e., bandwidth and energy) because broadcast messages flood the whole ad hoc network [88]. In addition, the Mobile IP protocol adopts a proactive approach for agent discovery and for executing movement detection and handoff, which is based on periodic broadcasting of agent advertisements. In some situations, this behaviour clashes with the ad hoc routing protocol operations, especially when they are carried out in an on-demand manner. Finally, since the addressing architecture in ad hoc networks is usually flat, specific techniques have to be designed to allow mobile nodes to decide whether a destination is located within the ad hoc network or in the Internet. The remaining of this section, reviews the different solutions that have been proposed to cope with these problems, both for proactive and reactive ad hoc networks, pointing out advantages and limitations of each of them.

Early research on MANETs focused on the design of proactive ad hoc routing protocols. Consequently, initial solutions for interconnecting ad hoc networks to the Internet described mechanisms for using Mobile IP on top of proactive routing. For instance, in [31] an initial design is presented to integrate standard IP routing to DSR [68], a source-based on demand ad hoc routing protocol. The basic principle is that each node participating in the same ad hoc network selects its home address from a common IP subnet. In this case, the gateway can be configured as a standard IP router between the MANET subnet and the external Internet. Mobile IP is then used to support the migration of mobile nodes from the Internet into and out of ad hoc networks. The idea is that mobile nodes piggyback Mobile IP AGENT SOLICITATION on DSR ROUTE REQUESTs to register with the FA colocated with the gateway. In addition, when the mobile node wants to communicate with a node outside of the ad hoc network, the FA sends a *proxy* ROUTE REPLY listing itself as the last hop in the route to the intended destination.

An alternative, and more elaborate, solution to provide Internet connectivity for reactive ad hoc networks is described in [69], and it is known as MIPMANET. This scheme assumes that ad hoc nodes forward traffic using AODV [100], a table-driven on-demand routing algorithm. In contrast with the approach adopted in [31], the MIPMANET architecture does not require that the mobile nodes' home addresses belong to the same IP subnet. When a mobile node wants Internet access it simply register its arbitrary home address with one of the available FAs, and tunnels each packet to the foreign agent with whom it is registered, which decapsulates the packets and forwards them to the destination. This tunnelling is used to emulate the concept of default routes into the on-demand ad hoc routing protocol. Since MIPMANET does not impose any network-prefix semantic within the ad hoc network, the sender must initiate a route discovery process using the AODV routing protocol to decide whether a destination is located within the ad hoc network or not. If the destination is not found within the ad hoc network, than the mobile node infers that the destination is in the wired Internet and tunnels the packets addressed to that destination to the FA. Furthermore, MIPMANET uses reverse

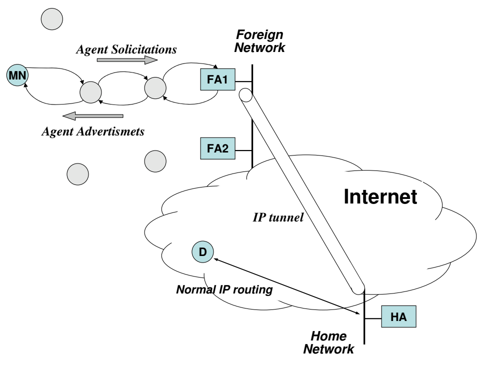


Figure 9.1: Illustration of the MIPMANET solution.

tunnelling as defined in [84], establishing an IP tunnel both in the forward direction (from home agent address to foreign agent care-of-address) and in the reverse direction. Figure 9.1 illustrates the fundamental components of the MIPMANET architecture. Concerning the foreign agent discovery mechanism, the MIPMANET scheme adapts the Mobile IP protocol to operate in a more on-demand fashion allowing FAs to periodically unicast Agent Advertisement messages to registered nodes. In addition, MIPMANET utilizes a new algorithm, called MIPMANET Cell Switching (MMCS), to determine when mobile nodes should register with a new foreign agent upon moving.

In [115] a solution similar to MIPMANET is presented to implement a Mobile IP foreign agent on a gateway for ad hoc networks running the AODV routing protocol. However, the scheme proposed in [115] adopts a somehow simpler approach, limiting the use of IP tunnels inside the ad hoc network. More precisely, MIP-FA gateways periodically advertise their presence through Agent Advertisement messages, and each mobile node maintains a list containing the IP addresses of available FAs. A mobile node that wants Internet access can register with the closest FA one of the care-of-addresses announced in the received Agent Advertisement messages, and then it updates the location information maintained by its HA. In addition, mobile nodes can proactively search for foreign agents by issuing RREQ packets addressing the *All Mobility Agents* multicast group [99]. When a FA receives a RREQ for a destination to which it has not an explicit route (note that the FA has explicit routes only for mobile nodes registered with it) it replies with a special RREP, called FA-RREP, informing the sender that the destination can be reached through the gateway. In other words, the gateway generates *proxy* RREP packets on behalf of nodes that might be present on the external Internet. Upon receiving a FA-RREP the source node does not use immediately this route, but waits for receiving normal RREP messages indicating that the destination node is located within the ad hoc network. Moreover, by properly setting the destination sequence numbers in the RREP message it is also possible to ensure that routes to nodes in the MANET will always have higher priority than routes established using FA-RREPs sent by gateways. The advantage of using FA-RREPs is that data packets can be transmitted to the FA using standard IP forwarding and it is not needed to use tunnelling within the ad hoc network.

The solutions described so far integrate reactive ad hoc routing protocols (DSR or AODV) with the Internet routing and the Mobile IP architecture. However, the basic design of many of the Mobile IP mechanisms, such as agent discovery, movement detection and reachability of the mobile node, follows proactive principles. Thus, it is opportune to find a trade-off between the on-demand operations of reactive ad hoc routing protocols and the overhead introduced by periodically broadcasting agent advertisements. For instance, MIPMANET proposed to use the MMCS algorithm [69] to detect and move to new foreign agents. According to this algorithm, a mobile node should register with another foreign agent if it is at least two hops closer to this new foreign agent than to its previous foreign agent, for at least two consecutive agent advertisements. In [105] it is described a protocol that limits the flooding of agent advertisements in an n-hop neighbourhood by using TTL scoping. In [7], Mobile IP is extended to manage multiple simultaneous connections with foreign networks. Based on the registered care-of addresses, multiple paths can be used for packets to and from a mobile node. Thus, enhanced throughput and a more reliable connection can be achieved. An alternative scheme is proposed in [30]. In that work, some heuristics are described to classify the traffic load of the gateways, such as to avoid selecting congested route to gateways. In addition, to reduce the flooding overhead due to solicitations, optimized searching algorithms are used such as the Expanding Ring Search method [100]. However, some intrinsic limitations of on-demand routing protocols as the inability to support default routes and to easily determine that an address is not present on the MANET, cannot be overcome without relying on complex address allocation schemes or resource-consuming IP-in-IP encapsulation [97].

For these reasons, recently, the integration of Mobile IP with proactive ad hoc routing protocols has gathered a lot of attention in the research community. A hierarchical approach to accomplish this integration is proposed in [26]. More precisely, Mobile IP is used to support macro-mobility between different IP domains, while the OLSR ad hoc routing protocol is adopted to support micro-mobility inside the MANET environment. As in prior work, the architecture proposed in [26] contains a gateway implementing a MIP-FA, allowing the OLSR-IP access network to be connected to the Internet. In addition, in this hierarchical architecture special nodes, called OLSR *Base Stations*, have been introduced to reduce the number of global location updates performed by Mobile IP. These base stations have both wired and wireless interfaces and implement the OLSR protocol on both of them. When a mobile node enters an IP-OLSR access network, it receives either periodic AGENT ADVERTISEMENT messages broadcasted by the gateways or unicast AGENT ADVERTISEMENT messages sent by the gateways in reply to the node's AGENT SOLICITATION. Once the mobile node is registered, it is attached to the OLSR-IP access network. This implies that the node has a host specific entry in the routing table for each IP destination address known locally on the MANET, while all the traffic for external networks is forwarded along a possible default route out of the MANET through the gateway. The HNA messages[2] issued by the gateway establish this binding between the gateway itself and the external networks. In such a way, it is not needed to have an explicit procedure to determine if a destination is present or not in the MANET. Note also that the use of IP tunnelling is limited to the communications between FA and HA, but it does not occur within the ad hoc network.

Albeit the considerable effort devoted to the design of solutions enabling an efficient integration of Mobile IP with the ad hoc routing protocols to provide a global mobility between Internet and MANETs, Mobile IP-based solutions have still a number of drawbacks. The first one is that in order to allow Mobile IP and ad hoc networking to cooperate it is needed to introduce further complexities and sub-optimal operations in the implementations of both Mobile IP and ad hoc routing protocols. Probably an integrated design of Mobile IP and ad hoc routing functionalities might be much more efficient, minimizing the overheads. In addition, Mobile IP was designed to handle mobility of devices in case of relatively infrequent mobility. Thus the overheads introduced to manage roaming and handoffs between foreign agents are a relevant issue in MANET environments, where handoff functions operating at the data link layer may be more suitable. Finally, when the technique of default routes is used to route the traffic from the mobile node to the closest gateway, the use of Mobile IP can easily lead to triangle routing. In fact, when the mobile node moves it can get closer to a gateway different from the one to which it is currently registered. As a consequence, the forward traffic leaves the ad hoc network through one gateway while the return traffic enters the ad hoc network through the new MIP-FA gateway to which the mobile node is registered. To solve this problem the authors of [48] have proposed to use Mobile IP reverse tunnelling [69], or explicit tunnelling to one of the gateway instead of using default routes. Note that changing between two MIP-FA based gateways do not cause a transport-layer session break because the mobile node has simply to register the new FA with its HA. As I will discuss in the following section, changing gateways in NAT-based solutions is much more critical.

9.2 NAT-based approaches

An alternative category of solutions to interconnect MANETs to the Internet is based on the implementation of Network Address Translation (NAT) modules [109] on the gateways. Traditionally, basic NAT is used to allow hosts in a private network to transparently access the external Internet. More precisely, a router with a NAT module implemented on it, behaves as an address translator, which dynamically maps the set of non-routable private addresses (selected from a reserved address range) used internally in the local domain to

[2]See Section 4.2 for a description of the OLSR protocol.

a set of globally routable network addresses. A variant of basic NAT is the Network Address Port Translation, or NAPT [109], which translates private IP addresses to a single globally routable IP address by using different transport layer ports. When implementing a NAT module in a gateway of the hoc network, this gateway translates the source IP address of outgoing packets with its IP address, which is routable on the external network. The return traffic is managed similarly, with the destination IP address (i.e., the NAT-gateway address) replaced with the IP address of the ad hoc source node. Note that *NAT-based approaches impose a limited addressing structure within the ad hoc network*. In fact, the ad hoc network is identified by one or more network-prefixes designated to be used in private networks, while the host-specific part of the address is autonomously administered within the MANET.

One of the earliest implementation of this method for reactive ad hoc networks was developed by the Uppsala University for the AODV routing protocol [1]. In that implementation mobile nodes are not aware of the available gateways enhanced with NAT capabilities, and these gateways use Proxy RREP messages to reply to RREPs destined for hosts on the Internet. The use of Proxy RREP messages is clearly inspired by work done in [31, 115]. To reduce the complexity of the implementation, the AODV-UU NAT solution assumes that all the addresses on the ad hoc network are allocated from the same private IP subnet. Thus, the gateways replies only to RREQ messages targeting destinations external to the ad hoc network. To avoid this limitation, it is possible to apply solutions similar to the ones defined in [115] for a MIP-FA based gateway. More precisely, the NAT gateway can be allowed to reply to all the received RREQ messages generating a Proxy RREP with the same sequence number of the received RREQ. Hence, a direct route to the designated destination not traversing the gateway will always have preference on the route announced by Proxy RREP messages.

Solutions based on the use of Proxy RREPs do not work correctly in multihomed networks, i.e., when multiple gateways are present in the ad hoc network. Indeed, to avoid transport-layer session breaks it is necessary to ensure that each packet from the same session is routed over a specific gateway since a NAT router translates both outgoing and incoming packets. However, it is difficult to control the gateway selection for an ad hoc node that is moving. To solve this problem, in [47] an alternative approach to provide Internet connectivity for on-demand routing protocols is described. First of all, the solution proposed in [47] defines a method for implementing gateway discovery using the AODV protocol. When a mobile node searches a route to its designated destination, it initially floods the network with normal RREQ messages. If the source node does not receive any reply, it issues a special RREQ message with a "Gateway"-flag set. Upon receiving this type of RREQ messages the gateways are allowed to reply with Proxy RREPs on behalf of external nodes. When the source node receives these replies, it becomes aware of the available gateways. Then, the ad hoc node selects one of these gateways according to some heuristic and it tunnels the packets addressed to external destinations to the selected gateway using for example IP-in-IP encapsulation [97]. Furthermore, the NAT gateway

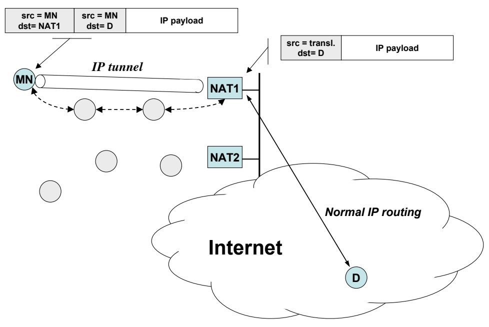


Figure 9.2: Illustration of the tunnelling operations in NAT-based solutions [48, 47].

will also tunnel the incoming packets returning from the Internet to the source nodes. Figure 9.2 illustrates how the IP-in-IP encapsulation method works when tunnelling the packets for an external host via a NAT gateway.

As described in [48], NAT gateways can be implemented also in proactive ad hoc networks. By using proactive routing protocols it is easy to advertise gateways within the ad hoc network, and default routes can be established between the mobile nodes and the closest gateway. However, instead of using default routes to send packets addressed to host located in the external Internet as proposed in [26], the solution proposed in [48] requires that each sender node establishes a tunnel with the selected gateway (e.g., by using IP-in-IP encapsulation [97] or minimal IP encapsulation [98]) to deliver packets addressing external IP networks. Employing explicit tunnelling instead of default routes ensures that each packet of the same transport-layer session is consistently routed through the same gateway, even if the source node moves. In papers [47] and [48] it is also addressed the problem of how enabling gateways adopting different mechanisms to cooperate while providing Internet connectivity. For instance, in multi-homing scenario some gateways may be NAT-based, and other gateways may be MIP-FA based. Again, a working solution for this issue is the use of explicit tunnelling that forces the packets generated from the same session to be routed the gateway selected at the beginning of the session itself. This prevents the session break when the source node gets closer to another gateway. However, this method requires that special techniques be implemented to allow source nodes to discover the different capabilities of the available gateways. To this end, extensions of the ad hoc routing protocols have to be devised. Finally, it is worth pointing out that most of the schemes described in this section rely heavily on the use of IP tunnels. However, tunnelling introduces a fixed overhead in the tunnelled IP packet headers. For instance, in case of IP-in-IP encapsulation 20 bytes of overhead are added in every outgoing packet. Experimental results presented in [48] indicate that in some situations the throughput obtained by a TCP session can suffer a 30% decrease. This clearly motivates the need of designing more efficient and lightweight solutions to provide Internet connectivity for ad hoc networks.

Although the NAT-based approach has undoubtedly advantages, such as to allow the ad hoc network to continue to use a private address internally, as well as it simplicity, it also arises several concerns from the application layer standpoint. In particular, NAT is not very suitable for incoming connections and this fact causes significant difficulties for peer-to-peer applications. Recently a few workarounds have been proposed to reduce the problems created by NAT, such as NAT-traversal techniques. However, a clear solution interoperable with existing NAT devices is still not available.

9.3 Layer 2.5 solutions

Recently other schemes have been proposed to provide ad hoc support below the IP layer. For example, in [6] label switching was employed to put routing logic inside the wireless network card. More recently, the LUNAR [121] ad hoc routing framework and the Mesh Connectivity Layer (MCL) [43] have been proposed. These solutions locate the ad hoc support between the data link layer and the network layer. This "layer 2.5" is based on *virtual* interfaces that allow abstracting the design of ad hoc protocols from both the specific hardware components and the network protocols. However, this interconnection layer requires its own naming and addressing functionalities distinct from the layer-2 addresses of the underlying physical devices. This significantly increases the system complexities and introduces additional header overheads. On the contrary, my proposed architecture exploits existing ARP capabilities, reducing implementation complexity, providing fully backward compatibility and ensuring minimal overheads.

9.4 Proposals for IPv6-based MANETs

Most of the research aimed at combining IPv6 and ad hoc networking has focused on the *design of mechanisms to configure globally routable IPv6 addresses within a MANET*. In general, these approaches treat the ad hoc network as an IPv6 subnet. According to this view, gateway nodes present in the ad hoc network act as default routers receiving all the traffic destined for IP subnets outside the MANET. Consequently, in this category of solutions, gateway discovery mechanisms and address autoconfiguration techniques are strictly interdependent.

The standard IPv6 architecture includes a stateless address autoconfiguration technique [119] but it cannot be applied in multi-hop topologies. There have been numerous proposals to extend this autoconfiguration protocol to operate in a MANET considering multi-hop operations, network partitioning and merging (for example, see [49]). However, the proposals that have received more attention in the research community are the schemes that deal both with address autoconfiguration and gateway discovery. One solution is described in [124], which defines both proactive and reactive strategies to discover the gateways within the ad hoc network. The network prefix, which is distributed by these Internet gateways, can then be used for configuring a (typically globally) routable IPv6 address for each ad hoc node. In [124] it is also described how Mobile IPv6 can be modified to be used in ad hoc network. In particular, Internet gateway advertisements will be used to detect node's movement instead of the neighbour discovery mechanisms used in conventional Mobile IPv6. In addition, the mobile node uses the globally routable address acquired from the Internet gateway as its care-of-address. Hence, no foreign agents are needed in the ad hoc subnet. Finally, packets sent outside the MANET contain a routing header that includes the address of the default gateway. This is a sort of emulation of the tunnel towards the gateway used in the MIPMANET protocol to avoid address reconfigurations after gateway handoffs. An alternative solution to interconnect IPv6-based MANETs to the fixed Internet is described in [66]. This scheme introduces the concept of "prefix continuity". More precisely, in the same MANET multiple subnets (i.e., network prefixes) can be used, but the network identifiers are assigned to visiting nodes such as that any node that selected a given prefix has at least one neighbour with the same prefix on its path to the selected gateway. This implies that the MANET is organized in clusters of hosts sharing the same network prefix. This network organization reduces the overheads introduced by flooding gateway advertisements.

For a comprehensive survey of the several approaches proposed for enabling Internet connectivity for IPv6-based MANETs, and a detailed description of the protocol operations, the interested reader is reminded to [106] and references herein.

Chapter 10

Proposed Architecture

The basic design principle I adopted during the definition of my proposal was to provide transparent communications between static nodes, which use traditional wired technologies, and mobile nodes, which use more advanced ad hoc networking technologies, employing mechanisms that run below the IP layer. As discussed in the introduction, in this work I address two major problems: address autoconfiguration and global Internet connectivity. However, before describing the details of my solutions, it is useful to illustrate the complete network architecture I propose for interconnecting heterogeneous ad hoc networks to the Internet. To this end, Figure 8.1 depicts the reference network architecture I consider in this work.

As illustrated in the figure, I envision an *extended LAN composed of a conventional LAN (the wired component) and several ad hoc clouds*. In this network mobile/portable nodes not in close proximity of the fixed networking infrastructure establish multi-hop wireless links to communicate with each other (e.g., using the IEEE 802.11 technology [5]) using an ad hoc routing protocol. The wired LAN and the ad hoc components are interconnected using gateways, which are special nodes provided with both wired and wireless interfaces. I also assume that the ad hoc routing protocol is running on both the gateways' interfaces. This implies that separate (i.e., not in direct radio visibility) ad hoc clouds are not disconnected because an ad hoc node can exchange routing messages with any other ad hoc node in the extended LAN also through the wired LAN. As explained in later sections, this architectural design allows transparent support for node mobility and facilitates the Intranet communications.

In my architecture multi-homing is permitted, i.e., multiple gateways can be located within the same ad hoc component. Consequently, specific mechanisms are required to support the handoff between gateways without transport-layer connection breaks. In general, between pairs of gateways in radio visibility of each other, two direct links can be established, both wired and wireless. The choice between the two links is demanded to the routing protocol. However, we can assume that the wired link has a lower link cost than the wireless link because wired technologies are still more reliable and have higher capacity than standard wireless technologies. As a consequence, a routing protocol selecting least-cost paths will always give preference to the wired path for inter-gateway communications. In Section 10.2, I will explain the implications of this assumption.

Concerning the addressing architecture of my network, I assume that *the extended LAN is a single address space.* Namely, all the nodes in the extended LAN, both ad hoc nodes and static ones, have an IP address with the same network identifier. To indicate the network identifier I use the standard notation $\mathrm{IP_S/L}$, in which $\mathrm{IP_S}$ indicates the network prefix, and L is the network mask length. For instance, in my test-bed I used $\mathrm{IP_S/L} = X.Y.96.0/22$. It is worth pointing out that *my solution does not impose any addressing hierarchy within the extended LAN*, and both ad hoc and wired nodes may have an arbitrary IP address belonging to the $\mathrm{IP_S/L}$ subnet. This implies that *the wired nodes are not aware of which are the ad hoc hosts and vice versa.*

Standard IP routing is used to connect the extended LAN to the core Internet. Regarding the ad hoc routing protocol, my scheme is specifically designed for being integrated with proactive routing protocols. Examples of these type or routing protocols for MANETs are the Optimized Link State Routing (OLSR) protocol [37] or the Topology Dissemination Based on Reverse-Path Forwarding (TBRF) routing protocol [89]. The motivation behind this design choice is that proactive routing protocols usually support gateway advertisements, allowing the gateways to use special routing messages to set up specific default routes in the ad hoc network. In addition, proactive routing protocols, adopting classical link state approaches, build complete network-topology knowledge in each ad hoc node. This topology information could significantly simplify the operations needed to acquire Internet connectivity. In this work, the reference ad hoc routing algorithm is OLSR, but my architecture is general and it is equally applicable to other proactive routing protocols.

To build and make operational this extended LAN, I have designed three main mechanisms:

- An address autoconfiguration protocol for the ad hoc nodes that takes advantage of the DHCP servers located in the wired LAN, which does not require that mobile nodes are aware of the identity/location of these DHCP servers;

- An adaptive proxy-ARP capability, which allows the gateways to intercept packets generated by the wired nodes and addressing the ad hoc nodes;

- An automatic procedure to set the proper network-specific routing entries needed by the ad hoc nodes to correctly forward their traffic to the external networks.

The following sections describe the operations performed by the aforementioned techniques and how the ad hoc components are transparently integrated into the wired infrastructure.

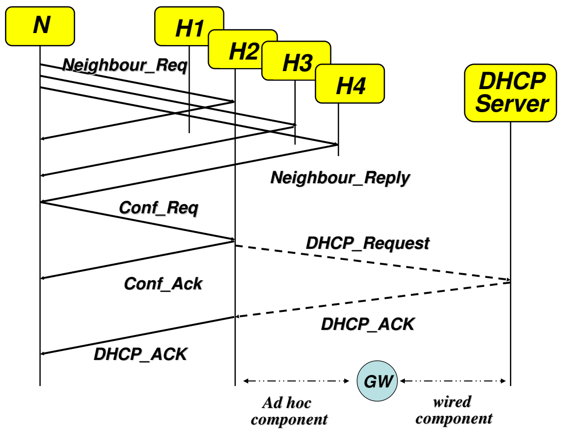


Figure 10.1: Message exchanges during the address autoconfiguration.

10.1 Address autoconfiguration

A prerequisite for proper routing is that all nodes are configured with a unique address. Thus, various protocols have been proposed in the literature for the purpose of address autoconfiguration in MANETs. Generally speaking, with protocols using *stateless approaches* nodes arbitrarily select their own address, and a Duplicate Address Detection (DAD) procedure is executed to verify its uniqueness and resolve conflicts. On the other hand, protocols based on *stateful approaches* maintain either centralized or distributed address allocation tables, and they execute distributed algorithms to establish a consensus among all the nodes in the network on the new IP address, before assigning it. Protocols proposed in [122] and [86] are well-known examples of the former and latter approach, respectively. A limitation of most of the early solutions is that they are designed to work in *stand-alone* MANET. To address this problem, the Internet Engineering Task force (IETF) has established the *AUTOCONF* working group with the main purpose of standardizing mechanisms to be used by ad hoc nodes for configuring unique local and/or globally routable IPv6 addresses[1]. In fact, it is now evident that using a unified strategy to select a global node address, to route the packets and to access the Internet may be beneficial, because complexities and overheads are reduced.

In this work, I propose a simple but effective protocol to assign a globally routable IPv4 address to the mobile nodes of the extended LAN by taking advantage of the easy access to the fixed infrastructure. In general, in a wired LAN unconfigured hosts may use the Dynamic Host Configuration Proto-

[1]For a general overview of the main approaches for address autoconfiguration in MANET, and the description of the most well-known protocols the reader is referred to [126]. In addition, draft specifications from the Autoconf WG can be found at `http://tools.ietf.org/wg/autoconf/`.

col [44] to query centralized servers to obtain IP configuration parameters (i.e., a unique IP address, a common netmask and, eventually, a default gateway). However, this solution is not straightforwardly applicable to ad hoc networks because the node running the DHCP server may not be permanently reachable by all nodes. In addition the legacy DHCP-server discovery procedure requires link-level connectivity between unconfigured nodes and the DHCP servers. To overcome these limitations, my scheme assumes that DHCP servers are located only in the wired LAN and employs a novel mechanism to allow unconfigured nodes to contact the DCHP servers through multi-hop paths. To achieve my goal I partially reuse some mechanisms from the MANETconf protocol described in [86]. In particular, in MANETconf a node requesting an address first searches for already configured nodes and selects one of its neighbours as initiator of the configuration process. However, MANETconf can be used only to allocate unique and private addresses to nodes in a stand-alone MANET. In my scheme, a mobile node not yet associated with the extended LAN executes a preliminary message handshake to discover reachable and already configured ad hoc nodes. Then, the unconfigured node that wants to join the ad hoc component elects one of the discovered neighbours as DHCP relay agent. A DHCP relay is an entity that is capable of relaying DHCP_DISCOVER broadcasts from a LAN which does not include a DHCP server to a network which does have one [44]. The proposed address autoconfiguration protocol is basically an extension of the functionalities of the legacy DHCP relay agents. More precisely, as illustrated in Figure 10.1, an unconfigured node N broadcasts special messages, called NEIGHBOUR_REQ, to discover other nodes that are within its radio visibility and that can interconnect him to the ad hoc component. To make the protocol more robust against messages losses and network dynamics, node N can periodically generate new NEIGHBOUR_REQ messages scanning each channel and operating mode (e.g., 802.1 $a/b/g$) supported by its interface. When a node that is already part of the ad hoc network correctly receives a NEIGHBOUR_REQ message, it discovers the physical address of the node N and it can unicast a NEIGHBOUR_REPLY message to node N. From the received NEIGHBOUR_REPLY messages, node N can build a list of available DHCP relay agents, and it will select one of them according to some heuristic (e.g., the one with the best signal quality, or the last one to reply). In the example shown in Figure 10.1, node N has selected node $H2$ as its proxy DHCP relay agent. Node $H2$ is informed about this choice through a CONF_REQ message. Node $H2$ acknowledges the correct reception of this request sending a CONF_ACK message to node N. After that, node $H2$ activates its *proxy* DHCP relay agent, which initiates the process of address assignment on behalf of node N using the standard DHCP protocol. Note that node $H2$ can contact the DHCP servers located in the wired LAN using unicast DHCP control messages, because node $H2$ is already part of the ad hoc component. The DCHP messages generated by node $H2$ are routed through one of the gateways according to the mechanisms that will be described in the following sections. The DHCP server receiving the request, will answer to the DHCP relay agent with a DHCP_ACK, containing the IP configuration parameters for the new node N. The configuration process is concluded when the DHCP relay forwards the DHCP_Ack message to the initial node N, which can now join the network. After joining the ad hoc component, node N may also turn itself into a DHCP relay agent for the DHCP server from which it received the IP configuration parameters, in order to support the configuration process of new mobile nodes. Finally, it is worth noting that my scheme does not need any initialization procedure because the gateways are directly connected to the wired LAN and can broadcast a DHCP_Discover message to locate available servers. Hence, the first mobile node to enter the ad hoc network may find at least one gateway capable of initiating the illustrated configuration process.

A limitation of the proposed autoconfiguration protocol is that the wired LAN should be permanently reachable by the ad hoc nodes in order to permit renewing the address lease with the DHCP servers. However, in the considered network scenario this limitation may be considered not problematic because it is reasonable to assume that the network is not highly dynamic. In fact, the extended LAN I envision will be used mostly as a flexible and cost-effective extension of the fixed networking infrastructure in enterprise buildings or campus facilities. In these contexts, users are semi-static or nomadic and are interested in having a continuous access to Internet and its centralized services (e.g., web browsing, access to centralized data repository, etc.).

10.2 Interconnecting ad hoc nodes to the fixed Internet

The basic assumptions of my architecture are the following: *i*) A proactive ad hoc routing protocol is used in the ad hoc components. This implies that ad hoc nodes' routing tables are populated with entries specifying the next-hop neighbour to forward a message to any other ad hoc node in the extended LAN; and *ii*) all the hosts in the extended LAN share the same IP network prefix. As previously introduced, the key mechanisms I propose to interconnect the ad hoc components (see Figure 8.1) to the wired LAN and the external Internet rely on the use of network-specific routes and some extended functionalities of the ARP protocol. For clarity, in the following I separately describe how communications are established and maintained between ad hoc nodes and hosts in the wired LAN (i.e., Intranet communications) or hosts in the external Internet (i.e., communications routed through the default router R shown in Figure 8.1).

10.2.1 Intranet connectivity.

As discussed in Chapter 8, the main problem to solve to support the Intranet connectivity is to guarantee that each ad hoc node can identify whether the destination node is within the ad hoc part or the wired part of the network, and vice versa. To explain how my scheme addresses this issue let us initially analyse the case of an ad hoc node N that wants to communicate with a wired host H. To perform its routing decisions, normally node N has three types of routing information in its routing table. First, the node N's routing table contains host-specific entries to reach any other ad hoc node, which are inserted by the proactive ad hoc routing protocol. Second, there is an entry that is automatically inserted by the TCP/IP protocol stack at the network boot, which provides the local reachability on its wireless interface. Namely, since the node N's IP address belongs to the IP subnet identified by the $\mathrm{IP_S/L}$ network/mask pair, then N's routing table contains a generic entry that indicates that all the IP addresses matching this network prefix may be directly reached through the wireless interface if a more precise routing entry is not known. Finally, node N's routing table contains the generic routing entry 0.0.0.0/0, which is the default route advertised by the gateways. When the ad hoc node N wants to send a packet addressed to a wired node H, node N checks its routing table. Since node N has not a host-specific routing entry for node H's IP address, it believes that node H is directly reachable on its wireless interface, and it will issue an ARP_REQUEST message targeting node H's physical address. This ARP_REQUEST fails because node H is not directly reachable on node N's wireless interface. To solve this inconsistency, node N needs a specific mechanism to discover that node H can be reached only through a gateway, although it shares with node N the same network prefix. To this end, I exploit the properties of the longest-matching rules used by the standard IP routing. Specifically, I configure the gateways such as to advertise two additional and more precise network-specific routes, which announce the reachability of the wired LAN through the gateways' wired interfaces. These network-specific routes are addressing the $\{\mathrm{IP_{S_{Low}}}/(\mathrm{L}+1)$ and $\{\mathrm{IP_{S_{High}}}\}/(\mathrm{L}+1)$ subnetworks, i.e., the two disjoint IP subnets whose union is equal to $\mathrm{IP_S/L}$[2]. The use of these additional network-specific routes, which are more precise than the route providing local reachability to $\mathrm{IP_S/L}$, ensures that each ad hoc node will forward the traffic to its closest gateway to communicate with any host on the local wired LAN.

The use of the two network-specific routes previously specified is a simple but effective way to guarantee correct routing of egress traffic from the ad hoc components. However, they may cause an inconsistent behaviour in the gateways. In fact, each gateway can directly communicate with a wired host H through its wired interface. On the other hand, each gateway also receives HNA messages sent by other gateways, setting up the additional routing entries advertised in these messages. Hence, when a gateway wants to send packets to a wired host on the local wired LAN (e.g., node H), the routing table lookup will choose one of the routing entries targeting the $\{\mathrm{IP_{S_{Low}}}\}/(\mathrm{L}+1)$ and $\{\mathrm{IP_{S_{High}}}\}/(\mathrm{L}+1)$ subnets, instead of the entry indicating the local reachability of the host H on the gateway's wired interface. The effect is that the IP packet will loop among the gateways until the TTL expires, without reaching the correct destination H. To resolve this problem I again exploit the properties of the

[2]To clarify this concept, let us assume that $\mathrm{IP_S/L} = X.Y.96.0/22$. Then, this network prefix can be split into the two smaller subnets $\{\mathrm{IP_{S_{Low}}}\}/(\mathrm{L}+1) = X.Y.96.0/23$ and $\{\mathrm{IP_{S_{High}}}\}/(\mathrm{L}+1) = X.Y.98.0/23$. Note that this procedure is always possible for $L < 32$.

longest-matching rules used by the standard IP routing. Specifically, each gateway automatically configures two additional routing entries bound to its wired interface. These two additional entries have the same network/mask as the ones announced in the HNA messages (i.e., $\{IP_{S_{Low}}\}/(L+1)$ and $\{IP_{S_{High}}\}/(L+1)$), but with lower metric. Hence, when a gateway wants to communicate with a host in the wired LAN, it will always give preference to its wired interface.

Let us now consider the reverse direction, i.e., a wired host H that wants to communicate with an ad hoc node N. Since node H is running standard IP routing, normally its routing table will have only two types of routing entries. First, there is an entry that is automatically inserted by the TCP/IP protocol stack at the network boot, which provides the local reachability on its wired interface. Second, there is a generic entry related to the default router (node R in Figure 8.1) and used to forward the packets addressing external IP subnets. Hence, node H has no sufficient routing information to decide whether the destination is in the ad hoc or wired part of the network. As stated previously my objective is to design a solution that does not require modifications of routing behaviours in the pre-existing wired part of the network. To achieve this goal, I implemented in each gateway an enhanced proxy-ARP server that masquerades the IP addresses of all the ad hoc nodes reachable through the gateway's wireless interface. When node H wants to communicate with node N, it issues a conventional ARP_REQUEST message targeting node N's physical address because host H believes that node N is locally reachable on its wired interface. When a gateway receives this ARP_REQUEST searching for an IP address that is reachable through its wireless interface, the gateway publishes its MAC address. As a consequence, that gateway intercepts all the packets sent by node H to node N. The intercepted traffic is then forward to the designated destination inside the ad hoc network using the ad hoc routing protocol.

The proxy-ARP mechanism I have designed differs from the standard proxy-ARP functionality defined in the RFC 1027 [32], because it is capable of adapting to the topology changes. More specifically, each proxy-ARP server will proactively check the gateway's routing table to publish on the gateway's wired interface only the IP addresses related to host-specific routing entries bounded to the gateway's wireless interface (this condition ensures that the gateway is the closest one to that ad hoc node). When there is a change in the routing table due to node mobility, the proxy-ARP reactively updates the list of IP addresses published on the gateway's wired interface. This ensures a transparent handoff between different gateways, as explained in details in Section 10.2.3. Finally, it is worth pointing out that there are not interdependencies between the OLSR protocol and the proxy-ARP servers running on the gateways. In fact, the OLSR protocol maintains the gateway' s routing table, which is independently read by the local proxy ARP server to build its list of masqueraded IP addresses. Legacy operations of both OLSR and ARP protocols are not affected by the proxy-ARP server actions.

The last aspect that needs to be discussed is the existence of some network configurations where asymmetric routing may occur, i.e., the forward path is different from the return path. Let us consider the case in which node N

discovers two gateways, say $GW1$ and $GW2$, which are at the same distance (in terms of hops) from node N. In this situation, the OLSR routing algorithm will randomly select one of these gateways as default gateway for node N. However, both gateways are allowed to send ARP replies for ARP requests issued by the wired node H for the ad hoc node N's IP address. In this case, the wired node H will update its ARP table using the information delivered in the last received ARP_Reply. Let us assume that $GW1$ is the default gateway for node N, but $GW2$ has sent the last ARP reply to node H. In this case, node H sends the traffic destined to node N to $GW2$, which routes it to node N. On the other hand, node N sends packets destined to node H to $GW1$, which forwards them to node H. It is important to note that asymmetric paths are not necessarily by themselves a problem. Indeed, both node N and node H correctly receive and send their packets. In addition the asymmetric routing occurs only in symmetric topologies. Thus, it is reasonable to assume that in this local environment both paths are characterized by similar delays.

10.2.2 Internet connectivity

Providing Internet connectivity for the ad hoc nodes is now intuitive since Internet connectivity can be considered as a special case of the Intranet connectivity explained above. The only additional requirement is that the gateways know the default router's IP address. However, the default router for the LAN is one of the IP configuration parameters that are provided in the DHCP_Ack messages used to configure both wired hosts and ad hoc nodes. Thus, when an ad hoc node wants to send a packet addressing external IP subnets, it will simply forward the packet to the gateway. Then, the gateway will deliver the packet to the default router using the same mechanisms adopted to communicate with any wired host in the LAN. Similarly the incoming traffic received from the Internet and targeting the IP address of an ad hoc node, will be forwarded by the default router to the gateway that operates the proxy-ARP for that IP address. Note that *assuming a single public address space for the extended LAN permits a global reachability of the ad hoc nodes from nodes located in the external Internet*, avoiding the difficulties typical of NAT-based solutions.

10.2.3 Support for gateway handoffs

In general, solutions to support Internet connectivity for ad hoc networks using default routes may experience transport-layer session breaks when the default routes change, depending on the network dynamics. To avoid transport-layer session breaks, in [47] it was proposed to replace default routes with explicit tunnelling between the mobile nodes and the gateways. However, this significantly complicates the implementation and introduces relevant overheads as shown in my experiments (see Chapter 11). On the contrary, in my architecture the mobility is supported in a transparent way for the higher protocol layers. In fact, when an ad hoc node moves and gets closer to a different gateway, there is only a change in the ARP caches of the session endpoint located in

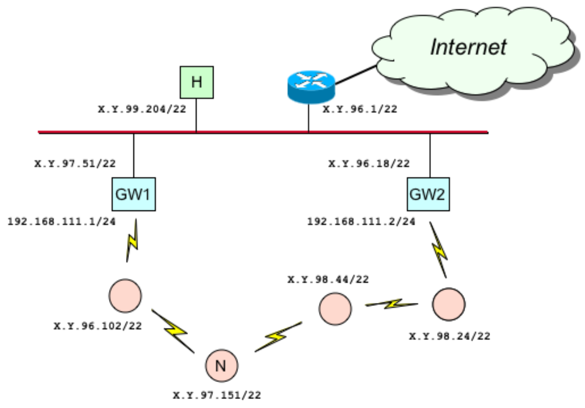


Figure 10.2: Illustrative Network Configuration.

the wired LAN, which does not cause the break of active transport sessions. To better clarify the actions occurring during an handoff let us consider an ad hoc node N with a TCP connection open with a wired node H, and using gateway $GW1$ as default route to reach the wired LAN. This implies that the node H's ARP cache contains a mapping between the node N's IP address and the gateway $GW1$'s MAC address. When node N moves and gets closer to a different gateway $GW2$ it updates its routing table and uses $GW2$ as new default gateway to reach the wired LAN. Simultaneously, $GW2$'s routing table also changes because the next hop for node N switches from $GW2$'s wired interface to $GW2$'s wireless interface. As a consequence, the proxy ARP running on $GW2$ inserts node N's IP address in the list of IP addresses the gateway $GW2$ publishes. In addition, it generates a GRATUITOUS ARP on the $GW2$'s wired interface for node N's IP address. This GRATUITOUS ARP updates the ARP tables in all the wired hosts that have a stale ARP entry for the node N's IP address, which was mapped with the MAC address of $GW1$'s wired interface. After this update the traffic destined to and/or originated from node N is correctly routed only through gateway $GW2$.

10.2.4 Example

To explain how the presented mechanism work, this section describes what happen in the simple network configuration depicted in Figure 10.2. For illustrative purposes I assume that the IP subnet of the extended LAN is $\mathrm{IP_S/L} = X.Y.96.0/22$[3]. I also assume that the mobile node involved in communication is the node N ($\mathrm{IP_N} = X.Y.97.151/22$).

Table 10.1 shows the node N's routing table. The entries 8, 9, and 11 are the ones induced by the HNA messages arrived from *GW1*. The entry 10 is

[3]On the gateways' wireless interfaces I set up private IP addresses to save address space. In this way, the gateways are globally reachable using the IP address on their wired interfaces.

automatically set up by the operating system when the wireless interface is configured with the IP parameters. Suppose that node N wants to deliver packets to the wired node H ($\mathrm{IP_H} = X.Y.99.204/22$). In this case, the routing table lookup on node N will indicate that the node H is reachable through the routing table entry 9. This because the routing table entry 9 is more specific than the entry 10. Consequently, the longest-match criterion applied to the routing table lookup, will result in node N correctly forwarding traffic to gateway *GW1* (i.e, the nearest one) to reach node *H*.

Table 10.2 shows the *GW1*'s routing table. In this example, *eth*0 is the *GW1*'s wireless interface and *eth*1 is the *GW1*'s wired interface. When gateway *GW1* wants to send packets to node *H*, it will found two routing table entries matching the same number of bits of node *H*'s IP address. These are entry 9 (derived from HNA messages received from *GW2*) and entry 11 (automatically configured on the gateway). However, entry 11 has a lower metric than entry 9 (i.e., metric 0 against metric 1). As a consequence, the packets destined to host H can be correctly forwarded to the host H on the local wired LAN through the *GW1*'s wired interface.

In the reverse direction the behaviour of the presented mechanism is as follow. Suppose that node H wants to send packets to the an ad hoc node *N*. Node H assumes that node N is on the same physical network. Hence, node H checks its ARP table for IP-MAC mapping and, if it is not present, it sends an ARP request. On the other hand, gateway *GW1* has an entry for node *N*'s IP address in it's routing table with a netmask 255.255.255.255, which is related to its wireless interface, while GW2 does not. Consequently, only *GW1* is allowed by the Proxy ARP server to answer with an ARP reply to node *H*. This ARP reply will insert the mapping [node *N*'s IP address - MAC address of *GW1*'s wired interface] into the node *H*'s ARP table. Thus, the packets sent from node H to node N will be delivered to *GW1*, which will forward them to node *N*. On the other hand, node N will reply to node H using *GW1*, as indicated by its routing table (see Table 10.1).

Table 10.1: Node N's Routing Table.

Entry	*destination*	*next hop*	*metric*	*interface*
1	$X.Y$.97.51/32	$X.Y$.96.102	2	*eth*0
2	$X.Y$.96.102/32	0.0.0.0	1	*eth*0
3	$X.Y$.98.44/32	0.0.0.0	1	*eth*0
4	$X.Y$.98.24/32	$X.Y$.98.44	2	*eth*0
5	$X.Y$.96.18/32	$X.Y$.96.102	3	*eth*0
6	192.168.111.1/24	$X.Y$.96.102	2	*eth*0
7	192.168.111.2/24	$X.Y$.96.102	3	*eth*0
8	$X.Y$.96.0/23	$X.Y$.96.102	2	*eth*0
9	$X.Y$.98.0/23	$X.Y$.96.102	2	*eth*0
10	$X.Y$.96.0/22	0.0.0.0	0	*eth*0
11	0.0.0.0/0	$X.Y$.96.102	2	*eth*0
12	127.0.0.0/8	127.0.0.1	0	*l*0

Table 10.2: GW1's Routing Table.

Entry	*destination*	*next hop*	*metric*	*interface*
1	$X.Y$.96.102/32	0.0.0.0	1	*eth*0
2	$X.Y$.97.151/32	$X.Y$.96.102	2	*eth*0
3	$X.Y$.98.44/32	$X.Y$.96.18	3	*eth*1
4	$X.Y$.98.24/32	$X.Y$.96.18	2	*eth*1
5	$X.Y$.96.18/32	0.0.0.0	1	*eth*1
6	192.168.111.2/32	$X.Y$.96.18	1	*eth*1
7	192.168.111.0/24	0.0.0.0	0	*eth*0
8	$X.Y$.96.0/23	$X.Y$.96.18	1	*eth*1
9	$X.Y$.98.0/23	$X.Y$.96.18	1	*eth*1
10	$X.Y$.96.0/23	0.0.0.0	0	*eth*1
11	$X.Y$.98.0/23	0.0.0.0	0	*eth*1
12	$X.Y$.96.0/22	0.0.0.0	0	*eth*1
13	0.0.0.0/0	$X.Y$.96.1	0	*eth*1
14	0.0.0.0/0	$X.Y$.96.18	1	*eth*1
15	127.0.0.0/8	127.0.0.1	0	*l*0

Chapter 11

Implementation and Experimental Results

To validate the correctness of the proposed mechanisms and to evaluate the system performance, I have deployed a small-scale testbed with two gateways and five mobile nodes. In this testbed I have prototyped the core functionalities of my architecture. In particular, I have developed the software components described in Section 10.2 to support the internetworking with the fixed Internet, while I have left to future work the implementation and testing of the address autoconfiguration scheme described in Section 10.1. To demonstrate that my solution provides higher performance in terms of per-connection throughputs than NAT-based schemes, in my testbed I have also implemented the mechanisms described in [48] to integrate NAT gateways with MANETs running OLSR routing protocol. Comparative tests have been conducted both in static and mobile configurations.

11.1 Testbed description

To carried out my analysis I set up a testbed using seven IBM R-50 laptops with Intel Pro-Wireless 2200 as integrated wireless card. All nodes used a Linux 2.6.12 kernel and run the OLSR Unik implementation in version 0.4.7, which is fully compliant with the RFC specification and also implements additional modules to support explicit tunnelling between ad hoc nodes and gateways. The ad hoc nodes were connected via IEEE 802.11b wireless links, transmitting at the maximum fixed rate of 11 Mbps. To generate asymptotic TCP traffic I used the iperf tool [1], while the saturated UDP traffic was generated with the MGEN tool [2]. Differently from other studies [48, 26], in which the network topology was only emulated by using the IP-tables feature of Linux, my experiments were conducted in realistic scenarios, with hosts located at the

[1]http://dast.nlanr.net/Projects/Iperf/.

[2]http://cs.itd.nrl.navy.mil/work/mgen/.

ground floor of the CNR building in Pisa. Finally, to obtain statistically reliable results I replicated each experiments five times and I measured the 95% confidence intervals.

11.2 Path life characteristics

A preliminary set of experiments was conducted to gather a better understanding of the OLSR behaviour, and the performance trade-offs between protocol overheads and responsiveness to network dynamics. The results of these tests are fundamental to isolate performance limitations depending on routing inefficiencies from the overheads introduced by the interconnection between the MANET and the fixed Internet. For these reasons, in the following I analyse how the OLSR parameters' setting influences the network performance and the path life in particular. The path life, whose formal definition will be introduced later in this section, is a measure of the routing protocol ability to maintain reliable and up-to-date topological information. Ideally, the routing protocol should be able to promptly react to link failures caused by radio link problems and node mobility, such as to eventually find alternative optimal routes. To this end, the OLSR routing protocol defines a set of procedures to monitor the link quality and to identify link and route changes, such as link sensing, neighbour detection and topology discovery. It is intuitive to note that the efficiency of these mechanisms has a significant influence on the route stability, the maintenance of end-to-end connectivity and the prompt reaction to topology modifications. Therefore, initially I investigated more in depth the impact of the routing protocol parameters on the performance of ad hoc networks, such as to identify an "optimal" parameter setting to be used during the tests on the Internet access performance.

First of all, I can notice that the OLSR protocol periodically generates routing control packets in order to refresh the topology information. The behaviours of the various OLSR procedures are therefore regulated by a set of parameters that establish the timing for the OLSR operations. The default constant values for these parameters are defined in the OLSR RFC [37]. More precisely, each node generates, with a period equal to $HELLO\_Interval$, HELLO messages to perform link sensing. The information provided in HELLO messages is considered valid for a $NEIGHB\_HOLD\_TIME$. Furthermore, periodic link reports, the TC messages, are generated by the MPRs. The validity time for TC message information is the TOP_HOLD_TIME, while the repetition period is the $TC\_Interval$. Finally, each gateway, being connected to external networks, generates HNA messages, providing information on the reachability of non-OLSR networks. By analogy with previous parameters, HNA_HOLD_TIME and $HNA\_Interval$ are the validity time and repetition period of HNA messages, respectively. It is intuitive to observe that I may enhance the routing reactivity to topological changes by reducing the maximum time interval between periodic control message transmissions. However, the effect of using different parameter settings from the default ones has not been

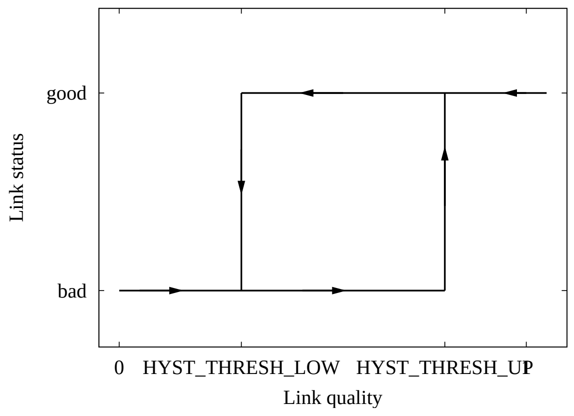


Figure 11.1: Illustration of the hysteresis process.

clearly quantified in the literature.

An additional feature of the OLSR protocol, which may impact the end-to-end connectivity characteristics, is the *link hysteresis* process [37]. The link hysteresis is a procedure designed to make more robust the link sensing against bursty losses or transient connectivity between nodes. More precisely, the OLSR protocol computes for each link between a node and its neighbours a value, called *link_quality*, which measures the reliability of that link. A link is considered "bad" if it allows HELLO messages to pass through it sometimes but not very often. An established link, i.e. a symmetric and reliable link, is considered pending and not usable for communications when its *link_quality* goes below a fixed threshold, known as $HYST\_THRESHOLD\_LOW$. Note that a pending link is not considered broken because the link properties are still updated for each received HELLO message. On the contrary, a link is considered as lost when its validity time expires. In this case, the link is purged from the neighbourhood list. On the other hand, a pending link is promoted to the established status only when its *link_quality* goes above a fixed threshold known as $HYST\_THRESHOLD\_UP$. It is quite obvious that it should be $HYST\_THRESHOLD\_UP \geq HYST\_THRESHOLD\_LOW$. Figure 11.1 illustrates the link hysteresis behaviour. The diagram points out that when $HYST\_THRESHOLD\_LOW <$ *link_quality* $< HYST\_THRESHOLD\_UP$ the link status remains unchanged.

A key implementation requirement for the link hysteresis is the availability of an appropriate measure of the *link_quality*. If some measure of the signal/noise level on a received message is available (e.g., as a link layer notification), then it can be used as an estimation of the *link_quality* index (after being normalized to the range $[0, 1]$). The OLSR specification [37] describes an alternative algorithm to estimate the *link_quality*, which does not require the use of link-layer information. This algorithm monitors the number of lost OLSR messages, Then, the exponentially smoothed moving average of the OLSR-packet transmission success rate is adopted as a measure of the $link\_quality$. Formally, every time an OLSR message is correctly received $link\_quality = (1 - HYST\_SCALING) \cdot link\_quality + HYST\_SCALING$, where the $HYST\_SCALING$ value is the smoothing factor of the estimator, which is a number fixed between 0 and 1. When an OLSR message is lost, the instability rule [37] is applied, that is $link\_quality = (1 - HYST\_SCALING) \cdot link\_quality$. Note that the status of a new discovered link is initially set pending and its $link\_quality$ value is fixed to $HYST\_SCALING$.

The behaviour of the hysteresis strategy is clearly determined by the specific setting of the algorithm parameters, and in particular by the memory size of the $link\_quality$ estimator and the threshold values. The OLSR specification suggests as default configuration $HYST\_THRESHOLD\_LOW = 0.3$ and $HYST\_THRESHOLD\_UP = 0.8$, and it adopts $HYST\_SCALING = 0.5$ as scaling factor. According to these values, even a perfect link (i.e., a link with $link\_quality = 1$) will be purged from the routing tables when two consecutive OLSR control packets are lost. I argue that the standard setting of the hysteresis parameters introduces a critical instability in the routing tables, because it is not infrequent to loose broadcast packets (as the OLSR packets are) when the channel is overloaded.

To validate my intuitions and to investigate the influence of the duration of repetition periods on the path life characteristics I performed a set of experiments considering various OLSR parameter settings. The network layout I used in my tests is depicted in Figure 11.2. It is a chain topology consisting of five wireless links. The distances between the ad hoc nodes are set up in such a way to form a 5-hop chain topology with non-volatile wireless links. Node GW is the gateway node attached to the wired LAN. Although derived in a specific network scenario, I believe that my findings are applicable in generic situations because the chain topology is one of the most critical network scenarios for the OLSR protocol, which has been designed particularly for large and dense networks.

I tested the effect of using four different OLSR configurations, which are listed in Table 11.1. My goal was to validate my intuitions on the over-pessimistic behaviour of the hysteresis process (the $set1$ configuration uses default OLSR parameters as in the std configuration, but the link hysteresis is disabled). In addition, I wanted to quantify the impact of reducing the time interval for periodic control message transmissions, and to evaluate the trade-off between improved protocol reactivity and increased protocol overheads. Thus, in $set2$ and $set3$ configurations all the repetition intervals are twice and four times shorter than the default ones, respectively. Note that the validity times have not been changed from the default values indicated in [37]. As reference scenario I consider a network without data traffic, where the OLSR protocol is running using a default setting (hereafter indicated as std^* configuration). Due to space limits, in the following figures I show the experimental results related to a 3-hop TCP connection activated from node $MN3$ to node GW but similar results where obtained also for different path lengths.

The first performance index I measured during the tests was the OLSR

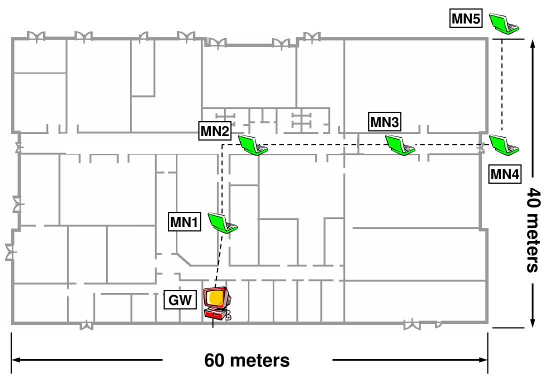


Figure 11.2: Network layout used to conduct tests in static conditions.

Table 11.1: OLSR parameter configurations.

OLSR parameters	*std*	*set1*	*set2*	*set3*
HELLO_INTERVAL(s)	2	2	1	0.5
NEIGHB_HOLD_TIME(s)	6	6	6	6
TC_INTERVAL(s)	5	5	2.5	1.25
TOP_HOLD_TIME(s)	15	15	15	15
HNA_INTERVAL(s)	5	5	2.5	1.25
HNA_HOLD_TIME(s)	15	15	15	15
Hysteresis	yes	no	no	no

overhead, defined as the average amount of OLSR traffic generated per unit time by each node (expressed in terms of bps). As Figure 11.3 illustrates, the routing protocol overheads increase by reducing the generation periods of control traffic. It is evident that by halving the repetition period of OLSR messages I almost double the total routing overhead. However, the total overhead is negligible because it is always lower than 2Kbps. From the shown results, I notice that some nodes (i.e., $MN1$ and $MN2$) generate more control traffic than others. This is due to the fact that in the OLSR protocol only the MPRs generate link state reports, and in my experiments nodes $MN1$ and $MN2$ act as MPRs for the other nodes. Node $MN3$ generates the least OLSR control traffic because is the end-point of the chain and it sends only HELLO messages. Node GW produces more routing traffic than node $MN3$ because it is the gateway node and it sends also HNA messages. On the other hand, it is less intuitive to explain why there is a slight reduction of protocol overheads when activating the TCP connection (*std* configuration) with respect to a network without data traffic (the $std*$ configuration). To provide clear reasons for this phenomenon, I should consider that each HELLO message contains the lists of sending node's single-hop and two-hop neighbours, while each TC message contains the list of all the network links. Hence, the sizes of routing protocol messages vary depending on the number of links that the routing protocol is able to discover and to maintain valid. As I will better explain later, using the *std* configuration instead of the $std*$ increases the number of link timeouts suffered from the OLSR routing (see Figure 11.5). Consequently, in the *std* case the routing messages deliver less topological information than in the $std*$ case. This explains the reduction in the generated overheads.

Figure 11.3 shows the amount of control messages produced by the routing protocol. However, not all the sent messages will be correctly received by nodes' neighbours due to collisions, radio problems, and so on. Therefore, to understand the routing behaviours it is fundamental to evaluate the loss probability. In particular, I measured the loss probability of routing traffic in terms of the percentage of OLSR control messages that a node has sent but that its neighbours have not receive. Figure 11.4 shows the average loss probability for OLSR traffic experienced by each node in the network. From the experimental results I observe that in the presence of heavy-loaded traffic the loss probability can be up to 20%. To explain these values I should note that the OLSR control traffic is encapsulated into UDP packets that are sent as broadcast frames. It is well recognized that the transmission of broadcast packets is unreliable on wireless channel. However, if I consider the $std*$ case the loss probability is very small and always lower that 2%. It is safe to assume that this low number of losses is mainly due to channel noise and the lack of layer-2 retransmissions for broadcast frames. On the other hand, as soon as I introduce data traffic in the network the loss probability of OLSR packets experiences an upsurge. This can be explained by considering the increase of the collision probability in an heavy-loaded network. While the MAC layer retransmit unicast frames to recover from congestion situations, broadcast frames are vulnerable to the collision events. Thus, the presence of data traffic inevitably degrades the per-

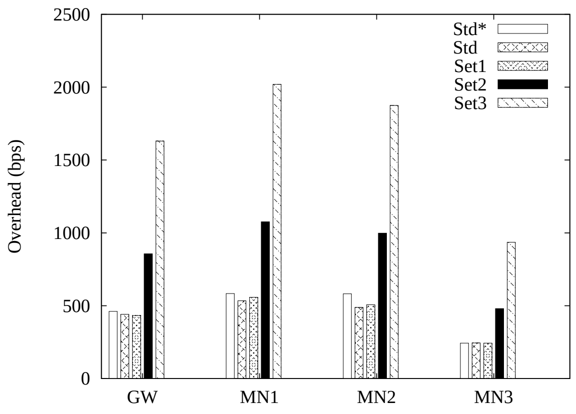


Figure 11.3: Per-node OLSR protocol overheads (TCP case).

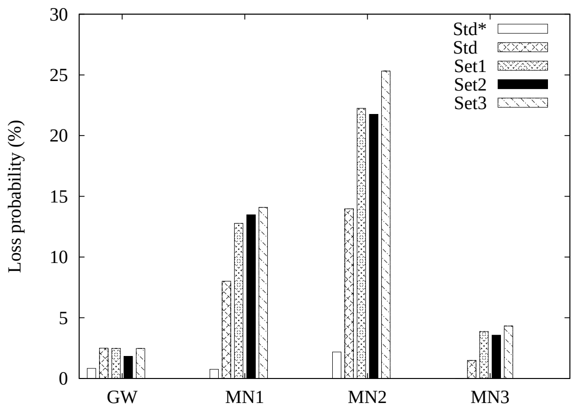


Figure 11.4: Loss probability of OLSR control traffic (TCP case).

formance of the OLSR routing protocol and its ability to efficiently distribute topology updates in case of radio problems or node mobility. Note that the loss probability of OLSR messages measured on node $MN2$ is higher that the loss probability measured on node $MN1$. By inspecting the experiment traces I discovered that this difference was due to the fact that link between node $MN3$ and $MN2$ was less reliable than the link between node $MN2$ and $MN1$. As a consequence the OLSR messages, which are not protected by layer-2 retransmissions, sent by node $MN3$ to node $MN2$ were more frequently subject to channel losses than the ones sent by node $MN2$ to node $MN1$.

To quantify the degradation of the routing protocol performance I introduce the concept of path life. More precisely, this index measures the portion of time during which the source has a valid route to its intended destination. Figure 11.5 show the path life of the route between node $MN3$ and node GW.

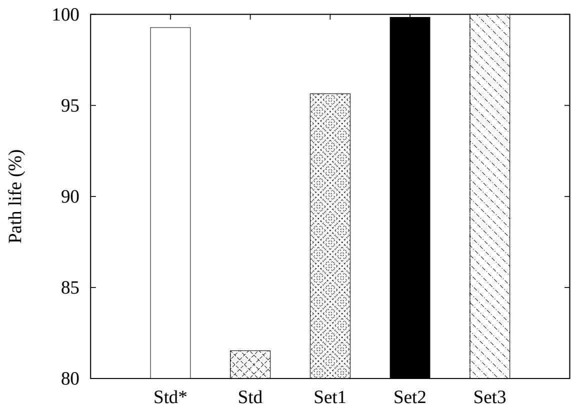


Figure 11.5: Path life of the route between node $MN3$ and node GW (TCP case).

When the OLSR routing is running without the interference of data traffic, the default parameter setting provides a 99% path life. However, the presence of unicast traffic generated by a single persistent TCP connection reduces the routing protocol reliability to 80%. By increasing the frequency the OLSR messages are generated I rapidly counter-balance the negative impact of unicast traffic and channel noise on broadcast routing messages. From the experimental results, I observe that there is no gain in reducing more than four times the repetition intervals because the $set3$ configuration is sufficient to ensure a 100% path life between $MN3$ and GW nodes in static configurations.

I have replicated the same tests substituting the TCP traffic with asymptotic constant-bit-rate (CBR) UDP traffic. My goal was to distinguish between the influence, if any, of flow controlled elastic traffic (TCP) and unresponsive inelastic traffic (UDP) on the OLSR routing performance. From the experimental results I observe similar trends but with a general decrease of routing performance. In particular, Figure 11.6 illustrates the loss probability experienced by each node in the network in the same configurations used during the TCP case. From the experimental results I can notice that the loss probabilities are higher for the central nodes when using UDP traffic instead of TCP traffic, with an increase from a maximum loss probability of 20% to 35%. To explain this increase in the loss probability I can observe that CBR traffic does not regulate its sending rate to limit network congestion. Thus, even when the routing protocol suffers a link failure, the UDP flow continues to generate packets at the same rate. On the contrary, TCP traffic reduces its sending rate when node $MN3$ looses its route to node GW. This is beneficial for the routing control packets, which have more chances to be correctly received.

It is intuitive to note that this further reduction in the percentage of OLSR control packets that are successfully distributed to neighbours, will negatively impact the ability of the routing protocol to maintain a stable and reliable

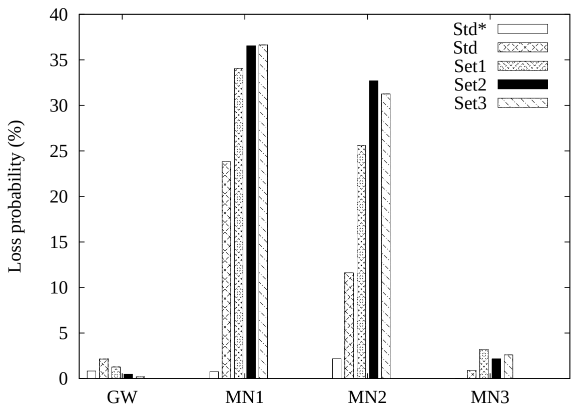


Figure 11.6: Loss probability of OLSR control traffic (UDP case).

end-to-end connectivity. To validate my observations, in Figure 11.7 I reports the path life of the route between node $MN3$ and node GW when an asymptotic CBR UDP flow is delivering UDP traffic from node $MN3$ towards node GW, saturating the wireless links. Concerning the *std* configuration there is a decrease from 80% to 35% of path life changing TCP with UDP, while for the $set1$ configuration the decrease is from 95% to 65%. It is necessary to employ at least the $set2$ configuration to observe reliable routing behaviours.

To summarise my findings, in Figure 11.8 I compare the TCP and UDP throughput for the various parameter settings considered in the experiments. First, we can note that the UDP throughput is always greater than the TCP throughput. This is obviously due to the additional overheads introduced by the TCP return traffic, which consists of TCP ACK packets. It is also straightforward to note that the improvement in the path life stability leads to a correspondent increase in the throughput performance. However, there is not an additional gain by further increasing the repetition frequency of OLSR messages beyond the $set3$ configurations because the throughput curves flatten out. A further observation derived from the shown results relates to the different behaviour of TCP and UDP traffic due to the use of flow control mechanisms in TCP flows. In particular, UDP flows utilize the channel resources in proportion to the path life. For this reason, when the path life is 35% (*std* setting) the UDP throughput is about 35% of the throughput obtained when path life is 100% ($set3$ setting). On the contrary, TCP behaviour is complicated by the use of flow-control that reduces the TCP sending rate when there is a packet loss. As a consequence, when path life is 80% (*std* setting) the TCP connections experience losses and the maximum retransmission timeout increases up to 16 seconds. This indicates that long time intervals separate consecutive retransmissions when the end-to-end connectivity is broken. Hence, even if the path is re-established by the routing protocol the TCP flow may be not aware of this because it has to wait for the retransmission timer expiration before

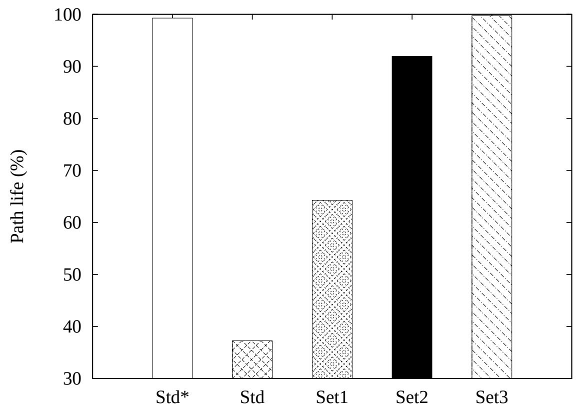


Figure 11.7: Path life of the route between node $MN3$ and node GW (UDP case).

retransmitting the last sent segment. Note that the increase of path life from 80% to 95% ($set1$ setting) reduces the maximum TCP retransmission timeouts to 4.5 seconds. this implies that TCP connections can utilize more efficiently the available path without introducing useless delays between retransmissions. Finally, using the $set3$ parameter setting the route instability is almost null and the maximum TCP retransmission timeout is negligible (less than 0.3 seconds).

If not otherwise stated, in the subsequent sections, in which I compare the efficiency of my proposed solution for interconnecting ad hoc networks to fixed Internet with respect to the NAT-based solution defined in [48], I will configure the OLSR protocol using the $set3$ parameter setting.

11.3 Performance constraints of Internet Access

To measure the performance limits of Internet access I conducted experiments in the network layout depicted in Figure 11.2. However, differently from the tests performed in Section 11.2, the final destination of the data traffic is not the GW node, but a server located in the wired part of the extended LAN. My goal is to verify that my scheme introduces less overhead than a NAT-based scheme using explicit tunnelling between ad hoc nodes and gateways. The network performances are measured in terms of the throughput obtained by TCP and UDP traffic. The differences between the per-connection throughput measured using my proposed scheme and the NAT-based solution specified in [48] are used to quantify the protocol overheads.

The first set of experiments is aimed at evaluating the impact on the per-connection throughput of the number of wireless links traversed in the ad hoc domain before reaching the gateway. In these tests the IP packet size is con-

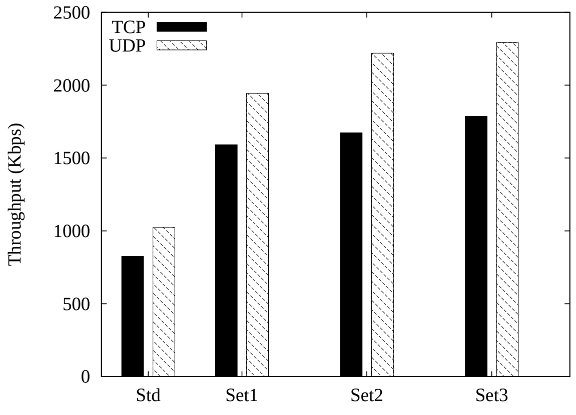


Figure 11.8: Comparison of TCP and UDP throughputs for a 3-hop chain.

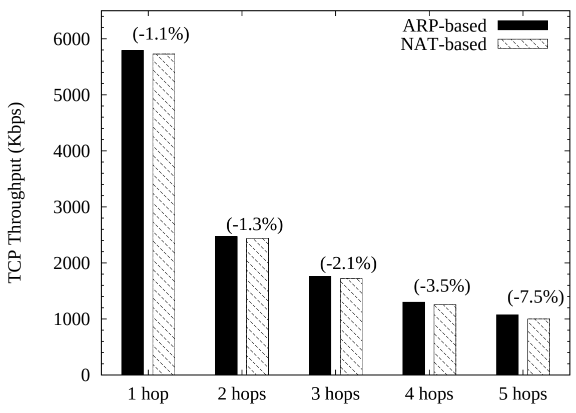


Figure 11.9: Comparison of TCP throughputs versus the number of hops.

stant and equal to 1500 bytes. Figure 11.9 compares the average throughput of a single TCP flow measured using my scheme (indicated with the label “ARP-based” in the plots) or the one proposed in [48] (indicated with the label “NAT-based” in the plots) versus the number of wireless hops needed to reach the gateway. Figure 11.10 reports the results obtained in similar configurations considering UDP flows. In the parenthesis it is shown the percentage difference between the throughput measured in the “NAT-based” case and the one measured in the “ARP-based” case for each network configuration.

Several useful considerations can be drawn from the shown results. First, my scheme guarantees a higher per-connection throughout in all the considered network scenarios. This can be easily explained by noting that in the NAT-based scheme the IP tunnel established between the sender node and the gateway uses IP-in-IP encapsulation. The additional IP header (20 bytes) can

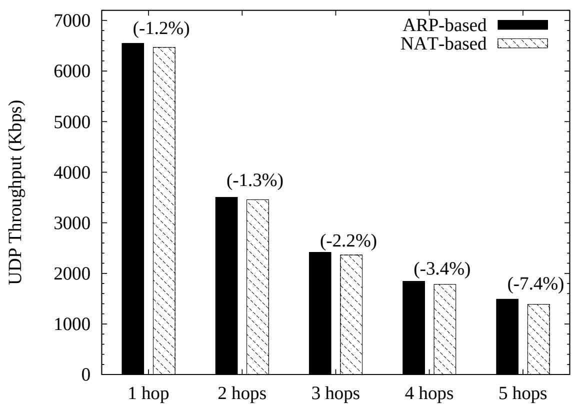


Figure 11.10: Comparison of UDP throughputs versus the number of hops.

appear a small overhead when compared to the overall packet size. In fact, on a one-hop connection the throughput degradations is less than 2 percent. However, this overhead is replicated on all the links traversed by the packets. Hence, the more hops are needed to reach the gateway, the higher is the throughput decrease. In fact, for a five-hop connection the throughput measured in the NAT-based case is about 7 percent less than in the ARP-based scheme. Another interesting observation derived from the shown results is that in both schemes the UDP throughput decrease is almost proportional to the number of hops (i.e., the throughput of a n-hop UDP flow is about n times lower than the throughput of a one-hop UDP flow), while for the TCP flows the throughput reduction is greater. This is due to the self-interference between the TCP data packets and the return TCP acknowledgements, which are small packets reducing the channel capacity. It is worth pointing out that my experiments were conducted at ground floor of the CNR building, where no access points were installed. However, it is still possible that the experimental measures would be affected by external interferences (e.g., access points located at higher floors of CNR building or nearby buildings, employees walking in the corridors, etc.). Hence, to guarantee homogeneous channel conditions between experiments, I interleaved ARP-based tests with NAT-based tests.

In the previous experiments I considered an IP packet size equal to 1500 bytes, and I measures the maximum throughput achievable in each network configuration. However, it is widely recognized that in typical Internet traffic the IP payload size is often smaller than the Ethernet maximum transmission unit [40]. It is intuitive to note that the overheads introduced by the IP-in-IP encapsulation needed to establish a tunnel between the sender node and the gateway degrades the throughput performance when the packet size decreases. To quantify this throughput reduction I conducted a second set of experiments measuring the throughput obtained by a 3-hop TCP and UDP connection versus the IP packet size, and I compared the results obtained with the ARP-based scheme

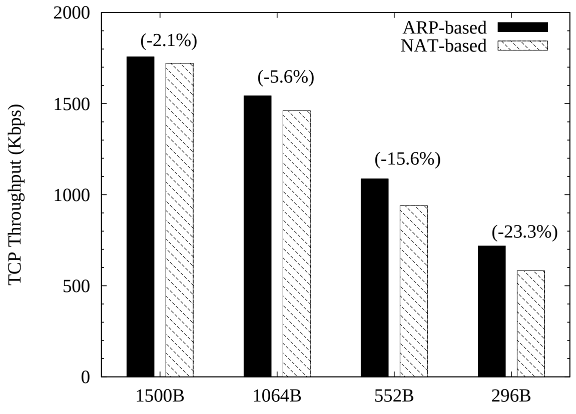


Figure 11.11: Comparison of TCP throughputs versus the IP packet size.

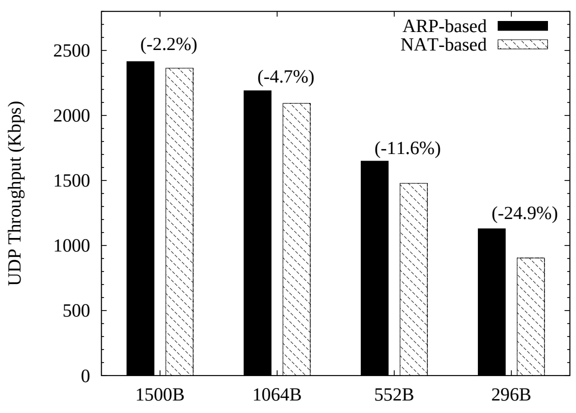


Figure 11.12: Comparison of UDP throughputs versus the IP packet size.

and the NAT-based scheme. Figure 11.11 shows the throughput measured for a 3-hop TCP flow, while Figure 11.12 shows the results in the UDP case. The shown results clearly indicate that the additional IP header added to the original packet is a significant overhead for small IP packets. For instance, when the IP packet size is 296 bytes, the throughput obtained using the NAT-based scheme is about 25 percent lower than the one measured in my scheme.

11.4 Performance constraints with gateway handoffs

To test the mobility support in a multi-homed network configuration I considered the network layout illustrated in Figure 11.13. Note that the considered network layout is a typical example of multi-homed scenarios because in the same ad hoc cloud the mobile nodes can reach two different gateways. A similar network configuration was also considered in [48].

The mobility test begins with the mobile node $MN4$ in position P1, where it is in radio visibility of gateway $GW1$. During the test node $MN4$ has a TCP (or UDP) flow established with a host H in the wired LAN. Using the NAT-based scheme, when node $MN4$ is in position P1 it establishes a tunnel with gateway $GW1$, and this tunnel is permanently maintained throughout the experiment, independently of the $MN4$'s position. On the contrary, using the ARP-based scheme, when node $MN4$ is in position P1 it sets up a default route to $GW1$ to reach the external fixed network, but the default gateway may change according to node mobility pattern. After 50 seconds it moves to position P2. The time needed to move from one position to the next one is always 10 seconds. On location P2, node $MN4$ reaches the gateway $GW1$ through $MN1$, i.e., using a 2-hop wireless path. After other 50 seconds, node $MN4$ moves to position P3. In this location gateway $GW2$ is 2 hops away, while gateway $GW1$ is 3 hops away. Consequently, using my scheme node $MN4$ switches to gateway $GW2$ to forward traffic addressing wired hosts. Moreover, the new default gateway $GW2$ begins to act as proxy ARP for the mobile node. On the contrary, using the NAT-based scheme, node $MN4$ continues to use gateway $GW1$, which is at a distance of 3 hops from node $MN4$. Finally, after other 50 seconds, host $MN4$ moves to position P4, where it is in radio visibility of $GW2$. However, the NAT-based technique forces node $MN4$ to tunnel its traffic towards gateway $GW1$ that is 3-hop away. Then, this movement pattern is repeated on the way back to position P1.

Figure 11.14 shows the throughput obtained by node $MN4$ in case of TCP traffic, while Figure 11.15 shows the throughput obtained by node $MN4$ in case of UDP traffic. The experimental results confirm that both my scheme and the NAT-based solution avoid a permanent TCP (or UDP) session break when the sender moves. However, in my scheme this is achieved by supporting a transparent handoff between the gateways $GW1$ and $GW2$, which does not require node $MN4$'s address reconfiguration. On the contrary, the NAT-based solution described in [48] achieves this result by ensuring that the packets sent

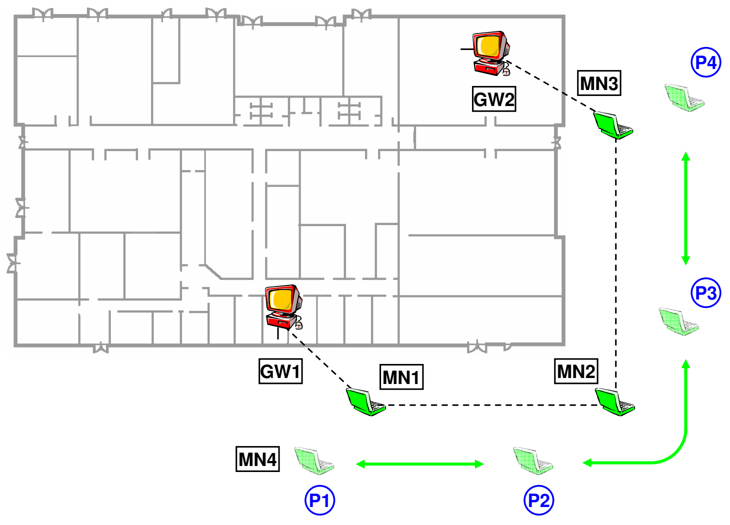


Figure 11.13: Network layout used to conduct tests with node mobility.

by node $MN4$ are always forwarded through gateway $GW1$, independently of the node $MN4$'s position. This is highly inefficient, because in position P3 and P4, node $MN4$ is closer to gateway $GW2$ than to gateway $GW1$ and it could use a shorter route to reach the wired LAN. For instance, in position P4 node $MN4$, using the NAT-based scheme, obtains a throughput that is three times lower than the one measured using my proposed solution.

Another interesting observation that can be derived from the experimental results regards the duration of the temporary connection breaks that may be caused by node movements and gateway handoffs. In particular, Figure 11.14 and Figure 11.15 show that a TCP or UDP connection may be temporarily unable to successfully deliver packets for intervals close to 15 seconds when there is a change in the node $MN4$' s routing table. The analysis of the causes of these throughput holes is quite complex because there is the interplay of different factors. Clearly, a fundamental role is played by the OLSR routing. For instance, let us consider the movement of node $MN4$ from position P1 to position P2. In this case node $MN4$ continues to use gateway $GW1$ to reach the wired LAN, but now the gateway can be reached only through the node $MN1$. The routing protocol may quickly discover that the gateway $GW1$ is now reachable using a 2-hop route, but it will not immediately invalidate the old 1-hop route to $GW1$. In fact, node $MN4$ continues to consider valid and usable the shorter 1-hop route to $GW1$ until the validity time of the direct link to $GW1$ does not expire (the link timeout is 6 seconds by default [37]). After this timer expiration, node $MN4$ has to rebuild its routing table, and this may require a few seconds. On the contrary, when node $MN4$ moves from position

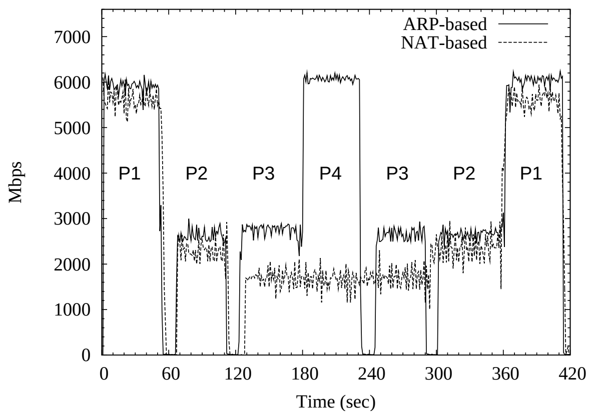


Figure 11.14: Comparison of TCP throughputs when node $MN4$ moves.

P2 to position P1 I do not observe a temporary loss of connectivity. This can be explained by noting that, after this movement node $MN4$ gets closer to gateway $GW1$ (from 2 hops to 1 hop). According to the OLSR routing algorithm, when a shorter route is discovered, the routing table is immediately updated and recomputed. Similar reasoning can be used to explain routing behaviour during node $MN4$'s movements between the other positions. Note that in my scheme the movement from position P2 to position P3 induces the change of default gateway for node $MN4$, while in the NAT-based scheme it causes the use of a 3-hop route instead of a 2-hop route to reach gateway $GW1$.

A second factor that affects the duration of throughput holes is the TCP flow control. In fact, when there is a link breakage, a TCP flow may suffer packet losses, increasing the retransmission timeouts. Hence, the delay introduced between two consecutive TCP retransmissions contributes to increase the interval during which no packets are sent on the channel. In other words, the correct route may be available but the TCP flow does not send packets because it is waiting for the expiration of the retransmission timer. Finally, a third aspect that influences the duration of throughput holes is the queueing delay. In fact, when there is a route change the mobile node has to issue a new ARP_REQUEST to determine the mapping between the IP address and the physical address of the new neighbour. However, in case of asymptotic UDP traffic a large number of packets may be queued in the interface transmission buffer and the transmission of the ARP message may experience significant delays.

11.4.1 Lessons learned from the test-bed

Several lessons can be learned from the experiences gathered during the test-bed implementation and the analysis of experimental results. My first observation refers to the trade-offs related to the use of IP tunnels within multi-hop ad

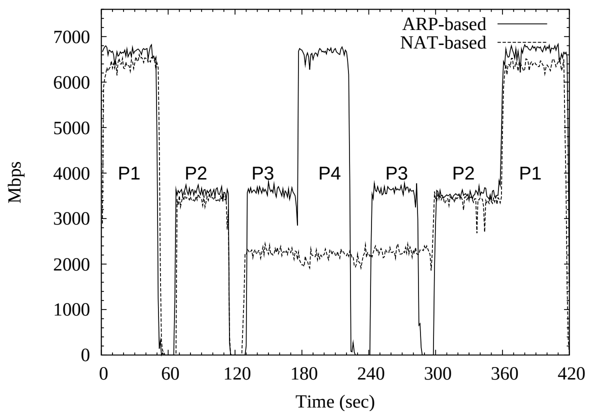


Figure 11.15: Comparison of UDP throughputs when node $MN4$ moves.

hoc environments. This technique is adopted in many existing solutions for enabling interconnection between MANETs and the Internet. In fact, the use of IP tunnels is beneficial to provide transparent support of multi-homing and gateway handoffs. On the other hand, IP tunnelling introduces evident inefficiencies in multi-hop environments because it impedes the use of the shortest paths available to reach the external networks. In multi-hop environments this may lead to significant throughput degradations.

Another relevant practical aspect that should be considered when comparing different solutions for enabling interconnectivity between Internet and multi-homed MANETs is the type of protocol overheads needed to support gateway handoffs. In particular, existing solutions usually require the address reconfiguration when the mobile node changes the default gateway. This address reconfiguration requires time and may contribute to increase the duration of temporary session breaks. Mobile-IP based solutions clearly introduce higher overheads than NAT-based solutions because, upon moving, the mobile node should register the new care-of-address with its home agent that may be far from the MANET. To reduce this heavy burden, the solution proposed in [26] introduces micro-mobility techniques. My scheme does not require address reconfigurations and the gateway handoff causes only an update in the lists of IP addresses masqueraded by the proxy ARP servers running on the gateway.

Finally, the protocol complexity should also be taken into account when evaluating the feasibility of a proposal. For instance, solutions employing overlay networks of underlay layers below IP may be introduce unacceptable implementation complexities. In addition, limiting the modifications to conventional IP mechanisms helps to easily implement the proposed solution under different environments and platforms.

Chapter 12

Conclusion

During my Ph.D. activity, I designed, prototyped and evaluated a practical and lightweight architecture to logically extend traditional wired LANs using multi-hop ad hoc networking technologies. The proposed architecture uses existing ad hoc routing protocols and can transparently interoperate with the fixed Internet infrastructure. I have shown that a simple approach exploiting proxy ARP servers and basic properties of the longest-matching rules used by standard IP routing is sufficient for establishing an heterogeneous network that appears to the external Internet as a single IP subnet. The protocol changes are quite limited and restricted to the gateway nodes. The experiments conducted in a prototyped system demonstrate that the proposed architectural design achieves an efficient use of network resources and provides superior throughput performance than an alternative NAT-based scheme.

I believe that there are several related aspects that are worth being further investigated in future work.

- The gateway selection procedure implicitly relies on the ad hoc routing protocol. In the case of OLSR, it is accomplished using shortest-path criteria. However, in a multi-homing scenario, several gateways can exist, which may be implemented using different technologies and may have different capabilities. Thus, there could be many benefits in designing cooperative heuristics to select gateways such as to obtain load balancing within the ad hoc network, or more efficient handovers.

- In this work I have considered basic IP services, i.e., unicast routing and dynamic address allocation. However, more sophisticated functionalities, such as multicast and QoS management, have been developed for the Internet. Therefore, my proposed architecture should be extended to facilitate the integration of these additional capabilities.

- The address allocation scheme described in this thesis allows the exploitation of DHCP servers to assign IP addresses that are topologically correct in the entire extended LAN. However, it is needed a detailed evaluation of the efficiency of my proposal and a comparative study with other auto-configuration schemes. In addition, I intend to explore how to extend my solution to deal with the typical problems that may arise due to node mobility, such as message losses, and network partitions and merging.

Part III

Transport Protocols in MANETs

Chapter 13

Introduction

Research on efficient transport protocols for ad hoc networks is one of the most active topics in the MANET community. Such a great interest is basically motivated by numerous observations showing that, in general, TCP is not able to efficiently deal with the unstable and very dynamic environment provided by multi-hop ad hoc networks. This is because some assumptions in its design are clearly inspired by the characteristics of wired networks dominant at the time when it was conceived.

To improve TCP performance in a MANETs environment, a number of proposals have been presented. The vast majority of these proposals are TCP modifications that address some particular inefficiency. The main design requirement is indeed to keep the improved transport protocol backward compatible with the legacy TCP, so that "improved" and "legacy" users may be able to communicate with each other. While I acknowledge the importance of TCP compatibility, during my Ph.D. I advocate a different design approach. The TCP inefficiencies over ad hoc networks are so many that a single patch is in general not sufficient to address them all. Thus, TCP would need a large number of modifications to work in a real environment. The practical integration and interoperability between such patches is a critical point, which is still to be addressed. Instead of mixing together patches to fix single flaws, I designed from scratch the *Transport Protocol for Ad hoc networks* (TPA), that is a transport protocol specifically tailored to the MANETs characteristics. *At the design stage* I don't worry too much about backward compatibility with the legacy TCP. I achieve interoperability with TCP at a later stage, as a single and coherent patch to the new protocol.

TPA is a lightweight transport protocol that provides a connection-oriented, reliable type of service. It differs from TCP in a number of ways. Specifically, the data transfer and the congestion control algorithms have been re-designed. Furthermore, TPA explicitly detects and deals with both *route failures* and *route changes.* TPA can leverage *cross-layer* interactions with the routing protocol, when available. For example, it is able to intercept and interpret *route failure* and *route re-establishment* messages. However, TPA works also with routing protocols that do not provide this type of information.

Many papers have pointed out that the drastic differences between MANETs and the legacy Internet may lead to poor performance of TCP over MANETs. However, almost all these studies rely on simulation, and many of them do not consider some important details (e.g., the routing protocol is often omitted). To the best of my knowledge, very few experimental analyses have been carried out so far [57, 71]. On the other side, previous experimental studies have shown that certain aspects of real MANETs are often not effectively captured in simulation tools [14]. Furthermore, available software and hardware products often use parameters settings different from those commonly assumed in simulation tools. Finally, real operating conditions are often different from those modeled in simulation experiments. For example, interferences caused by WiFi hotspots or other devices in the proximity are inevitable in practice. For all the above reasons, I decided to perform a comparison between TCP and TPA performance in a *real* multi-hop ad hoc network. To this end, I implemented a TPA prototype (briefly described in Section 15.9), and set up a multi-hop network testbed to compare TPA with TCP. Firstly, I focus on a string network (throughout referred also as *chain topology*) with variable number of hops (Section 16.6.1). I used this network topology to analyse TCP performance in a real testbed and over different routing protocols. My experimental outcomes are normally aligned with simulation results, and show that TCP performance in multi hop ad hoc networks is sub-optimal and strong depends on the link quality and on the routing protocol parameters. In addition, I also found some results contrasting with simulation. Specifically, I discovered that in a real world the TCP optimal operating point moves with respect to that measured by simulation. I also used this setup to investigate the sensitiveness of TPA to its main parameters and its behaviour over different routing protocols (i.e., OLSR and AODV), and to find its best configurations. The comparison between TPA and TCP over the string topology show that TPA is able to improve TCP performance (in its best configuration) both in terms of increased throughput, and reduced number of (re)transmissions. Specifically, TPA throughput is between 5% and 19% greater than the TCP throughput, and, furthermore, TPA retransmits between 64% and 94% less data segments. This is an important result, as it states that TPA delivers greater throughput, while reducing energy consumption and network congestion *at the same time*. I then consider different topologies (i.e., a cross topology, Section 16.6.2), and mobile scenarios (Section 16.6.3). Also in these cases TPA outperforms TCP with respect to both performance figures.

I also analysed TPA performance over more complex network topologies (Chapter 17). Since it is difficult to construct sophisticated network topology for testbed evaluation, I used, to complete my analysis, the ns-2 simulator [3]. Specifically, using ns-2 I compared TPA and TCP performance over simple network topologies, like the cross and the parallels topologies. The simulative results show that TPA is able to improve TCP performance both in terms of throughput and fairness. Specifically, TPA throughput is between 5% and 6% greather than TCP throughput, and TPA fairness is between 7% and 16%

greater than TCP fairness. I then considered more complex network topologies, like the grid topology and the random topology. Over this topologies, TPA achieves an increment in fairness between 10% and 34%. However, while in the grid topology TPA is able to improve the throughput of about 3.7%, over the random topology TPA experiences a reduction of throughput between 1.5% and 3.6% respect to TCP. Finally, I also considered a scenario where 50 nodes moves over a 1000 x 1000 area. Also in this scenario TPA outperforms TCP. Specifically, TPA achieves an increment in throughput between 1% and 4%, and an increment in fairness between 3.2% and 18.6%.

To conclude TPA analysis, I start investigating how to address, the well-known unfairness problems among concurrent connections (Section 17). I integrated in TPA the Adaptive Pacing mechanism, a popular proposal for improving TCP fairness [46], and I compared the performance of TPA with Adaptive Pacing and TCP with Adaptive Pacing over the previous scenarios. With the adaptive pacing mechanism enabled, TPA is able to increase the TCP throughput up to 39% granting an increment in fairness up to 5.6% respect to TCP. In addiction, TPA is able to reduce the number of retransmitted segment up to 78.7%. Also in this case I used the ns-2 simulator to perform my analysis.

The following thesis Part is organized as follow. Chapter 14 describes the main specificity of MANET that condition TCP behaviour and the major proposals aimed to improve TCP's performance in such environment. Chapter 15 reports the complete description of TPA features, and the description of TPA prototype implementation. Chapter 16 describes the testbed and the experimental methodology used to evaluate TPA and TCP performance in a real testbed, and reports the results obtained from the analysis. Chapter 17 reports the results of a simulative analysis of TPA. It also reports some results about unfairness mitigation in TPA. Finally, Chapter 18 concludes this thesis Part.

Chapter 14

TCP over MANET

14.1 Introduction

TCP (Transmission Control Protocol) is the *de facto* standard for reliable connection-oriented transport protocols, and is normally used over IP (Internet Protocol) to provide end-to-end reliable communications to Internet applications. Although TCP is independent from the underlying network technology, some assumptions in its design are clearly inspired from the characteristics of wired networks dominant at the time when it was conceived. For example, TCP implicitly assumes that nodes are static (i.e., they do not change their position over time), and it also implicitly assumes that packet loss is almost always due to congestion phenomena causing buffer overflows at intermediate routers. Unfortunately, these assumptions do not hold in MANETs, and TCP exhibits poor performance, when it operates in such environments.

In multi-hop ad hoc networks, packet losses due to interference and *link-layer* contentions are largely predominant, while packet losses due to buffer overflows at intermediate nodes are rare events. The TCP protocol reacts to packet losses originated by *link-layer* contentions by activating the window-based congestion-control mechanism. This may lead to throughput degradation and instability [11, 55, 131, 130, 92]. In addition, in multi-hop ad hoc networks nodes may be mobile. This may further degrade the TCP performance [93]. A survey of TCP performance and improvements over Ad hoc Networks can be found in [59].

The rest of this chapter is organized as follows. Section 14.2 describes the main problem encountered by TCP over MANETs. Section 14.3 discusses the major proposal aimed to improve TCP's performance in such environment.

14.2 TCP's challenges

The performances of TCP degrades in MANETs environment. This happen because TCP has been optimized for operate over wired networks. In MANETs, TCP has to face new challenges arising from the specificity of these new networks. In the following I describe the main specificity of MANET that impact TCP behaviour.

14.2.1 Lossy channel

Wireless media are characterized by high, variable bit error rates (BERs). Compared with wired networks, wireless networks are susceptible to loss rates about two orders of magnitude higher. Wireless induced losses are caused primarily by fading, interferences from other equipments, diverse environmental obstructions, and signal attenuation. Any of these factors may induce either single or bursty packet losses. In multi-hop ad hoc networks the effects of channel error are more serious, since a connection may travel over multi-hop wireless links.

In order to increase the transmission success, link layer protocols implement mechanisms to recover from a transmission error. For example, the IEEE 802.11 MAC protocol uses link-layer retransmissions (see Chapter 3) to mitigate the high BER. However, these mechanisms do not entirely solve the problem, and the high BER can still impact TCP performance. For example, in 802.11 based networks, broadcast packets are neither acknowledged nor retransmitted, and hence they are highly vulnerable to collisions and channel errors. Since routing protocol control packets are broadcast packets, the high BER can influence the routing protocol behaviour. This may indirectly degrade TCP performance (see Section 16.5.2).

14.2.2 Interaction between TCP and MAC Protocols

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* and the *exposed* terminal problems (see Section 3) can arise inducing throughput degradation on TCP flows and fairness related issue. A *hidden* node is the one that is within the interfering range of the intended receiver but out of the sensing range of the transmitter. The receiver may not correctly receive the intended packet due to collision from the hidden node. An *exposed* node is the one that is within the sensing range of the transmitter but out of the interfering range of the receiver. Though its transmission does not interfere with the receiver, it could not start transmission because it senses a busy medium, which introduces spatial reuse deficiency. Since multi-hop ad hoc networks are characterized by multi-hop connectivity, these problems heavily affects their behaviour.

[130, 132] shown that the *hidden* and the *exposed* terminal problem may significantly affect TCP performance in a multi-hop ad hoc environment. Specifically, using a string topology network, Xu et al. [130, 132, 131] shown that TCP may encounter the the following problems:

- *instability problem*: the instantaneous throughput of a TCP connection may be very unstable (dropping frequently to zero) even when this is the only active connection in the network.

- *incompatibility problem*: in case of two simultaneous TCP connections, it may happen that the two connections can not coexist: when one connection develops the other one is shut down.

- *one-hop unfairness problem*: with two simultaneous TCP connections, if one connection is single-hop and the other one is multiple-hop, it may happen that the instantaneous throughput of the multiple-hop connection is shut down as soon as the other connection becomes active (even if the multiple-hop connection starts first). There is no chance for the multiple-hop connection once the one-hop connection has started.

The above problems have been revealed in a string network topology like the one shown in Figure 14.1 where the distance between any two neighboring stations is 200 m and stations are static. According to the 802.11 based WaveLan, the nominal transmission radius of each station has been set to 250 m (each station can thus communicate only with its neighboring stations). Furthermore, the sensing and interfering ranges have been set to twice the transmission range, i.e., 500 m, the typical setting of the ns-2 simulator. Below I will provide a brief explanation of how the *instability problem* arises.

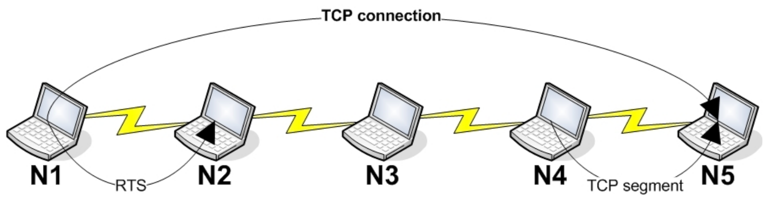


Figure 14.1: Instability Problem.

Figure 14.1 shows one TCP connection, spanning from node *N1* to node *N5* (*four-hop* connection). Let us assume, for example, that node *N4* is transmitting a TCP segment to node *N5*, and node *N1* wants to transmit a frame to node *N2*. *N1* cannot hear the RTS from *N4* because it is out of its transmission range and, thus, it is not aware of *N4* transmission. Consequently, according to the 802.11 MAC protocol, *N1* tries to send an RTS frame to *N2*, and then wait for the corresponding CTS frame. However, *N2* never receives this CTS frame. Most of the RTS transmission attempts tried by *N1* results in a collision at *N2* due to the interference of *N4* (*hidden* station problem). *N2* is in the interfering range of *N4* since in the considered setting the interfering range is twice the transmission range. However, similar situation happens if we consider a smaller interference range (which is more realistic). Let us assume that *N2* is out of the interference range of *N4* . In this case, even if *N2* successfully receives the RTS frame, it is not able to reply with the corresponding CTS frame, again due to *N4*. This because *N2* can sense the transmission of *N4* since *N4* is within the sensing range of *N2*. This inhibits *N2* from accessing the wireless medium (*exposed* station problem). After failing to receive the CTS frame from *N2* for seven times, *N1* reports a link breakage to its upper layer. If a reactive routing protocol is used, like [68, 100], *N1* starts a route discovery process. Obviously, while looking for a new route no data packet can flow along the connection and this makes the instantaneous throughput drop to zero. In sort, the *hidden* and the *exposed* terminal problems may produce a lack of ACKs at the TCP sender, leading it to activate the window-based congestion-control mechanism. As soon as a timeout expires, TCP reduces the window size to one packet even if the network isn't congestionated, reducing the connection throughput.

The TCP congestion window size plays an important role on TCP performance in such environment. [131] and [55] show that there exists an optimal value of the TCP congestion window size at which the link-layer contention is minimized and TCP throughput is maximized. However, TCP congestion control doesn't operate around this optimal value, and typically grows its average window size much larger. This produce an exacerbation of the *hidden* and the *exposed* terminal problems, a decrease in throughput (in the order of 5-30% with respect to the optimal case), and an increase in packet losses. Furthermore, the exponential backoff strategies used by TCP to compute the RTO may exacerbates the situation. When a link failure happens, if multiple timeouts occur, TCP tends to increase the RTO rapidly even there is no congestion. This increase in RTO value may further harm TCP performance.

The above example explains how the *instability* problem arises. Similar explanation can be provided for the *one-hop unfairness* problem. Figure 14.2 shows two TCP connections. The first connection is from node *N1* to node *N2* (*one-hop* connection), while the second connection is from node *N5* to node *N3* (*two-hop* connection). If we assume that *N1* is transmitting a TCP segment to *N2*, node *N4* cannot contact node *N3*, for the same reason explained above. However, there are other factor that affects the performance of the *two-hop* connection. Since *N4* is in the interfering range of *N2*, it has to defer when *N2* is sending. Therefore, *N4* can transmit an RTS frame only when *N2* is not sending. In addition, in the *one-hop* connection as soon as *N1* receives a TCP ACK from *N2*, it can immediately prepares itself to send another TCP segment. This means that *N4* has very few opportunity to find the channel free. Finally, TCP segments transmitted by *N1* are usually much larger in size than RTS frames that *N4* tries to transmit. In conclusion, the time available for *N4* for successfully accessing the channel is very small. In addition, the exponential backoff scheme used by the 802.11 MAC protocol always favours the last succeeding station. This “capture” effect [130, 132] is not peculiar of the string network topology. Gerla and al. observed the same phenomenon even in other scenarios [127].

From the above examples, it emerges that several features of the multi-hop ad hoc environment contribute to the poor performance of TCP. Specifically, the *hidden* and the *exposed* station problems and the random backoff scheme of the 802.11 MAC protocol contribute to a severe unfairness between TCP

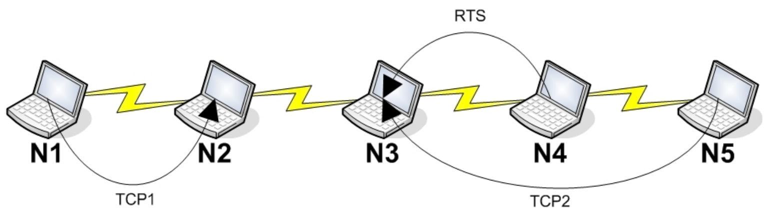


Figure 14.2: One-hop unfairness problem.

connections and also produces poor performance even in the case of a single TCP connection.

14.2.3 Path Asymmetry

Path asymmetry in ad hoc networks may appear in several forms as loss rate asymmetry an route asymmetry. Loss rate asymmetry takes place when the backward and the forward paths suffer a different level loss. This asymmetry occurs because packet losses depend on local constraints that can vary from place to place. This may impact the routing protocol behaviour, and consequently, TCP performace (see Section 16.5). Route asymmetry takes place when TCP data and TCP ACKs follow different paths. This asymmetry may be an artifact of the routing protocol used. Route asymmetry increases routing overheads and packet losses in the case of a high degree of mobility. This because when nodes move, using a different path for the forward and the reverse traffic, increases the probability of route failures experienced by the TCP connectsions.

14.2.4 Node Mobility

Node mobility may severely degrades TCP performance in mobile ad hoc networks (MANETs) [8, 33, 45, 79, 62, 54, 116]. This is due to the inability of the TCP protocol to manage efficiently mobility effects. Node movements may cause *route failures* and *route changes*[1] which may results in packet losses and delayed ACKs at the sender side. TCP misinterprets these events as congestion signals and activates the congestion control mechanism. This may leads to unnecessary retransmissions and throughput degradation. In addition, the process for discovery a new route may take significantly long time, and it depends on the underlying routing protocol, mobility pattern of mobile nodes, and traffic characteristics. If route discovery time is longer than TCP's RTO, multiple timeouts expiration may exacerbates the situation. The situation may be further aggravate if the sender and the receiver of a TCP connection go into different network partitions. This because TCP retransmits unacknowledged segments even if there is no route available to the destination. In such a cases,

[1]Another factor that can lead to route failures are the link failures caused by the congestion on the wireless channel.

multiple consecutive timeouts expiration produce an increment of the RTO value. If the RTO grows to a too high value, since the TCP sender does not have indications on the route re-establishment event, the throughput and the session delay will degrade because of the large idle time.

14.2.5 Routing Protocol Strategies

Routing protocols play a key role on TCP performance. This because in MANETs the network topology may changes frequently an rapidly. When a topology change occurs, the routing protocol is responsible for discovering a new route connecting sender and receiver. In such a situation, the routing protocol must perform the route discovery in a quick fashion to prevent the ongoing TCP connection from being disrupted. Even if the topology is static, the routing protocol strategy may impact TCP performance [91]. In the following are reported some examples on how the routing protocol can affects TCP performance.

In DSR [68] routing protocol, every node keeps a cache of routes it has learned or overheard in order to minimize the transmission overhead involved in route advertisements. The problem with this approach is that in a dynamic environment the probability of stale routes is not negligible. So, unless the stale routes can be detected and recovered in a fast manner, TCP can be led to backoff state, which deteriorates its performance. The problem of stale route in DSR was studied in [61, 62] 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.

As discussed in Section 4.1, AODV may take two different approaches for link failure detection. It can either exploit link failure notifications from the underlying layer, or relay upon a periodic exchange of HELLO messages. Both this approach have their drawback. The link failure notifications mechanism make TCP protocol sensible to loss of packets due to link layer contention [87]. As described in Section 4.1, with this mechanism enabled, after any link breakage the AODV sender must perform route discovery process. Consequently, even if the network is static, loss of packets due link layer contention activates the procedure of route discovery, and TCP throughput may falls to zero for a non negligible period of time. On the other hand, if the HELLO messages are used to maintain local connectivity, the unreliability of broadcast messages may produces false link-failure detections (see Section 16.5.2). These false link-failures degrade TCP performance. With this mechanism enabled, an accurate estimate of the impact of AODV parameters, like *Hello_Interval* and *Allowed_Hello_Loss*, on TCP performance is an ongoing research activity.

Also with OLSR routing protocol there are many parameters that can impact TCP performance. As for AODV, the HELLO_INTERVAL and the NEIGHB_HOLD_TIME parameters play an important role on TCP performance, as noted in Section 11.2. In addition, also the HYSTERESIS mechanism implemented in OLSR Unik [120] significantly affect TCP performance. As for AODV protocol, an accurate study on the impact of OLSR parameters on TCP performance is an ongoing research activity.

Routes in MANETs are short-lived due to frequent link breakages. To reduce delay due to route re-computation, some routing protocols [38, 78, 75, 81, 134, 123] have been designed to maintain and use multiple routes between a sender-receiver pair. With this routing protocol strategy, the probability that there is no path from the sender to the receiver is effectively reduced. However, when multi-path routing protocol are used, TCP encounter many problems. For example, if multiple paths are used simultaneously one problem is that average round trip time (RTT) estimation is not accurate under multipath routing. Namely, the average RTT over several paths may be much shorter than the max RTT (on the longest path). Thus, TCP sender may prematurely timeout packets which happen to take the longest path. Moreover, packets going through different paths may arrive at the destination out of order and trigger duplicate ACKs, which in turn may trigger unnecessary TCP congestion window reductions. [78] shows that using multiple path simultaneously actually degrade TCP performance.

Summarizing, the ability of routing protocol to manage the various characteristics of MANETs, and the strategies used by routing protocols to maintain connectivity between nodes, play an important role on TCP performance.

14.2.6 Power Constraints

MANETs are a collection of mobile nodes. Consequently, batteries carried by each node have limited power supply, and a limited battery lifetime. This is a major issue in ad hoc networks, since each node acts as an end system and as a router at the same time, with the implication that additional energy is required to forward and relay packets. This constraint, imposes that each protocol for MANETs should be designed with special attention to power consumption. As a consequence, also TCP should use the scarce power resource in an "efficient" manner. This means that TCP should be modified to minimize the number of unnecessary retransmissions.

14.3 Related Work

The following section presents various proposals available in literature to improve TCP performance over MANETs. To classify the main related works I used an approach similar to that proposed in [59]. Specifically, I grouped the main proposals to improve TCP performance over Ad hoc Networks into four categories, as follow. The first category, hereafter called *Proposals to Distinguish Between Losses Due to Route Failures and Congestion*, includes TCP enhancements that aim to discriminate between packet loss due to route failures and packet loss due to congestion. The second category, hereafter called *Proposals to Reduce Route Failures*, includes solutions that address the problem of frequent route failures in MANETs. The third category, hereafter called *Proposals to Reduce Wireless Channel Contention*, includes proposals that address the problem of contention on the wireless channel. Finally, the fourth category, hereafter called *Proposals to Improve TCP Fairness*, consists of solutions that address the problem of TCP unfairness in Ad hoc Networks. I also classified the proposals that belong to each category into two set: *cross layer proposals* and *layered proposals*. The cross layer proposals rely on interaction between different layers of the Open System Interconnection (OSI) architecture. Layered proposals rely on modifications of OSI layers that do not use information provided by other layers. The layered solution can be further classified based on the OSI layer at wich they operate. To conclude, I have also dedicated a separate section to that solutions that implement novel transport protocols.

14.3.1 Proposals to Distinguish Between Losses Due to Route Failures and Congestion

Many papers [62, 12, 45, 8, 33, 54] have shown that TCP performance degrade in the presence of node mobility. Node movements may cause route failures and route changes which results in packet losses and delayed ACKs at the sender side. TCP misinterprets these events as a sign of congestion and activates the congestion control mechanism. This may leads to unnecessary retransmissions and throughput degradation. In addition, as noted in [12], factors as MAC failure detection (i.e., the time spent by the MAC to discover a route failure) and route computation latencies (the time taken by the routing protocol to recompute the route after a route failure) also contribute to the degradation of TCP performance. In the following I report the main proposals aimed to discriminate between packet loss due to route failures and packet loss due to congestion.

Cross Layer Proposals

Chandran et al. [33] propose a feedback-based approach (TCP-F) to handle problems related to node mobility. TCP-F uses the Route Failure Notification (RFN), and the Route Re-establishment Notification (RRN) messages to distinguish between losses due to route failure and losses due to network congestion. Specifically, when the routing agent of a node detects the disruption of a route, it explicitly sends a RFN packet to the source. On receiving the RFN, the source goes into a *snooze* state. The TCP sender in snooze state will stop sending packets and will *freeze* all its variables such as timers and congestion window size. The TCP sender remains in this snooze state until it is notified of the restoration of the route through a RRN packet. Upon receiving the RRN, the TCP sender will leave the snooze state and will resume transmission based on the previous sender window and timeout values. TCP-F also uses a timer to avoid blocking scenario in the snooze state. The TCP sender, upon receiving the RFN, triggers a route failure timer. When this timer expires the congestion control algorithm is invoked normally. The authors report an improvement by using TCP-F over TCP.

Holland et al. [62] propose an approach similar to that in [33]. They use an Explicit Link Failure Notification (ELFN) message to freeze the TCP state upon a route failure. This message is is piggybacked onto the route failure message sent by the routing protocol to the sender, and it contains the sender and receiver addresses and ports, as well as the TCP segment's sequence number. On receiving the ELFN message, the source responds by disabling its retransmission timers and enters a "standby" mode. However, unlike the proposal in [33], during the standby period, the TCP-ELFN sender periodically probes the network to check if a route has been re-established. If the acknowledgment of the probe packet is received, the TCP sender leaves the standby mode, resumes its retransmission timers, and continues the normal operations. The authors also study the effect of varying the time interval between probe packets. They also evaluate the impact of the RTO and the Congestion Window (CW) on the restoration of the route. They found that a probe interval of 2 sec. performs the best, and they suggest making this interval a function of the RTT instead of giving it a fixed value. For the RTO and CW values upon route restoration, they found that using the prior values before route failure, performs better than initializing CW to 1 packet and/or RTO to 6 sec. (the default initial value of RTO in TCP Reno and NewReno). This technique provides significant enhancements over standard TCP, but further evaluations are still needed. For instance, the authors only considered the DSR routing protocol, while to a complete analysis other routing protocols should be used (like OLSR [37]). However, Anantharaman et al. [12] report that in the case of high network load , TCP-ELFN performs worse than standard TCP. This because TCP-ELFN uses periodic probe messages to detect route re-establishment. In addition, Monks et al. [83] find that even in the case of light load ELFN performs worse than standard TCP by 5 percent in the case of static ad hoc networks. This because, link failures due to link layer congestion, produce undesired ELFN messages.

Liu et al. [79] propose to leave the TCP implementation unchanged, and insert a thin layer (ATCP) between IP and standard TCP to improve the TCP behaviour. ATCP deals with problems related to high Bit Error Rate, node mobility, and classic network congestion. To discover the network state and to react consequently, ATCP utilizes the network state information provided by ECN (Explicit Congestion Notification) [104] and ICMP "Destination Unreachable" messages. On receiving a "Destination Unreachable" message, ATCP puts the TCP sender into persist mode and itself enters the *disconnected state*. During this state the TCP agent is frozen and no packets are sent until a new route is found by probing the network. A replay to the probing messages remove TCP from persist mode and moves ATCP back into *normal state*. The ECN is used as a mechanism to explicitly notify the sender about network congestion along the route being used. Upon reception of the ECN, ATCP moves into its *congested state* and does nothing. In other words, ATCP does not interfere with TCP's normal congestion behaviour. After TCP transmits a new segment, ATCP returns to its normal state. To detect packet losses due to channel errors, ATCP monitors the received ACKs. When ATCP sees that three duplicate ACKs have been received, it puts TCP in the persist state. After that, ATCP enters the *loss state* and quickly retransmits the lost packet from TCP's buffer. After receiving the next ACK, ATCP returns to its *normal state* and moves TCP to the normal state. Note that ATCP allows interoperability with TCP sources or destinations that do not implement ATCP. ATCP was implemented in a testbed and evaluated under different scenarios, such as congestion, lossy links, partition, and packet reordering. In all cases the transfer time of a given file using ATCP yielded better performance than TCP. However, the used scenarios was somewhat special, since neither wireless links nor ad hoc routing protocols were considered. In fact, the authors used an experimental testbed consisting of five PCs equipped with Ethernet cards. With these PCs, the authors formed a four-hop network. However, certain assumptions, such as an ECN-capable node as well as the sender node being always reachable, might somehow be hard to meet in a mobile ad hoc context. Also, ATCP suffers the same problem encountered by TCP-ELFN, and it does not menage the problem related to link layer contention.

Kim et al. [72] propose TCP-BuS. To menage nodes mobility, TCP bus uses two control messages, like in [33]. These messages are called Explicit Route Disconnection Notification (ERDN) and Explicit Route Successful Notification (ERSN). On receiving the ERDN from the node that detected the route failure, called the Pivoting Node (PN), the source stops sending segments. Similarly, after route re-establishment by the PN using a Localized Query (LQ), the PN will transmit the ERSN to the source. On receiving the ERSN, the source resumes data transmission. During the Route ReConstruction (RRC) phase, packets along the path from the source to the PN are buffered. To avoid timeout events during the RRC phase, the retransmission timer value for buffered packets is doubled. As the retransmission timer value is doubled, the lost packets along the path from the source to the PN are not retransmitted until the adjusted retransmission timer expires. To overcome this, an indication is made to the source so that it can retransmit these lost packets selectively. When the route is restored, the destination notifies the source about the lost packets along the path from the PN to the destination. On receiving this notification, the source simply retransmits these lost packets. However, the packets buffered along the path from the source to the PN may arrive at the destination earlier than the retransmitted packets. In this case the destination will reply by duplicate ACK. These unnecessary request packets for fast retransmission are avoided. In order to guarantee the correctness of TCP-BuS operation, the authors propose to transmit reliably the routing control messages ERDN and ERSN. The reliable transmission is done by overhearing the channel after transmitting the control messages. If a node has sent a control message but did not overhear this message relayed during a timeout, it will conclude that the control message is lost and it will retransmit this message. TCP-Bus introduces many new techniques for the improvement of TCP. The novel contributions of this paper are the buffering techniques and the reliable transmission of control messages. In their evaluation the authors found that TCP-BuS outperforms the standard TCP and TCP-F under different conditions. The evaluation is based only on the ABR routing protocol, and other routing protocols should be taken into account. Also, TCP-BuS did not take into account that the pivoting node may fail to establish a new partial route to the destination. In this case, what will happen to the packets buffered at intermediate nodes is not handled by the authors. In addition, like ATCP, TCP-Bus suffers the same problem encountered by TCP-ELFN, and it does not menage the problem related to link layer contention.

Transport Layer Proposals

Dyer et al. [45] propose a sender-based modification of the standard TCP (fixed-RTO) to distinguish between congestion and route failures. Specifically, they consider the number of in-sequence timeouts expiration as a sign of route failure. After two consecutive timeout expirations, the TCP sender assumes that a route failure has occurred and, hence, keeps the Retransmission TimeOut (RTO) constant until the route is re-established and the retransmitted packet is acknowledged. Dyer et al. evaluate the fixed-RTO proposal by considering different routing protocols and different TCP extentions, as the TCP SACK and the delayed acknowledgment. They report that significant enhancements are achieved when using fixed-RTO with reactive routing protocols. In this proposal, the hypothesis that two consecutive timeouts are only produced by route failures need more analysis, especially in cases of congestion. In addition, the authors do not evaluate the impact of channel contention on their proposal.

Wang et al. [125] propose a modification of TCP called TCP-Door. This proposal does not require the cooperation of intermediate nodes and is based on out-of-order (OOO) delivery events. OOO events are interpreted as an indication of route failure. The detection of OOO events is accomplished either by means of a sender-based and a receiver-based mechanisms. The sender-based mechanism uses the non-decreasing property of the ACKs sequence numbers to detect the OOO ACK packets. To permit the TCP sender to discover an out-of-order delivery, the authors propose to add to the TCP ACK header a one-byte TCP option, called ACK duplication sequence number (ADSN). When TCP receiver sends the first ACK for a data segment, the ADSN option is set to zero. Whenever it sends out a duplicate ACK for the same sequence number, it increments the ADSN number. This way, each duplicate ACK will carry a different ADSN field and the TCP sender can compare this field to detect an OOO delivery. The receiver-based mechanism, instead, needs an additional two-byte TCP option, called the TCP Packet Sequence Number (TPSN), to detect OOO Data packets. Starting from zero, and incremented with every data segment sent (including retransmissions), this TPSN records the exact order of the data segment stream. This is different from the normal TCP sequence number because the latter refers only to the data segment stream a retransmitted segment always carries the old data segment sequence number. With TPSN option, the TCP receiver can detect OOO reliably. If the receiver detects an OOO event, it should notify the sender by setting a specific option bit, called the OOO bit, in the ACK packet header. Once the TCP sender knows about an OOO event, it takes the following two response actions: it temporarily disables congestion control, and instantly recovers during congestion avoidance. In the former action, the TCP sender disables the congestion algorithm for a specific time period (T_1). That is, for a time period T_1 after detecting an OOO, TCP sender will keep its state variables constant, such as the retransmission timer (RTO) and the congestion window size. In the latter action, if the congestion control algorithm was invoked during the past time period (T_2), the TCP sender should recover immediately to the state before the invocation of the congestion control. The authors make the time periods T_1 and T_2 a function of the RTT. Wang et al. analyse TCP-DOOR in different scenarios. Their results show that sender-based and receiver-based mechanisms behave similarly. Thus, they recommend the use of the sender detection mechanism as it does not require notifications from the sender to the receiver. Regarding the two actions mentioned above to be taken upon an OOO event detection, they have found that both actions lead to significant improvement. In general, TCP DOOR improves TCP performance up to 50 percent. Nevertheless, the supposition that OOO events are the exclusive results of route failure deserves much more analysis. Actually, multipath routing protocols such as TORA [94] may produce OOO events that are not related to route failures. In addition, also this work does not address the problem relate to channel contention.

Fu et al. [53] propose an extension of TCP-NewReno called ADTCP. ADTCP uses joint statistics of different end-to-end metrics to distinguish between different network states like congestion, channel error, route change, and disconnection. To identify congestion, the authors use the Inter-packet delay difference (IDD) and the Short-term throughput (STT) metrics. IDD Metric measures the delay difference between consecutive packets, and it reflects the congestion level along the forwarding delivery path. STT metric is also intended for network congestion identification, and it provides observation of throughput over a predefined time interval. When ADTCP is not in the congested state, it uses the Packet out-of-order delivery ratio (POR) and the Packet loss ratio (PLR) metrics to distinguish between channel-error and route-change states. ADTCP takes a burst of high POR sample values as an indication of a route change and a high PLR value as an indication of a high rate of channel error. ADTCP detects a disconnection event when packet delivery is interrupted for non-congestion reasons for long enough to trigger a retransmission timeout at the sender.

14.3.2 Proposals to Reduce Route Failures

Node mobility may produce frequent Route Failures and Route Changes. These events impact the performance of TCP. In the following I report the main proposals aimed to reduce the amount of Route Failure.

Cross Layer Proposals

Koparty et al. [74] propose Split TCP. Split TCP scheme splits long TCP connections into shorter localized segments to reduce the number of route failure in connections that have a large number of hops, and to improve fairness between multiple flows. The interfacing node between two localized segments is called a proxy. The routing agent decides if its node has the role of proxy according to the inter-proxy distance parameter. The proxy intercepts TCP segments, buffers them, and acknowledges their receipt to the source (or previous proxy) by sending a local acknowledgment (LACK). A proxy is also responsible for delivering the segments at an appropriate rate to the next local segment. Upon receipt of a LACK (from the next proxy or from the final destination), a proxy will purge the segments from its buffer. To ensure source-to-destination reliability, an ACK is sent by the destination to the source, similar to what occurs in standard TCP. In fact, this scheme also splits the transport layer functionalities into end-to-end reliability and congestion control. This is done by using two transmission windows at the source, the congestion window and the end-to-end window. The congestion window is a sub-window of the end-to-end window. While the congestion window changes in accordance with the rate of arrival of LACKs from the next proxy, the end-to-end window will change in accordance with the rate of arrival of the end-to-end ACKs from the destination. At each proxy there would be a congestion window that would govern the rate of sending between proxies. Simulation results indicate that an inter-proxy distance of between three and five impact favourably on both throughput and fairness. The authors report that an improvement of up to 30 percent can be achieved in the total throughput by using Split TCP. The drawbacks are large buffers and network overhead. Also, this proposal makes the role of proxy nodes more complex, as for each TCP session they have to control segment delivery to succeeding proxies.

The proposal in [56] tries to reduce the number of route failures and the route reconstruction latency. These targets are achieved by switching to a new route when a link of the current route is expected to fail in the future. This technique is coupled with the on-demand routing protocols AODV and DSR. The mechanism to predict failure is power-based. More specifically, when an intermediate node along a route detects that the signal power of a packet received from its upstream node drops below a given threshold, called the preemptive threshold, this intermediate node will detect a routing failure. On detecting this event, the intermediate node will send a route warning to the source node. Upon receiving a route warning, the source initiates a route discovery in order to find a higher quality path to switch to. The value of the preemptive threshold appears to be critical. Indeed, in the case of a low threshold value, there will not be sufficient time to discover an alternate path before the route fails. Also, in the case of a high threshold value the warning message will be generated too early. To overcome the fluctuations of the received signal power due to channel fading and multipath effects, which may trigger a preemptive route warning and cause unnecessary route request floods, the authors use a repeated short message probing to verify the correctness of the warning message. When the signal power of the packet sent from an adjacent node drops below the preemptive threshold, the node which received the packet with sub-threshold signal strength starts pinging the adjacent node (which transmitted the packet that was received with below-threshold power). Upon receiving the ping, a node immediately responds with a pong. Upon receiving the response, the original node (which received the packet with low power) pings the adjacent node again and receives a pong again. n such ping-pong responses are monitored for signal strength. During this monitoring period if the total number of bad packets received is above a certain threshold value k, then a route warning is sent back to the source. Also if there is no response to a ping within a timeout-period a route warning is sent back. Using simulations the authors show that their scheme yields a reduction of the number of route failures and decreases latency by 30 percent. It should be noted that this scheme is "packet receipt event-driven" and that failures cannot be detected if no packets are transmitted.

Klemm et al. [73] propose a solution similar to that in [56]. However, in [73] each node keeps a record of the received signal strengths of 1-hop neighboring nodes. Using these records, the routing protocol predicts link break events. To alleviate the effects of mobility, Klemm et al. propose two mechanism to be integrated at the MAC layer: the Proactive and the Reactive Link Management (LM). Proactive LM tries to predict link breakage, whereas Reactive LM temporarily keeps a broken link alive with higher transmission power to salvage packets in transit. The authors also provide a modification of AODV that allows the forwarding of packets in transit on a route that is going down while simultaneously initiating a search for a new route. To predict link break due to mobility, each node measures the signal strength (P_r) of each packet received, and observes how P_r changes over time. Klemm et al. also propose a mechanism to reduce link failures. Specifically, they modify the MAC layer in order to double the number of retransmission attempts if there is a high probability that the neighbour is still within transmission range. To determine if a node is still within the transmission range, each node measures the signal strength (P_r) of each packet received, and observes how P_r changes over time. However, this mechanism works only when the level of network congestion is not to high, and the authors said that this method will have to be complemented by other techniques that can estimate the level of congestion. The authors use simulations to show that their scheme significantly improve TCP performance.

He et al. [60] propose a scheme to use a narrow-bandwidth, out-of-band busy tone channel to make reservation for broadcast frame, and to perform link error detection. With the proposed scheme, the sender sends short control messages in the busy tone channel to protect broadcast frames. In addition, the sender sends some short control messages in the busy tone channel to identify a false link failure. The authors use both analytical and simulation analysis to validate their solution. Analytical results show that the proposed scheme can reduce the collision probability of broadcast frames and can alleviate the problem of false link failure. Simulations results show that the proposed scheme can improve TCP throughput by 23% to 150%, depending on user mobility, and effectively enhance both short-term and long-term fairness among coexisting TCP flows in multihop wireless ad hoc networks.

Network Layer Proposals

Lim et al. [78] propose to improve the path availability of TCP connections using multipath routing protocols. The authors found that the original multipath routing deteriorates TCP performance due to the inaccuracy in average RTT measurement and out-of-order packet delivery. Thus, they introduce a new variation of multipath routing, called *backup path routing*. Backup path routing uses only one path at a time but it maintains some backup paths and can switch from current path to another alternative path rapidly if current path fails. Using simulations, the authors observed that maintaining one primary path and one alternate path for each destination yields the best TCP performance. The author use three criteria for the destination node to choose the good paths: the *shortest-hop path*, the *shortest-delay path* and the *maximally disjoint path*. The shortest-hop path means the number of hops from the source to the destination is the smallest. The shortest-delay path refers to the path from which the destination receives the first RREQ packet. The maximally disjoint path is for selecting the secondary path after the primary path has been decided. That is the secondary path is the one which has the fewest overlapped intermediate nodes with the primary path. The authors then consider a combination of these criteria to select the primary and the secondary paths. The first combination consists of selecting the *shortest-hop path* as the primary and the *shortest-delay path* as the alternate. The second combination consists of *selecting the shortest-delay* path as the primary and the *maximally disjoint path* as the alternative. Comparing the two selection schemes, the authors found that the first scheme outperforms the second scheme. As a result of using the second scheme, routes tend to be longer in number of hops. Comparing TCP performance over the DSR routing protocol with backup routing, the authors report an improvement in TCP throughput of up to 30 percent with a reduction in routing overheads. These results are based on various mobility and traffic load scenarios.

Chung et al. [87] propose a simple modification to reactive routing protocols to reduce the number of false link failure. At first, the authors redefine the throughput instability problem as a “re-routing instability problem”, since it is caused by the triggering of the re-routing function and is not specific to TCP traffic alone. Then, they propose to adopt a “don’t-break-before-you-can-make” modification to the existing ad-hoc routing protocols. With this strategy, after a link failure the old route will continue to be used until a new one can be established. At the same time, the source node is informed about the link failure event and starts route discovery procedure. After a new route is created, all nodes discard the previous route and switch to the new one for transmission. The modified routing agent can still switch to a new route successfully in a real-break case. The proposed mechanism has been implemented with AODV. The authors use simulations to shown that their mechanism eliminates the instability problem.

14.3.3 Proposals to Reduce Wireless Channel Contention

Many papers [132, 131, 130, 55, 34, 92, 85] have shown that TCP performs poorly even in static MANETs. In a static environment the maximum achievable throughput is limited by the interaction (at the MAC level) between neighboring nodes [76]. According to the IEEE 802.11 MAC protocol, each node must sense the medium before starting transmissions. In addition, interferences may cause collisions at the destination node. Hence, it can be shown that in a string (or chain) topology, like the one shown in Figure 14.1, the expected maximum bandwidth utilization is only 0.25 [76]. However, the 802.11 MAC protocol is not able to find the optimum schedule of transmissions by itself. In particular, in a chain topology it happens that nodes early in the chain starve later nodes (similar remarks apply to other network topologies as well), as noted in Section 14.2.2. Thus, in practice performance is even worse than expected.

The above limitations are inherent to the characteristics of multi-hop ad hoc networks, and cannot be accounted to the TCP protocol. However, the interaction between TCP mechanisms (mainly the congestion control algorithm) and MAC-layer issues (hidden/exposed node problem, exponential backoff scheme, etc) may lead to several, unexpected, serious instability and fairness problems in some specific scenarios, as shown in Section 14.2.2. The TCP congestion window size is also responsible for suboptimal performance in almost every scenario which may result in throughput degradation and instability [131, 55]. In [55] it has been shown that, for a given network topology and traffic pattern, there exist an optimal value of the TCP congestion window size at which the TCP throughput is maximized. However, TCP does not operate around this optimal value and typically grows its average window size much larger, leading to decreased throughput (throughput degradation is in the order of 5-30% with respect to the optimal case) and increased packet losses. The very reason for this suboptimal behaviour is the origin of packet losses. Unlike traditional wired networks, in MANETs packet losses caused by buffer overflows at intermediate nodes are rare events, while packet losses due to link-layer contention are largely dominant.

In the following I report the main proposals aimed to reduce wireless channel contention.

Transport Layer Proposals

Many papers try to reduce channel contention limiting the TCP congestion window. Xu et al. [131], and Fu et al. [55], propose to explicitly bound the TCP congestion window size to reduce channel contention. They show that explicitly bounding the TCP congestion window size to a small value significantly improves TCP performance over static multi-hop networks. Xu et al. [131] also highlights that the delayed acknowledgment option (RFC 1122) can reduces the contention on the wireless channel and is beneficial to TCP performance. On the other side, [55] presents a methodology, based on considerations about spatial reuse, for calculating the optimal TCP congestion window size for different network topologies and traffic patterns. Both [131] and [55] use a fixed value for the TCP congestion window size. An interesting issue is how to dynamically select the maximum congestion window size to achieve optimal throughput in dynamic scenarios. To this end, Chen et al. [34] bound the problem of properly setting the maximum congestion window size to the bandwidth-delay product (BDP) of multi-hop paths. They prove that regardless of the MAC layer being used, the value of the BDP in bytes at multi-hop routes cannot exceed the value of the round-trip hop-count (RTHC) times the size of data packets in bytes of these multihop routes. This is done by assuming similar bottleneck bandwidths along the forward and reverse route. The authors propose an algorithm to adjust the maximum congestion window size of TCP according to the RTHC of the routes used. Using their algorithm they report an improvement in TCP performance of up to 16 percent even in mobile scenarios that contain multiple TCP.

Papanastasiou et al. [92] and Nahm et al. [85] propose not to limit the congestion window size, and to modify the algorithm used by TCP to increase the congestion window. Papanastasiou et al. [92] propose the Slow Congestion Avoidance Scheme (SCA). SCA increases the sending rate during the congestion avoidance phase more slowly than in the legacy TCP protocol, so as to reduce the number of *on-the-fly* packets. Nahm et al. [85] propose the fractional window increment (FeW) scheme. FeW is similar to SCA and uses a fractional increment of the TCP congestion window to keep the network load at a reasonable level and stabilize the sending window to a relatively small value.

Altman et al. [11] and Oliveira et al. [42] present two mechanisms to reduce the contention over the wireless channel based on a modified version of the standard delayed ACK mechanism (RFC 1122). Altman et al. [11] propose a novel delayed ACK scheme that allows to delay more than two ACKs. In the standard delayed ACK, the TCP receiver uses a fixed coefficient d set to 2 to delay ACKs, where d represents the number of TCP segments that the TCP receiver should receive before it acknowledges these segments. Altman et al. propose to dynamically vary the value of d with the sequence number of the TCP segments. Specifically, the authors define three thresholds, $l1$, $l2$, and $l3$, such that $d = 1$ for packets with sequence number N smaller than $l1$, $d = 2$ for packets with $l1 \leq N \leq l2$, $d = 3$ for $l2 \leq N \leq l3$, and $d = 4$ for $l3 \leq N$. The authors use simulations to study the packet loss rate, throughput, and session delay of TCP New Reno and TCP New Reno with their proposal, in the case of short and persistent TCP sessions on a static multihop chain. They show that their proposal, with $l1 = 2$, $l2 = 5$, and $l3 = 9$, outperforms standard TCP as well as the delayed ACK option for a fixed coefficient $d = 2, 3, 4$. They suggest that better performance could be obtained by making d a function of the sender's congestion window instead of a function of the sequence number.

Oliveira et al. [42] propose TCP-DAA. TCP-DAA implements a dynamic adaptive strategy for minimizing the number of ACK segments in transit in the network, permitting TCP receiver to delaying up to four ACKs as in [11]. To minimize the number of unnecessary retransmissions by timeout, the authors propose to decrease the number of duplicate ACKs for triggering a retransmission by the fast retransmit mechanism from 3 to 2 packets. In addition, they propose to increase fivefold the regular retransmission timeout interval for compensating the maximum of four delayed ACKs. Moreover, Oliveira et al. propose a mechanism to adjust the timeout interval of the delayed ACK based on packet inter-arrival times. Similarly to what the TCP sender does, with this mechanism the receiver uses a low-pass filter to smooth the packet inter-arrival intervals. Upon arrival of a given data segment, the receiver calculates the smoothed packet inter-arrival intervals, and uses this value to set the timeout interval at the receiver. The authors use simulations to study the packet loss rate, throughput, and session delay of TCP NewReno and TCP-DAA. They also compare their proposal with that in [11], and with that of TCP with the standard delayed acknowledgment, and a *cwnd* clamped to 3 segments. The results show that TCP-DAA outperforms all the considered protocols. Is useful to underline that the authors use a *cwnd* limited to 4 segments for TCP-DAA.

Network Layer Proposals

Cordeiro et al. [39] propose a novel algorithm, called COPAS (COntention-based PAth Selection), which incorporates two mechanisms to enhance TCP performance by avoiding capture conditions. COPAS implements two novel routing techniques in order to contention-balance the network. First it uses disjoint forward (for TCP data) and reverse (for TCP ACK) paths to reduce the conflicts between TCP segments travelling in opposite directions. Second, it employs a dynamic contention-balancing technique that continuously monitors network contention and selects routes with minimum contention to avoid capture conditions. Specifically, when network contention on a route exceeds a certain threshold, called the backoff threshold, a new and less contended route is selected to replace the high contended route. Also, any time a route is broken, in addition to initiating a route re-establishment procedure, COPAS redirects TCP segments using the second alternate route. COPAS measures the contention on the wireless channel as a function of the number of times a node has backed off during each time interval. Then, each intermediate nodes continuously piggyback its contention information on packets flowing through the forward and reverse paths. Source and destination nodes can thus monitor the status of the reverse and forward routes respectively. Cordeiro et al. compared COPAS and DSR using simulations. The authors found that COPAS outperforms DSR in term of TCP throughput and routing overheads. However, the use of COPAS, as reported by the authors, is limited to static networks or networks with low mobility because as nodes move faster, using a disjoint forward and reverse route increases the probability of route failures experienced by TCP connections. This may induce more routing overhead and more packet losses.

Link Layer Proposals

Fu et al. [55] propose two link layer techniques to reduce the problem related to channel contention: a Link-RED algorithm to tune the wireless link's drop probability, and an adaptive link-layer pacing scheme to increase the spatial channel reuse. The goal of these mechanism is to let TCP operate in the contention avoidance region. Link Random Early Detection (RED) aims to reduce contention on the wireless channel by monitoring the average number of retransmissions at the link layer. When this number becomes greater than a given threshold, the probability of dropping/marking is computed according to the RED algorithm [52]. Since it marks packets, Link RED can be coupled with ECN to notify the TCP sender about the congestion level. Adaptive pacing lets a node further back-off an additional packet transmission time when necessary, in addition to its current deferment period (i.e. the random backoff, plus one packet transmission time). This extra backoff interval helps in reducing contention drops caused by exposed receivers, and extends the range of the link-layer coordination from one hop to two hops, along the packet forwarding path. Link-RED and adaptive pacing work together as follows. Adaptive pacing is enabled by LRED. When a node finds its average number of retries to be less than a predefined threshold, it calculates its backoff time as usual. When the average number of retries goes beyond this threshold, adaptive pacing is enabled and the backoff period is increased by an interval equal to the transmission time of the previous data packet. This way, a better coordination among nodes is achieved under different network load. The authors use simulations to show that these simple techniques lead to 5% to 30% throughput increase compared with standard TCP. These mechanism also improve the fairness between multiple TCP sessions.

14.3.4 Proposals to Improve TCP Fairness

Many paper have shown that channel contention introduce fairness related issue in addition to a severe throughput degradation. In [130, 117, 127] the authors study how TCP connections share the bandwidth of the channel in Ad hoc Networks, and they report some unfair bandwidth sharing using the actual MAC 802.11 in a mutli-hop environment. [129], instead, shows that also in scenarios where TCP crosses wireless ad hoc and wired networks, the TCP unfairness problem persists. In the following I report the main proposals aimed to improve TCP fairness.

Link Layer Proposals

Yang et al. [133] propose a mechanism to improve fairness among TCP flows crossing wireless ad hoc and wired networks. The authors, to send data packets, adopt the “non work-conserving scheduling” policy for ad hoc networks instead of the “work-conserving scheduling”. This is done as follows. The link layer queue sets a timer whenever it sends a data packet to the MAC. The queue outputs another packet to the MAC only when the timer expires. The duration of the timer is updated according to the queue output rate value. Specifically, the duration of the timer is a sum of three parts $D1$, $D2$, and $D3$. $D1$ represents the queue estimation on how long the channel needs to transmit this packet if no contention occurs. It is equal to the data packet length divided by the bandwidth of the channel. $D2$ is a delay, the value of which is decided by the recent queue output rate. The queue calculates the output rate by counting the number of bytes (C) it outputs in every fixed interval T. Thus the value of $D2$ is updated every T seconds. $D3$ is a random value uniformly distributed between 0 and $D2$. $D3$ is used to randomize the delay added in the queue, to avoid synchronization phenomenon, and to reduce collisions. The heuristic behind their solution is to penalize greedy nodes with a high output rate by increasing their queueing delay D2 and to favour nodes with small output rates. For routing packets, the authors still treat them as high priority packets over data packets. Upon arrival, they will be enqueued before all other data packets. Once the queue knows from MAC that it can transmit a packet, the routing packet at the head of the queue is dequeued immediately regardless of whether there is a timer pending. Unlike data packets, the queue will not set any timer after sending a routing packet and will not count them in calculating C. By means of simulations, the authors report that their scheme greatly improves fairness among TCP connections at the cost of moderate total throughput degradation.

Xu et al. [127] show that RED does not solve TCP’s unfairness in MANETs because the congestion does not happen in a single node, but in an entire area involving multiple nodes. The local packet queue at any single node cannot completely reflect the network congestion state. To solve this problem, xu et al. propose a Neighbourhood RED (NRED) scheme, which extends the original RED scheme to operate on the distributed neighbourhood queue. As RED does, each node keeps estimating the size of its neighbourhood queue. Once the queue size exceeds a certain threshold, a drop probability is computed by using the algorithm from the original RED scheme. Since a neighbourhood queue is the aggregate of local queues at neighboring nodes, this drop probability is then propagated to neighboring nodes for cooperative packet drops. Each neighbour node computes its local drop probability based on its channel bandwidth usage and drops packets accordingly. The overall drop probability will realize the calculated drop probability on the whole neighbourhood queue. Thus, the NRED scheme is basically a distributed RED suitable for ad hoc wireless networks. Using simulations, the authors verify the effectiveness of their proposal and the fairness improvement of TCP.

Other papers try to solve the fairness problem in MANETs. Jiang et al. [67] propose and evaluate the use of a distributed *max-min air-time allocation* algorithm to approximate the proportional fairness objective. Huang et al. [67] propose a distributed algorithms that allow each node of Ad hoc networks to determine its *max-min per-link* fair share in a global ad-hoc network without knowledge of the global topology of the network.

Transport Layer Proposals

In [46] the authors address the problem of segments' burst by introducing a modified version of the *Adaptive Pacing* mechanism available for the Internet, and called it *TCP-AP*. *TCP-AP* spreads the segments transmission according to a transmission rate that is dynamically computed. Moreover it incorporates a mechanism to identify incipient congestion and to adjust consequently the transmission rate. In more detail, *TCP-AP* calculates the segments transmission rate taking into account the spatial reuse constraint of IEEE 802.11 multi-hop network. The authors of [55] showed that, in a chain topology, only nodes 4 hops away from each other can transmit simultaneously. Based on this result, the authors of [46] used the *4-hop propagation delay (FHD)*, i.e. the time needed for a segment produced by node i to reach node $i+4$, to calculate the segment transmission rate. *TCP-AP* sender calculates the *FHD* using the RTT estimation and the number of hops of the connection. As anticipated, to evaluate the sender transmission rate, *TCP-AP* uses also an estimate of the link-layer contention. Specifically, the authors of [46] proposed the *coefficient of variation of recently measured RTT (cov_{RTT})* as a measure of contention. With this mechanism, the transmission rate depends on the *FHD* index, and on the link-layer contention. Thus, the connection rate slows down when the contention increases. The authors used simulation to show that TCP-AP outperform TCP NewReno in terms of fairness and throughput.

14.3.5 Ad hoc Transport Protocols

All the above mentioned solutions rely on TCP modifications that address some specific inefficiency of TCP itself, or rely on routing or link layer improvements aimed at improve TCP performance. However, some authors followed a different approach and designed new special-purpose transport protocols specifically tailored to operate over MANETs. ATP [116] (Ad hoc Transport Protocol) and TPA (the transport protocol proposed in this thesis) belongs to this category.

ATP is a Cross Layer Proposal, since it requires assistance from both network and link layers. ATP is completely antithetic to the legacy TCP as it relies upon *rate-based* transmissions, *network-supported* congestion detection and control, *no retransmission timeout*, *decoupled* congestion control and reliability, etc. Each node in the ATP path, piggybacks its available rate in data segments. This information is collected and consolidated by the ATP receiver, and is then periodically sent back to the ATP sender. The ATP sender uses this information to derive the transmission rate, and perform congestion detection. Another ATP feature is that it doesn't use retransmission timeouts for reliability. Instead, it relies upon selective ACKs to report back to the sender any new hole observed by the receiver in the data stream. This feedback is generated on a periodic basis. A drawback of ATP is that it requires assistance from all intermediate nodes along the connection path to perform its tasks. This may make it unsuitable for those environments where the network layer protocol does not provide such a support.

Opposite to ATP, even if novel in many respects, TPA conserves some TCP characteristics adapting them to the new environment. For example, it still uses a window-based transmission scheme and it preserves the end-to-end semantic of TCP. In addition, TPA works properly also without any assistance from the underlying protocol (e.g., when ELFN messages are not provided by the network layer protocol). A complete description of TPA protocol is provided in Chapter 15, while a complete analysis of TPA is provided in Chapters 16 and 17.

Chapter 15

The TPA Protocol

15.1 Introduction

TPA is a transport protocol specifically designed to operate over MANETs, that provides a reliable, connection-oriented type of service. The main TPA design goals are defined by observing the TCP limitations when used over multi-hop ad hoc networks. The main TPA design goals can be summarized as follows:

- The data transfer policy should be more resilient to late and out-of-order segments, and smoothly work with multi-path routing. In TCP all these circumstances typically trigger timeout or useless re-transmissions at the sender, both reducing the throughput, and wasting energy and bandwidth.
- Several papers have shown that the optimal transmit window size of a TCP connection over multi-hop ad hoc networks is limited to a few segments (e.g., [55]). The flow and congestion control algorithms should take this into consideration, and can thus be greatly simplified with respect to TCP. Furthermore, when congestion is over, the transport protocol should try to exploit available bandwidth more promptly than the TCP additive increase policy does.
- Congestion and route failures are different phenomena in ad hoc networks. However, TCP basically has no notion of route failure, and manages this phenomenon through congestion control. Instead, congestions and route failures should be managed via different algorithms, to accommodate distinct and more refined policies.
- Route changes in ad hoc networks can be frequent events, and new paths can provide significantly different performance in terms of delay, congestion level, etc. The transport protocol should adapt quickly to new paths' features, and discarding statistics about old paths. This is not the case

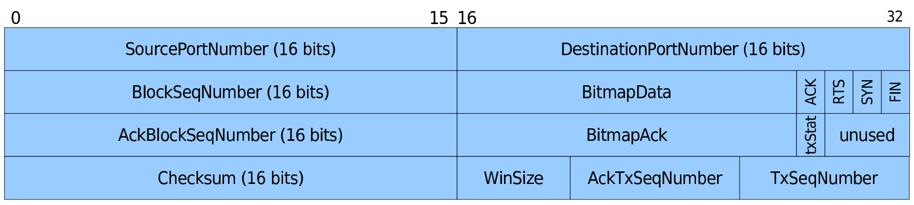


Figure 15.1: TPA header.

in TCP, where, for example, the algorithm used to estimate Round Trip Times and set the Retransmission Timer privileges old statistics with respect to new samples. While this prevents statistics' flapping, it is not the correct way to manage route changes.

- Previous papers have shown that Delayed ACK techniques can be helpful to reduce the network congestion, and ultimately increase the transport-protocol efficiency. I thus include this mechanism in the TPA design.

In the rest of this section I present the TPA aspects that are designed to meet the above goals.

15.2 TPA Segment Structure

The TPA segment consists of an *header field* and a *data field*. The *data field* contains a chunk of application data. The MSS (Maximum Segment Size) limits the maximum size of a segment's data field. The smallest TPA header is composed of 16 bytes. Figure 15.1 shows the structure of the TPA segment. It is usefull to note that TPA is full-duplex. This means that an host A may be receiving data from host B while it sends data to host B (as part of the same TPA connection). Consequently, TPA header contains the fields of both data and ACK segments. The header includes the following fields:

- **SourceportNumber and DestinationPortNumber fields**: These fields are used for multiplexing/demultiplexing data from/to upper layer applications. These two values combined with the source and destination fields in the IP header, uniquely identify each connection.

- **BlockSeqNumber**: TPA collects a number of bytes – corresponding to K TPA segments – from the application, encapsulates these bytes into TPA segments, and then with these segments builds a block of segments. Then, TPA starts to transmit the segments belonging to this block using the mechanism described in Section 15.3. The *BlockSeqNumber* field identifies the block to which the data segment belongs.

- **BitmapData**: This field consists of 12 bits and identifies the position of the data segment within the block (see Section 15.3). For example, if the i^{th} bit of the *BitmapData* is set, the data segment is the i^{th} segment within the block.

- **AckBlockSeqNumber**: The *AckBlockSeqNumber* field identifies the block to wich the acknowledged segment belong. This field is valid only if the ACK flag is set.

- **BitmapAck**: This field consists of 12 bits and describes all the segments belonging to the current transmission block correctly received by the destination. A bit set in the BitmapAck indicates that the corresponding segment within the block *AckBlockSeqNumber* has been correctly received by the destination. The receiver can acknowledge more than one segment by setting the corresponding bits in the BitmapAck. This field is valid only if the ACK flag is set.

- **Flag field**: This field contains 4 bits. The **ACK** bit indicates that the value carried out in the *AckBlockSeqNumber*, *BitmapAck*, and *AckTxSeqNumber* fields are valid. The **RST** bit resets the connection. The **SYN** and **FIN** bits are used for connection setup and teardown (see Section 5.2).

- **WinSize**: This field contains the size of the receiver window, which defines the number of segments the TPA receiver is willing to accept from the sender (see Section 5.4).

- **txStat**: This field is used by the TPA sender to announce its status (*congested* or *not congested*) to the receiver (see Section 15.7).

- **Checksum field**: This field is used for error detection. 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.

- **TxSeqNumber**: This field is used by the sender to identify the data segment sent. Each time TPA send a data segment, it increments the *TxSeqNumber* field by one. When TPA changes the transmission block, it reset the *TxSeqNumber* field.

- **AckTxSeqNumber**: This field is used by the sender to identify the segment that has generated the ACK. For example, if the ACK was generated by a data segment with the *TxSeqNumber* field set to i (15.7), then the receiver sets the *AckTxSeqNumber* field to i. This field is needed since the *BitmapAck* field does not identify the segment that generates the ACK. The *AckTxSeqNumber* field is valid only if the ACK flag is set.

- **unused**: this field is reserved for future use.

15.3 Data Transfer

TPA is based on a sliding-window scheme where the window size varies dynamically according to the flow control and congestion control algorithms. The flow control mechanism is similar to the corresponding TCP mechanism (see Section 5.4) while the congestion control mechanism is described in Section 15.6.

TPA tries to minimize the number of (re)transmissions in order to save energy. To this end, data to be transmitted are managed in blocks, with a block consisting of K segments, whose size is bounded by the Maximum Segment Size (MSS). The source TPA grabs a number of bytes – corresponding to K TPA segments – from the transmit buffer[1], encapsulates these bytes into TPA segments, and transmits them reliably to the destination. Only when all segments belonging to a block have been acknowledged, TPA takes care to manage the next block. Each segment header includes a *sequence number* field that identifies the block to which the segment belongs, and a *data_bitmap* field consisting of K bits to identify the position of the segment within the block. The TPA header also includes two fields for piggybacking ACKs into data segments: *acknowledgement number* and *ack_bitmap*. The *acknowledgement number* identifies the block containing the segment(s) to be acknowledged, while a bit set in the *ack_bitmap* indicates that the corresponding segment within the block has been correctly received by the destination. Of course, it is possible to acknowledge more than one segment by setting the corresponding bits in the bitmap (a single ACK contains information for all the segments within the block).

Segment transmissions are handled as follows. Whenever sending a segment, the source TPA sets a timer and waits for the related ACK from the destination. Upon receiving an ACK for an outstanding segment the source TPA performs the following steps: i) derives the new window size according to the congestion and flow control algorithms (see below); ii) computes how many segments can be sent according to the new window size; and iii) sends next segments in the block (see Figure 15.2a). On the other hand, whenever a timeout related to a segment in the current window expires, the source TPA marks the segment as "timed out" and executes steps i)-iii) as above, just as in the case the segment was acknowledged (see Figure 15.2b).

In other words, for each block TPA first performs a transmission round during which it sends all segments within the block, without retransmitting timed-out segments. Then, the sender performs a second round for retransmitting timed-out segments, which are said to form a "retransmission stream" (see Figure 15.3). In the second round the sender performs steps i)-iii) described above with reference to the retransmission stream instead of the original block. This procedure is repeated until all segments within the original block have been acknowledged by the destination. If an ACK is received for a segment belonging to the retransmission stream, that segment is immediately dropped from the stream.

[1]A block may include less than K segments if the buffer does not contain a sufficient number of bytes.

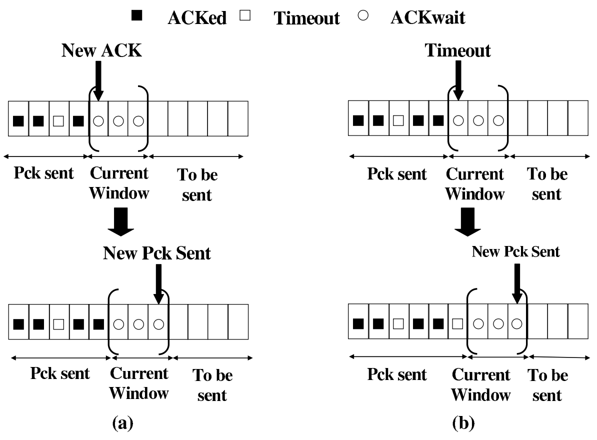


Figure 15.2: ACK reception (a), and timeout expirations (b).

The proposed scheme has several advantages with respect to the retransmission scheme used in TCP. First, the probability of useless transmissions is reduced since segments for which the ACK is not received before the timeout expiration are not retransmitted immediately (as in the TCP protocol) but in the next transmission round. Second, TPA is resilient against ACK losses because a single ACK is sufficient to notify the sender about all missed segments in the current block. Third, the sender does not suffer from out-of-order arrivals of segments. This implies that TPA can operate efficiently also in multi-hop ad hoc networks using multi-path forwarding [95].

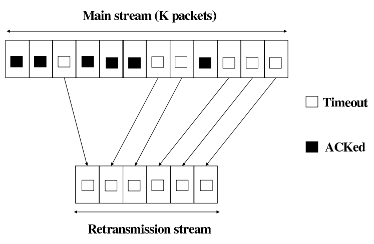


Figure 15.3: Retransmission Stream.

15.4 Route Failure Management

Like many other solutions [33, 62, 79, 114], TPA can exploit, if available, the *Explicit Link Failure Notification* (ELFN) service provided by the network-layer for detecting route failures.

Upon receiving an ELFN, the source TPA enters a *freeze* state where the transmission window size is limited to one segment. In general, I assume that the network layer does not provide route re-establishment notifications. Thus, at the expiration of each retransmission timeout (see Section 15.5), TPA sends segments in the main or retransmission streams (as dictated by the Data Transfer algorithm), probing the network for a new route. To limit the number of segments sent when there is no available route, while in the freeze state, the value of the retransmission timer doubles after each timer expiration [13]. Therefore, TPA realizes that the route has been re-established as soon as it receives an ACK for the latest segment sent. Upon reception of such an ACK, TPA i) leaves the *freeze* state; ii) sets the congestion window to the maximum value $cwnd_{max}$; and iii) starts sending new segments [13]. On the other hand, if route re-establishment messages are available, the TPA behaviour can be further optimized. Specifically, in the freeze state TPA can refrain from transmitting any segment, waiting for a route re-establishment message.

Even if the underlying layer does not provide the ELFN service, the sender TPA is still able to detect route failures as it experiences a number of consecutive timeouts. Specifically, the sender TPA assumes that a route failure has occurred whenever it detects th_{ROUTE} consecutive timeouts. In this case it enters the *freeze* state, and behaves as described above.

15.5 Route Change Management

A transport protocol designed to operate in multi-hop ad hoc networks should react more quickly to route changes than TCP does. To understand why, let us briefly explain how TCP and TPA evaluate the Retransmission Timeout (RTO).

Similarly to TCP, TPA estimates the connection RTT, and uses this estimate to set the Retransmission Timeout (RTO). Both parameters are derived in the same way as in the TCP protocol, i.e.:

$$ERTT_{rtt}(n) = g \times RTT(n) + (1-g) \times ERTT_{rtt}(n-1)$$

$$DEV_{rtt}(n) = h \times |RTT(n) - ERTT_{rtt}(n)| + (1-h) \times DEV_{rtt}(n-1)$$

$$RTO(n) = ERTT_{rtt}(n) + 4 \times DEV_{rtt}(n)$$

where: i) $ERTT_{rtt}(n)$ and $DEV_{rtt}(n)$ are the average value and standard deviation of the RTT estimated at the n^{th} step, respectively; ii) RTT(n) denotes the n^{th} RTT sample; iii) RTO(n) is the retransmission timeout computed at the n^{th} step; and iv) g and h $(0 < g,\ h < 1)$ are real parameters [111].

Multi-hop ad hoc networks are far more dynamic that wired networks, and route changes can be fairly frequent even in static configurations. Whenever a route changes, the new path may differ from the previous one in number of hops. Or, new links in the path may have different properties e.g. in terms of interference. This means that, after a route change, segments may experience a significant variation in the RTT and the re-transmission timeout might be no longer appropriate for the new path. However, the standard TCP essentially uses a low-pass filter to estimate RTT, and thus RTO converges to a value appropriate to the new path after long time. To avoid useless re-transmissions, the TPA protocol must detect route changes as soon as they occur, and modify the RTT estimation method to achieve quickly a reliable estimate for the new RTT. In practice, TPA detects that a route change has occurred either i) when a new route becomes available after a route failure; or ii) when th_{RC} consecutive samples of the RTT are found to be external to the interval $[ERTT_{rtt} - DEV_{rtt}, ERTT_{rtt} + DEV_{rtt}]$ (th_{RC} consecutive late segments or th_{RC} consecutive early segments). Upon detecting a route change, TPA replaces the g and h values in the ERTT and DEV estimators with greater values ($g1$ and $h1$), so that the new RTT estimates are heavily influenced by the new RTT samples. This allows to achieve a reliable estimate of the new RTT immediately after the route change has been detected. Finally, after n_{RC} updates of the estimated RTT, the parameter values are restored to the normal g and h values.

15.6 Congestion Control Mechanism

Also congestion phenomena are quite different in multi-hop ad hoc networks with respect to wired networks. The work in [55] shows that congestions in multi-hop ad hoc networks mainly occur because of link-layer contention and not because of buffer overflow at intermediate nodes, as in the case of wired networks. Congestions due to link-layer contentions manifest themselves at the transport layer in two different ways. An intermediate node may fail in relaying data segments to its neighboring nodes and, thus, it sends an ELFN back to the sender node (provided that this service is supported by the network layer). This case, throughout referred to as *data inhibition*, cannot be distinguished by the sender TPA from a real route failure. On the other hand, an intermediate node may fail in relaying ACK segments. In this case, throughout referred to as *ACK inhibition*, the ELFN (if available) is received by the destination node (i.e., the node that sent the ACK), while the source node (i.e., the node sending data segments) only experiences one or more (consecutive) timeouts. Whenever the sender TPA detects th_{CONG} (with $th_{CONG} >= 1$) consecutive timeout expirations, it assumes that an *ACK inhibition* has occurred, and enters the *congested* state. The source TPA leaves the congested state as soon as it receives th_{ACK} consecutive ACKs from the destination.

If the network layer does not support the ELFN service, the only way to detect both data and ACK inhibitions is by monitoring timeouts at the sender. Congestions and route failures are no longer distinguishable. Hence, th_{CONG} and th_{ROUTE} collapse in the same parameter, and the *freeze* and the *congested* states collapse in the same state.

Many papers have shown that for TCP connections spanning a number of hops not to large, 2 and 3 are the optimal values for the maximum *congestion window size* (cwnd). Specifically, [55] shows that in a h-hop chain topology network, the TCP's optimal throughput is achieved when TCP uses a *cwnd* bounded to $h/4$ segments. Consequently, only for connections spanning a number of hops greather than 13 the optimal *cwnd* is higher than 3. [34] instead, shows that for TCP connections spanning a number of hops up to 12, a *cwnd* of 2 is the optimum, while 3 is the optimal *cwnd* value for connections up to 20 hops. Consequently, I decided to implement in TPA a congestion control mechanism that is window-based as in TCP, but with a maximum congestion window size ($cwnd_{max}$) in the order of 2-3 TPA segments. As a consequence, in TPA the maximum and minimum values of the *cwnd* size are very close, and the TPA congestion control algorithm is very simple. In normal operating conditions, i.e., when TPA is not in the *congested* state, the congestion window is set to the maximum value, $cwnd_{max}$. When TPA enters the *congested* state, the congestion window is reduced to 1 to allow congestion to disappear.

15.7 ACK Management

Many papers [11, 42, 131], have shown that the *Delayed ACK* mechanism can significantly improve the TCP performance. Based on these results I implemented the *Delayed ACK* mechanism in TPA, as well. The concept of delayed acknowledgments recommended in RFC 1122 is that the TCP receiver should send ACK segments for *every other* DATA segments received. I implemented in TPA a delayed ACK mechanism similar to that proposed in RFC 1122. When the TPA sender is not in *congested* state, the TPA receiver sends back one acknowledgement *every other* segment received, or upon timer expiration. Otherwise, if TPA sender is in *congested* state, the TPA receiver sends back one ACK for *each segment* received. This latter feature permits TPA receiver to does not introduce any additional delay when the sender uses a *cwnd* size set to one segment. The TPA sender uses the *txStatus* flag of the TPA segment header to announce its status (*congested* or *not congested*) to the receiver.

I also implemented a modified version of *Delayed ACKs* presented above. Specifically, to minimize the number of ACKs in transit in the network, I modified the receiver to delay up to $cwnd_{max}$ segments. Specifically, when the TPA sender is not in *congested* state, the TPA receiver sends back one acknowledgement *every* $cwnd_{max}$ DATA segments received (i.e. every three segments received if $cwnd_{max}$ is set to three), or upon timer expiration. Otherwise, if the TPA sender is in *congested* state, the TPA receiver sends back one ACK for *each* segment received. Delaying the ACK more than this makes no sense, as the sender would not be able to transmit anything more until a timeout expires at the receiver (and an ACK is thus forcibly sent). Throughout this thesis I will refer to TPA with modified version of the *Delayed ACK* technique as *TPA**.

In both TPA variants, the interval that triggers the ACK transmission is set to a constant value (typically, 100 ms).

15.8 TPA/TCP interoperability

Despite differing from TCP in a number of features, TPA could be adapted to work in a TCP environment, as well. It was not the main point of my Ph.D. activity to investigate this issue, since I focus on the performance improvements granted by the new TPA mechanisms. However, I want here to briefly provide an idea on how TPA could be extended with small modifications to be interoperable with TCP.

The main feature I consider here is how to allow a TPA sender to exchange data with a TCP receiver. As far as the information exchanged between the endpoints, it should be noted that the main difference between TCP and TPA is the way sequence numbers are encoded. Specifically, TPA uses a 2-dimensional scheme (i.e., block number, data/ack bitmap), while TCP uses a 1-dimensional scheme (i.e., sequence number). To make these schemes interoperable, it would be sufficient to add an optional mapping function at the sender side, which provides bidirectional mappings between 2-dimensional and 1-dimensional schemes. Once such a mapping is in place, a TPA sender could work unmodified also with TCP receivers, because all TPA functions described in Section 15.3 through 15.6 are implemented at the sender side, and, as far as interactions with the receiving endpoint are concerned, are based on the ACKs received from the receiver, and on the RTT estimates. Therefore once the TPA sender is able to interpret the ACKs sent by the TCP receiver (which is guaranteed by the mapping function), it is also able to implement all the proposed features. Clearly, the performance of a TPA/TCP connection would be suboptimal, because the TCP receiver will just send cumulative ACKs instead of ACK bitmaps describing the status of the whole block under transmission. However, the data transfer will be possible also in this case. Probably, the SACK option of TCP can be beneficial in TPA/TCP connections.

A straightforward alternative exploits the fact that all major operating systems come with a complete TCP implementation. Therefore the choice between TCP and TPA could be done while opening a connection. Specifically, the `tpa_connect()` function can generate a TCP SYN segment announcing TPA availability (e.g., by using a reserved bit in the TCP header). If the other endpoint does not implement TPA, the sender will revert to a standard TCP connection.

15.9 TPA Protocol Implementation

As anticipated, to test TPA behaviour, during my phd I compared TPA and TCP performance in a real as well as in a simulation environment. To this end, I implemented a TPA prototype in a GNU/linux environment and then I ported this code to the ns2 simulation environment. In the following I will introduce the real word implementation of TPA.

Network protocols are usually implemented in the kernel space and can be accessed by network application through an interface consisting of a set of system calls (e.g., the socket-based interface). Security and performance are the main motivations behind this approach. I refer to such an organization as monolithic [118] because all protocols stacks supported by the system are implemented within a single address space. However, there are several factors that motivate a non monolithic organization, i.e., implementing network protocols out of the kernel space. The most obvious of these factors are ease of prototyping, debugging and maintenance. Another factor may be an improved system stability. When developing protocols in a user-level environment, an unstable stack affects only the application using it and does not cause a system crash. Therefore, I decided to prototype TPA using a user-level implementation. Figure 15.4 shows the TPA location in the network protocol stack. Since TPA only requires a datagram service it is implemented on top of the UDP/IP protocols that are accessed through the socket-based system calls.

When using a traditional transport protocol implemented in the kernel space, a user application can access protocol services through the socket interface. Therefore, messages generated by the application are passed down from the user space to the kernel space. The segmentation process of application messages and the processing of each single segment are performed by the kernel. On the other hand, when using the TPA protocol, messages generated by the application are accumulated by TPA to form data blocks. Blocks are then segmented and each single segment is sent through the network. In my TPA prototype, the segmentation process and the processing of each single segment according to TPA protocol specifications, are obviously performed by the user-level library implementing TPA. Therefore, in my implementation, each single segment (not the entire message generated by the application) is passed down from the user space to the kernel space. This approach may introduce an additional overhead in processing data. However, I think that in multi-hop ad hoc networks this increment in overhead should have a little impact on TPA performance. This because the rate at which nodes communicate is lower than that of traditional wired network. It is also useful to highlight that the aim of my prototype was the performance evaluation of TPA and the performance comparison between TPA and TCP. To this end I used in my experimental analysis laptops running only the essential services. In other words, laptops only ran the routing protocol and the application generating network traffic. This to reduces the overhead encountered by TPA.

Even if my prototype was implemented only to evaluate the performance of TPA, in the design of its software architecture I spent time to allow a transparent and easy integration of legacy TCP applications with TPA. To this end I equipped the TPA implementation with an Application Programming Interface (API) similar to that provided by GNU/Linux operating system for TCP. This API can be used by software developers for implementing network applications based on TPA. The TPA protocol and the relative API has been implemented by using the C programming language. This allows the following benefits:

1. it permits to manage bits very quickly and efficiently
2. it is fully compatible with all Unix/Linux systems

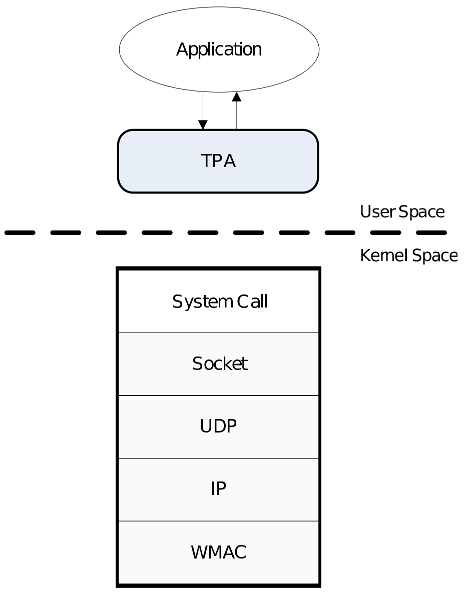


Figure 15.4: Protocol stack.

It is usefull to underline that the current TPA implementation doesn't contain in the segment header neither the source or destination port fields, since we demanded UDP to multiplex communications. In addiction, it doesn't implement the checksum mechanism. Consequently, the size of TPA header is 12 byte, and TCP and TPA segments can carry the same amount of applycation data.

In the following subsections I describe the API and the software organization of TPA.

15.9.1 Application Programming Interface

As anticipated, the main design target of the TPA API was a transparent and easy integration of legacy TCP applications with the TPA protocol. To hit this target I followed the following design principles:

- Transparent integration with applications. The original semantics of applications must be preserved without changing their source code.
- Socket API exportation. The socket application programming interface (API) provided by TCP protocol must be preserved. This allows the re-use of a legacy application on top of TPA with only minor changes. This requirement is a consequence of the previous one.

Based on this principles I implemented for TPA an API similar to that provided in a GNU/Linux system for TCP. Specifically, it consists of a set of functions, each one of which has a correspondent TCP socket system call. Table 15.1 shows the list of functions provided by the TPA API, whereas Table 15.2 reports the meaning of each function. Using this API, a legacy TCP application can be re-used on top of TPA by simply introducing very minor modifications. Specifically, it's enough to introduce the *tpa_* prefix at each TCP socket system call to adapt a TCP application to operate with TPA. To better clarify this issue, Table 15.3 shows the changes that must be introduced in a client program when passing from TCP to TPA. It is usefully to observe that, for the sake of simplicity, not all the functionality provided by the TCP socket functions are supported by the TPA API. For example, the GNU/Linux accept() function extracts the first connection request on the queue of pending connections, creates a new connected socket, and returns a file descriptor referring to that socket. Then, the listening socket (i.e. the socket passed to accept() as argument) can be used again to accept further connections requests. The tpa_accept() function instead, doesn't make difference between the listening socket and the connected socket. Therefore, with the current version of TPA prototype is not possible to implement a concurrent server. Another difference resides in the tpa_send() function. Specifically, it accepts only messages of size multiple of a block size. I will introduce all the missing capability in a future release of TPA prototype. However, as anticipated, the main target of my prototype was the performance evaluation of TPA protocol and my prototype was not implemented to support all TCP applications.

Table 15.1: Functions provided by the TPA library.

int tpa_socket();
int tpa_connect(**int** sockfd, **struct sockaddr_in** *serv_addr, **socklen_t** addrlen);
int tpa_bind(**int** sockfd, **struct sockaddr_in** *my_addr, **socklen_t** addrlen);
int tpa_listen(**int** sockfd, **int** backlog);
int tpa_accept(**int** sockfd, **struct sockaddr** *addr, **socklen_t** *addrlen);
int tpa_send(**int** sockfd, **const void*** packetbuf, **int** packetlen, **int** flags);
int tpa_recv(**int** sockfd, **void** *buff, **int** len, **int** flags);
void tpa_close(**int** sockfd);

15.9.2 Software organization

This section provides a description of the TPA prototype implementation in a GNU/Linux system, i.e., which are the entities that implement the TPA prototype and how this entities can communicate with each other. It is proper to underline that my prototype implements only a subset of TPA capabilities. Specifically, it doesn't support the capability of piggybacking ACKs into data segments. In other words, data belonging to a TPA connection can travel only

Table 15.2: Meaning of functions provided by the TPA library.

Function name	Meaning
tpa_socket()	Creates a TPA socket
tpa_connect()	Connects the socket to the specified address
tpa_bind()	Gives to the socket the local address
tpa_listen()	Specifies a willingness to accept incoming connections and a queue limit for incoming connections
tpa_accept()	Listen for TPA connections requests
tpa_close()	Closes the TPA connection
tpa_send()	Sends data over a TPA connection
tpa_recv()	Receives data over TPA connection

Table 15.3: A simple example showing changes to be introduced in a legacy application to use the TPA protocol.

Standard Library	TPA Library
/* ..... */	/* ..... */
s = socket(...);	s = tpa_socket(...);
connect(s, ...);	tpa_connect(s, ...);
send(s, ...);	tpa_send(s, ...);
recv(s, ...);	tpa_recv(s, ...);
/* ..... */	/* ..... */
close(s);	tpa_close(s);

in one direction. In the following I will refer to *sender side* as the side of TPA connection where data segments are generated and to *receiver side* as the side of TPA connection where ACKs are generated.

Process communication

TPA was implemented with distinct execution flows that interact according to the client/server and producer/consumer models. Specifically, I structured the TPA software module by means of three processes: a *data-processing* process, a *sender* process, and a *receiver* process (see Figure 15.5). The *data-processing* process collects data passed by the *application* process in a buffer to form blocks. Data blocks are then passed to the *sender* process that manages it according to the TPA specification. Finally the *receiver* process is in charge of processing data coming from the network and sending ACKs back to the sender. In this model, processes resident on the same machine communicate each other by using the FIFOs [112] and the signal [113] mechanisms, while the *sender* and the *receiver* process use a UDP socket to transmit data and ACK segments. In the following I will describe how the TPA processes communicate with each other.

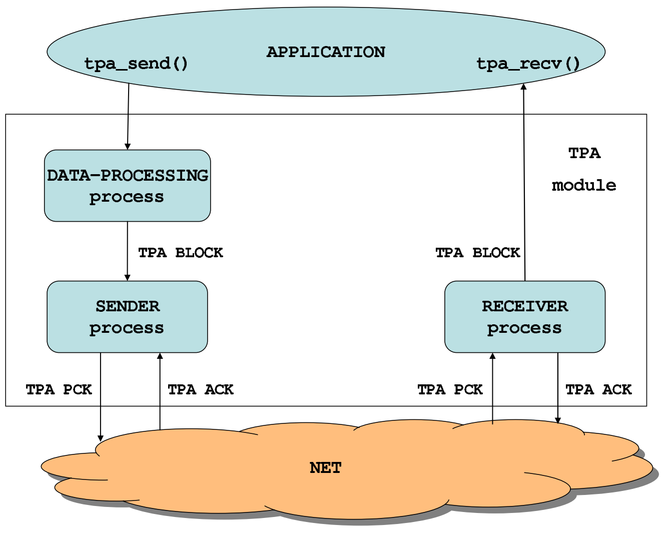


Figure 15.5: Inter-Process Communications.

As previously mentioned, the *data-processing* and the *sender* processes implement the sender side of a TPA connection. They are created by the `tpa_connect()` function upon successful completion of the free-way handshake used to set-up a TPA connection. The `tpa_connect()` function also creates the communication FIFOs used by the sender side processes to communicate. The application working on top of TPA must uses the tpa_send() function (see Figure 15.5) to pass data to the *data-processing* process. This function first reads the *data_to_app* FIFO to checks if the *data_processing* process is ready to accept data. Then, if it is, tpa_send() passes the application data to the process *data-processing* using the *app_to_data* FIFO. As soon as tpa_send() terminates the application data to send, it closes the *app_to_data* FIFO and pauses until the reception of the *SIGUSR1* user-defined signal. This signal is used by the *data-processing* process to inform the application process that all application data has been successfully transmitted to the destination. On reception of this signal, the tpa_send() function returns with the number of bytes successfully sent.

The *data_processing* process collects the application data in a buffer to form TPA blocks. It receives the application data through the *app_to_data* FIFO. As soon as a block is full, *data_processing* checks the state of the *sender* process. If it has declared its availability to accept newer block, *data_processing* passes the block to the *sender* process using the *data_to_send* FIFO and returns to gather application data. Otherwise, it pauses until the reception of the *SIGUSR1* signal. This signal informs *data_processing* about the willingness of the *sender* process to accept another block of data. On reception of this signal, *data_processing* passes the block to the *sender* process and restarts to read the *app_to_data* FIFO for further application data. If *data_processing* finds the *app_to_data* FIFO closed, it pauses until the reception of the *SIGUSR1* signal. On reception of it, *data_processing* sends the *SIGUSR1* signal to the application, informing it that all blocks have been successfully transmitted to the destination. The *data_to_app* FIFO is used by *data_processing* to inform the application about its willingness to accept data.

The *sender* process implements the core functionality of TPA protocol. Specifically, it receives a TPA blocks through the *data_to_send* FIFO and manages it according to the TPA protocol specification. As mentioned early, the *sender* process uses a UDP socket to send and receive TPA segments. This because it only requires a datagram service. As soon as the *sender* process terminates to send all packets belonging to the current block, it sends the *SIGUSR1* signal to the *data-processing* process. This signal informs the *data-processing* process that the *sender* process can receive another block of segments. The *sender* process also monitors the *app_to_send* FIFO. This FIFO is used by the tpa_close() function to inform the sender process that the application want to close the TPA connection. On reception of a special message through the *app_to_send* FIFO, the *sender* process starts the procedure to close the TPA connection. I can observe that the *sender* process must monitor three file descriptors (the UDP socket and the *app_to_send* and *data_to_send* FIFOs). To perform this task, the it uses the poll() function. This function allows a program to monitor multiple file descriptors, waiting until one or more of this become ready3 (e.g., input possible).

The *receiver* process implements the receiver side of a TPA connection. More specifically, it receives TPA data segments and generates ACKs according to the TPA specification. It also passes the in order segments received to the application using the *recv_to_app* FIFO. The *receiver* process is created by the tpa_accept() function upon successful completion of the free-way handshake. tpa_accept() also creates the *recv_to_app* FIFO. The application working on top of TPA must use the tpa_recv() function to receive TPA segments. More specifically, tpa_recv() reads the *recv_to_app* FIFO for TPA segments and returns as soon as it collects the number of bytes specified in its parameters. Upon reception of the FIN segments, the receiver process closes the *recv_to_app* FIFO to inform the application that the sender side have closed the connection. As soon as tpa_recv() finds the *recv_to_app* FIFO closed, it returns with a 0 value. At this point the application can close the receiver side of the connection using the tpa_close() function.

At the end of the tear down phases all TPA processes terminate.

15.9.3 Timer implementation

The TPA protocol requires distinct timers for each packet sent. However, in a GNU/Linux process is not possible to use more than one software timer at the same time. To overcome this problem, in my TPA prototype I implemented a list of data structures, each of which representing a timer, managed by a *timer scheduler* [77]. With this technique, I can implement multiple timers using only one software timer.

Each data structure representing a timer has a field (*expire time*) representing the value to be assigned to the software timer. The software timer is started for the timer represented by item (*active timer*) at the head of the list of timers. On timer expiration, the *timer scheduler* removes the item at the head of the list and sets the seconds item in the list as the *active timer*. The software timer is then restarted with the *expire time* of the new *active timer*. The timer scheduler, whenever a new timer is started or a timer belonging to the list is removed, must update the *expire time* of each item in the list and must eventually reorder the list of timers. Specifically, the timer scheduler must preserve the ordering between items in function of their *expire time* (i.e., the first item of the list must represent the smallest timer in the list).

Figure 15.6 reports a simple example showing the insertion of new timers in the list of timers. I first start a timer that will expire in 4 seconds (Figure 15.6a). To schedule this timer, the *timer scheduler* inserts an item with value 4 in the list of timers. This item becomes the *active timer*. After one second, a new timer that will expire in 9 seconds is started. To schedule this new timer, the *timer scheduler* inserts in the list of timers a new item with value 6 (Figure 15.6b). This value is given by the difference between the timer value (9) and the remaining value for the active timer (4-1). After other 2 seconds, a new timer that will expire in 3 seconds is started. To schedule this new timer, the *timer scheduler* inserts in the list of timers a new item with value 2 (Figure 15.6c). This value is given by the difference between the timer value (3) and the remaining value for the *active timer* (4-3). Since this timer has been inserted in the second position of the list (to maintain the ordering between items), the *timer scheduler* must also update the value for the last item. Specifically, it

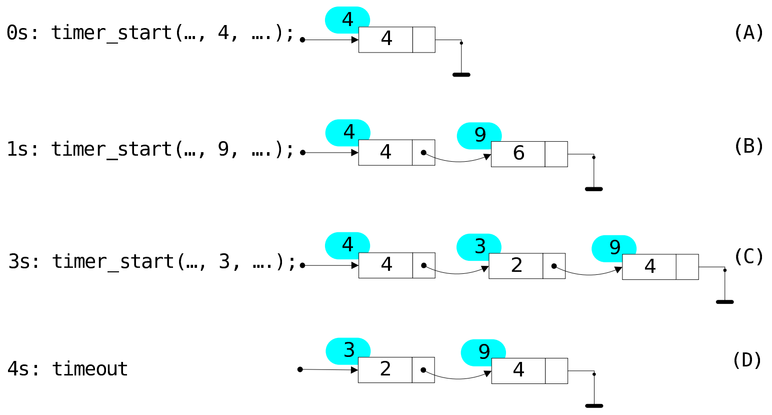

0s: timer_start(…, 4, ….);
4
4
(A)
1s: timer_start(…, 9, ….);
4
4
9
6
(B)
3s: timer_start(…, 3, ….);
4
4
3
2
9
4
(C)
4s: timeout
3
2
9
4
(D)

Figure 15.6: Implementation of software timers.

sets the value for last item to 4 seconds, where 4 is the difference between the new inserted timer (2) and the timer value (6). When the timer expires, the timer scheduler chooses the second item of the list as the *active timer* and sets the *software timer* to its value (2) (Figure 15.6c).

To achieve a finer time granularity, in the *timer scheduler* implementation I used the system call setitimer() to set the software timer. This allows to achieve an accuracy in the order of milliseconds. This is very important since in multi-hop ad hoc networks the round trip time of connections is in the order of milliseconds.

Chapter 16

Experimental Analysis of TPA

16.1 Introduction

Previous experimental studies have shown that certain aspects of real MANETs are often not effectively captured in simulation tools [14]. Furthermore, available software and hardware products often use parameters settings different from those commonly assumed in simulation tools. Finally, real operating conditions are often different from those modeled in simulation experiments. For example, interferences caused by WiFi hotspots or other devices in the proximity are inevitable in practice. For all the above reasons, I used my TPA prototype to compare TCP and TPA performance in a real environment. Specifically, I compared TCP and TPA performance over two static network topology (i.e., chain and cross topologies) and over one dynamic network topology (i.e., roaming node topology).

In the past year almost all TCP studies relied on simulation. To the best of my knowledge, very few experimental analysis have been carried out so far [55, 57, 71]. In addition, these studies lucks of important details. Fu et. all [55] didn't use any routing protocol in their analysis. Also in [71] the authors didn't use any routing protocol in their analysis. In addition, they limited the *cwnd* of TCP to 3/2 h, where h is the number of hops between the sender and the destination node, while previous studies suggested different values [76, 131, 55, 34]. Gupta et. all [57] instead, didn't clump the TCP congestion window size. For these reasons, I started my TPA analysis with a detailed study of TCP behaviour in a real environment. This study permit me to find the optimal setting for TCP over a chain network topology. I then used this optimal setting to perform a fair comparison between TCP and TPA performance. To make my analysis more accurate, I used in the experiments two very popular routing protocols, i.e., AODV [100, 1], and OLSR [37, 120] which take a different approach on building and maintaining routes (reactive vs. proactive).

This Chapter is organized as follow. Section 16.2 describes the testbed used in the experimental analysis. Section 16.4 describes the methodology used to perform my experimental analysis. Section 16.3 describes the performance metrics used to compare TCP and TPA performance. Section 16.5 reports the main results of TCP analysis over a chain topology network. Section 16.6 discusses the results obtained from the experimental comparison between TPA and TCP. Section 16.7 concludes the Chapter.

16.2 Testbed Description

My testbed consisted of IBM R-50 laptops equipped with integrated Intel Pro-Wireless 2200 wireless cards. All laptops were running the Linux Kernel 2.6.12 with the latest available version of the ipw2200 driver (1.1.2). Wireless cards followed the IEEE 802.11b specifications with maximum bit rate set to 2 Mbps (which is the setting used in the vast majority of related works). The RTS/CTS mechanism was enabled and RTS/CTS threshold was set to 100 bytes so that RTS/CTS handshake was active for data segments and disabled for ACKs. This setting protected *long* data segments from collisions, while avoiding RTS/CTS overhead for *short* ACK segments, which are less likely to collide. To make possible to reproduce the desired network topology in an indoor environment I reduced the transmission power of the wireless cards.

In all experiments I used *ftp-like* traffic, i.e., the sender node had always data ready to send. Indeed, file-sharing applications are expected to generate similar type of traffic. To this end, I developed a simple client/server application using Linux sockets operating with TCP protocol and I adapted this application to operate with TPA sockets as well (see Subsection 15.9.1). Several papers have shown the advantage of limiting the maximum TCP *congestion window size* (*cwnd*) in multi-hop ad hoc networks. To be fair, I compared TPA and TCP with limited *cwnd* size. As far as TCP, I used the *TCP_WINDOW_CLAMP* socket parameter that permits to bound the size of the TCP *advertised window*. The *segment payload size* in all the experiments was equal to 1460 bytes. To capture the TCP traffic I used *tcpdump*, while to analyse the experiments results I used *tcpstat* and *tcptrace* (enhanced by my shell scripts). Since TPA was implemented in the user-space, to capture the TPA traffic I used my TPA code.

As anticipated, I compared TPA and TCP performance by considering two different routing protocols, i.e., AODV and OLSR. AODV (Ad hoc On-demand Distance Vector) is a well-known reactive protocol [100]. It can use two different mechanisms for neighbour discovery and local connectivity maintenance (see Section 4.1), i.e., link layer information provided by the underlying MAC protocol, or HELLO messages periodically broadcast by each node to announce its presence in the one-hop neighbourhood. In my testbed I used the AODV implementation for Linux by the Uppsala University [1], version 0.9.1. To maintain local connectivity I set AODV to use HELLO messages since my ipw2200

Table 16.1: TPA Operational Parameters.

Parameter	Value
th_{RC} (TPA)	3 segments
n_{RC} (TPA)	3
g	0.125
h	0.25
g1	0.25
h1	0.5
th_{ROUTE} (TPA)	1 segment
th_{ACK} (TPA)	1 segment
Block Size (TPA)	12 segments

driver didn't provide link-layer failure notifications. All the AODV parameters were set to their default values.

OLSR (Optimized Link State Routing, [37]) is an optimization for mobile ad hoc networks of the classical link state algorithm (it is thus a proactive protocol). OLSR periodically floods the network with route information so that each node can build locally a routing table containing the complete information of routes to all possible destinations within the ad hoc network (see Section 4.2). Similarly to AODV, OLSR employs a neighbour discovery procedure based on HELLO messages. In my testbed I used the OLSR_UniK implementation for Linux, version 0.4.10 [120]. I set all the parameters to their default values, and disabled the OLSR *hysteresis* mechanism, because it was shown to degrade TCP throughput in an unacceptable way (see Section 11.2).

Table 16.1 shows the operational parameters for the TPA protocol. As far as TCP, I configured it to obtain a NewReno behaviour, with Delayed-ACKs in addition.

16.3 Performance measures

In my analysis I considered the following two performance measures:

- *Throughput*, i.e., the average number of bytes successfully received by the final destination per unit time.
- *Retransmission index*, i.e., the percentage of segments re-transmitted by the TPA/TCP sender.

The *throughput* was measured at the application layer as the number of bytes successfully received by the destination process in a given time interval, divided by the duration of the time interval. The *re-transmission index* (*rtx*) measures the average number of times a packet has to be retransmitted to be successfully received by the destination. Thus, it was obtained as:

$$rtx = \frac{pktRtxSrc}{pktRcvDest}$$

where *pktRtxSrc* is the number of packets retransmitted by the source, and *pktRcvDest* is the number of nonduplicated packets successfully received by the destination.

The re-transmission index allows us to evaluate the ability of TPA/TCP to handle transmission in an efficient way. It is worthwhile to emphasize that re-transmitted segments consume energy and generate congestion both at the sender and intermediate nodes. As nodes in a multi-hop ad hoc network may have limited power budget, and wireless bandwidth is a scarce resource, it is important to manage (re)transmission efficiently. Therefore, a small value for the re-transmission index is highly desirable.

16.4 Experimental Methodology

When dealing with real testbeds one of the main difficulties is that experiments cannot be repeated exactly in the same way since external conditions may vary from time to time – sometimes during the same experiment – and there is definitely no control on them. Therefore, successive experiments carried out under the same operating conditions may provide outcomes that differ significantly from each other. This makes comparison of performance measurements obtained in different scenarios or operating conditions hard or even impossible.

To achieve more statistical accuracy, I replicated each experiment multiple times, so to obtain the 90% confidence intervals of the measured performance metrics below the 10%. I then averaged the performance measures over the entire set of replicas. I organized each replica of the experiments as follow. In the experiments of section 16.5 I first evaluated the performance of TCP using a *cwnd* size bounded to 2 segments. Then I evaluated the performance of TCP using a *cwnd* size bounded to 3 segments. Finally, I evaluated the performance of TCP using a *cwnd* size bounded to 4 segments and an unclamped *cwnd* size. In the experiments described in Section 16.6 I added TPA protocol to the experiments schedule of each single replica. Specifically, I first evaluated the performance of TCP and TPA using a *cwnd* size bounded to 2 segments for both protocols. Then I evaluated the performance of TCP and TPA using a *cwnd* size bounded to 3 segments for both protocols. Finally, I evaluated the performance of TCP using an unclamped *cwnd* size and the performance of *TPA** using a *cwnd* size bounded to 3 segments.

Each replica consisted of a file transfer. To perform multiple replicas the whole process of experimentation (data generation, logging and archiving) was automated using my shell scripts.

16.5 TCP Analysis

As noted in 14.3, an easy way to improve TCP performance over MANET is to clump its transmission window. However, in literature only simulative and theoretical studies are used to evaluate the optimal size for the TCP *cwnd*. Furthermore, previous studies do not study the behaviour of TCP over OLSR for varying *cwnd* size, nor the performance of TCP over AODV when HELLO Messages are used to discover neighbour nodes. In these experiments I investigated the influence of the *maximum congestion window size* on TCP performance in a real environment using AODV and OLSR as routing protocols.

To evaluate the optimal setting for TCP I used a chain topology network with hops count ranging from 1 to 4 (Figure 16.10). It is worth pointing out that this testbed was deployed in a real working environment with possibly interfering electrical appliances, people roaming around, etc. The experiments consisted in a file transfer of about 120s. To generate network traffic I used my *ftp-like* program. In all the experiments node *N1* was the sender, while the receiver (and the number of active nodes) depended on the specific chain length. For example, in the 3-hop scenario, node *N4* was the receiver (node *N5* was not active). I chose the transmission power of wireless cards and the distance between nodes in such a way that only adjacent nodes were within the transmission range of each other. However, since the transmission range of nodes is not a perfect circle and may vary from time to time [15], I used the `iptables` firewall to filter MAC packets and guarantee the desired topology. For example, I set up the firewall of node *N3* in such a way to accept only packets from node *N4* and node *N2*. I will comment on this choice later on in the performance evaluation section.

16.5.1 Influence of the maximum congestion window size

To evaluate the influence of the maximum congestion window (*cwnd*) size on TCP performance I clamped the congestion window size to some specific values. Specifically, I considered maximum *cwnd* sizes of 2, 3, and 4, and performed also experiments where the window size was unclamped (*uc*). In the following I will refer to TCP with maximum *cwnd* size of *W* as *TCP-W* and I will refer to TCP with an unclamped *cwnd* as *TCP-uc*.

Previous simulation studies [34, 55] have shown that in the considered scenario (i.e., chain topology network), the optimal value for TCP *cwnd* is 2. However, as anticipated, previous studies do not highlight the behaviour of TCP over OLSR for varying cwnd size, nor the performance of TCP over AODV when Hello Messages are used to discover neighbour nodes. Therefore I evaluated through ns-2 [3] the optimal value of TCP *cwnd* in these configurations, using the same operational parameters used in experimental analysis. Figure 16.1 shows that also my simulative analysis indicates that 2 is the optimal value for TCP *cwnd*.

The results obtained in my experimental analysis, both in terms of throughput and percentage of retransmissions, are shown in Figure 16.2 through Figure

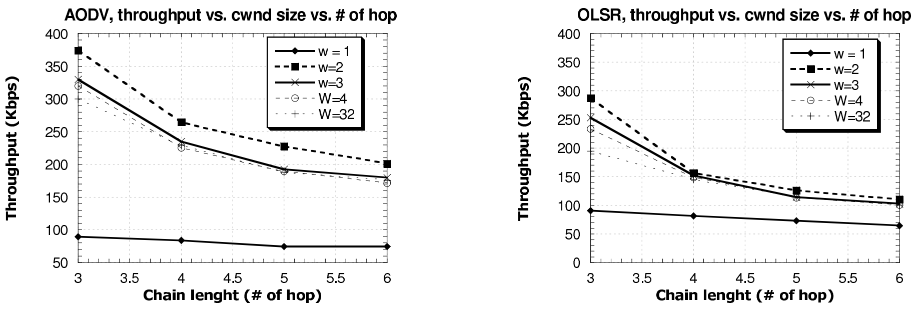


Figure 16.1: Throughput over AODV (left) and OLSR (right) vs. number of hops vs. cwnd size. NS-2 results.

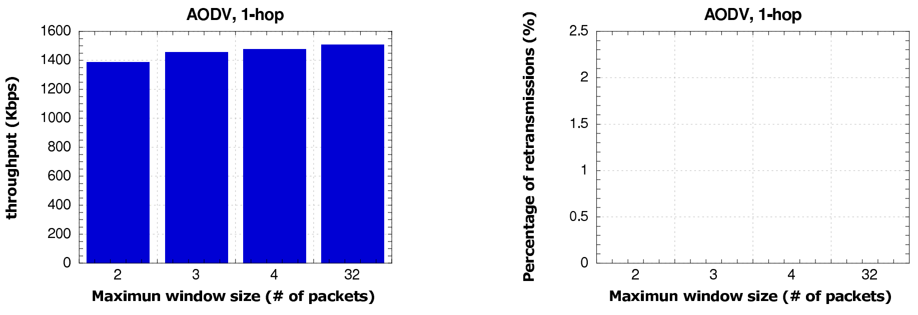


Figure 16.2: Throughput (left) and percentage of retransmitted segments (right) vs. maximum congestion window size in the 1-hop scenario. The routing protocol is AODV.

16.5. These results highlight in the considered environment a maximum *cwnd* of size 2 never provides optimal performance. Specifically, in the *1-hop* scenario an *unclamped* congestion window seems to be the best choice. This is because in the *1-hop* scenario there is no competition between neighboring nodes as in multi-hop scenarios. In the other scenarios, a *cwnd* limitation appears to be beneficial but the optimal *cwnd* size appears to be 3 (in the *2-hop* scenario the throughput with maximum *cwnd* equal to 4 is slightly better, but the retransmission is significantly higher). This discrepancy with previous simulation results is due to a different behaviour between the TCP version implemented in the Linux distribution used in my testbed and the one implemented in common simulation tools (e.g ns-2 [3]) when the maximum *cwnd* of size is set to 2. By a detailed analysis of traces I found that the simulated TCP receiver sends back *one* acknowledgement *every other* segment, while the real (i.e., Linux) TCP receiver sends back *one* acknowledgement *every* segment. When the maximum *cwnd* size is 3 (or larger) both the real and simulated TCP send back one acknowledgement every other segment. The increased number of acknowledgments managed in the real testbed when the maximum *cwnd* size is equal to 2, makes the throughput suboptimal.

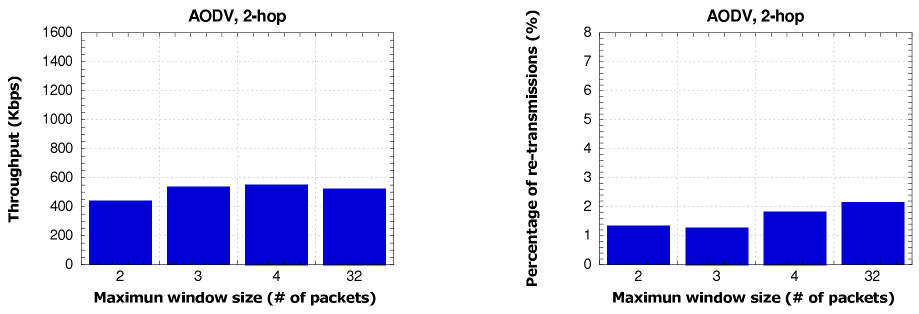


Figure 16.3: Throughput (left) and percentage of retransmitted segments (right) vs. maximum congestion window size in the 2-hop scenario. The routing protocol is AODV.

To confirm my conclusion I used the ns-2 simulation tool [3], and ran simulation experiments where I modeled the above conditions (note that I used the same AODV-UU code both in the real and in the simulated experiments). Specifically, I set the delayed ACK option at the TCP receiver when the maximum cwnd size was equal to 3, 4 and, 32, respectively. Instead, I disabled this option in case of maximum cwnd size was equal to 2. Therefore, in the latter case the TCP receiver sends back one TCP ACK *every* segment received , while in all other cases it sends one TCP ACK *every other* segment. I observed that the optimal window was 3 as in the real experiments. From the above results it also appears that TCP throughput with optimal cwnd size is not so different from that with unclamped congestion window. This is in contrast with previous simulation studies which observe a significant throughput improvement with optimal cwnd size. This discrepancy can be explained in terms of

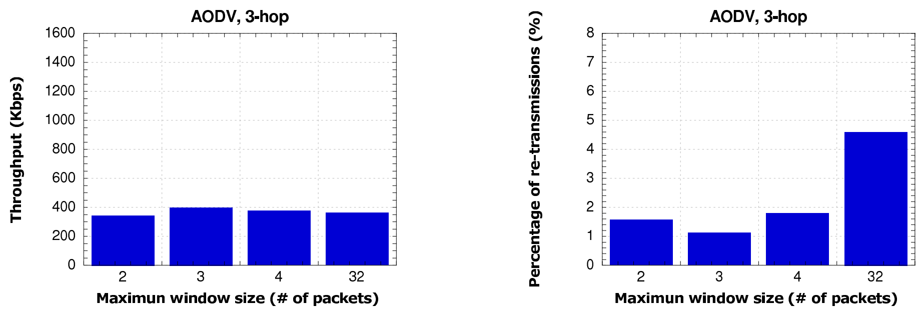


Figure 16.4: Throughput (left) and percentage of retransmitted segments (right) vs. maximum congestion window size in the 3-hop scenario. The routing protocol is AODV.

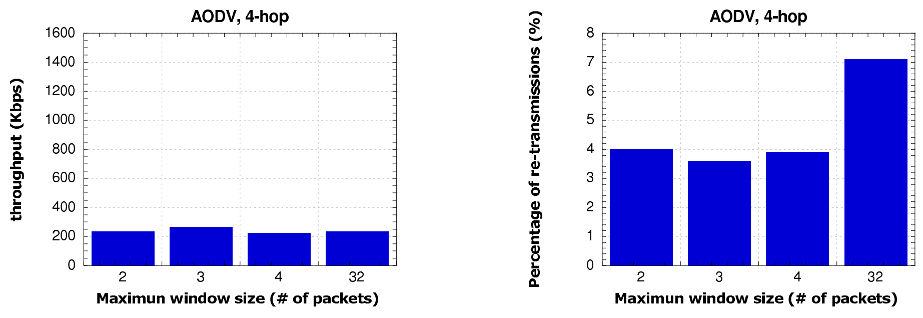


Figure 16.5: Throughput (left) and percentage of retransmitted segments (right) vs. maximum congestion window size in the 4-hop scenario. The routing protocol is AODV.

the mechanism used by the AODV routing protocol for detecting link failures. More precisely, the link failure detection mechanism based on HELLO messages generates frequent route failures with associated throughput oscillations and performance degradation. This issue is described in detail in the next section.

16.5.2 Influence of Hello messages

AODV may take two different approaches for link failure detection. It can either exploit link failure notifications from the underlying layer (provided that this service is available), or rely upon a periodic exchange of HELLO messages. In the former case AODV learns that a link failure has occurred as soon as it receives an explicit notification from the underlying layer (hereafter, this approach will be referred to as AODV-LL). In the latter case each node listens for HELLO messages that are periodically broadcast by each other node in the network. A node assumes that a link failure has occurred if it has previously received a HELLO message from a neighbour and, then, for that neighbour does not receive any packets (HELLO messages or anything else) for more than a predefined threshold (hereafter, this approach will be referred to as AODV-HELLO). In the AODV-UU implementation only HELLO messages and AODV control messages (e.g., RREQ and RREP) are considered for neighbour connectivity assessment (i.e., data messages are not taken into consideration).

The AODV protocol in my testbed uses HELLO messages since the ipw2200 driver (version 1.1.2) does not provide link failure notifications. Assuming default parameter values (HELLO_$INTERVAL = 1s$, and $ALLOWED$_HELLO_$LOSS = 2$), HELLO messages are sent every 1s, and the timeout associated with link failure detection is 2s. In other words , a link failure is assumed in my testbed if a node fails to receive two consecutive HELLO messages from its neighbour. To compare the TCP behaviour with AODV-LL and AODV-HELLO I thus used the ns-2 simulation tool [3]. Specifically, I assumed that the interference range (IF_RANGE) is equal to the carrier sensing range (CS_RANGE) and both are twice as large as the transmission range, and set all the other parameters as in the experimental testbed.

Figure 16.6 and Figure 16.7 show the throughput (left-side plot) and congestion window size (right-side plot) vs. time with AODV-LL and AODV-HELLO, respectively. These results are related to the *3-hop* scenario with maximum cwnd of size 2. However, I found similar results for the other scenarios and maximum cwnd sizes as well. I can observe that with AODV-HELLO, the short *link-failure* detection timeout (2s) causes false *link-failure* detections that forces AODV to trigger a new route discovery process. During route discovery process no segment is transmitted towards the final destination and the instant throughput decreases to zero, as shown in Figure 16.7 (left). In addition, the TCP sender experiences delayed ACKs and/or timeouts which trigger the congestion control mechanism. This is because the cwnd size decreases to one when the throughput is null as shown in Figure 16.7 (right).

When using AODV-LL there is no *link-failure* notification from the data link layer below and, hence, the TCP cwnd size and throughput remain con-

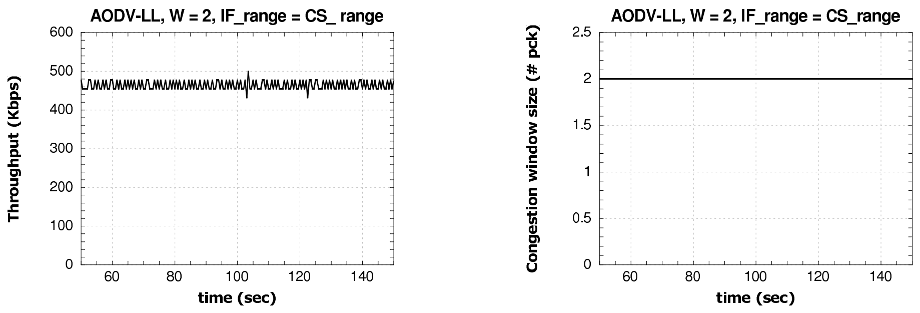


Figure 16.6: Throughput (left) and congestion window size (right) vs. time when using AODV-LL in the 3-hop scenario with maximum cwnd of size 2. The interference range (IF_Range) is assumed equal to the Carrier Sensing Range (CS_Range).

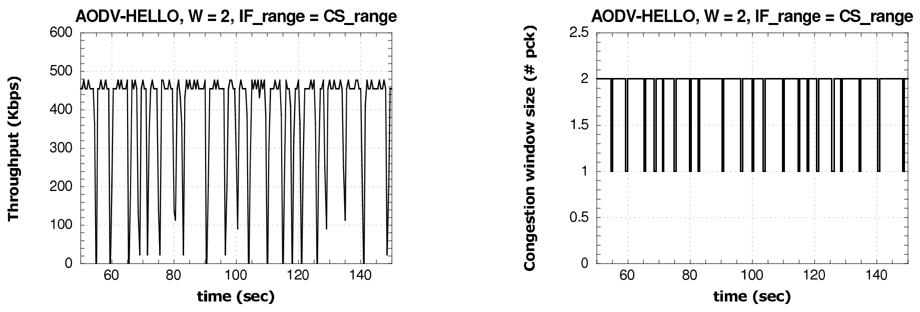


Figure 16.7: Throughput (left) and congestion window size (right) vs. time when using AODV-Hello in the 3-hop scenario with maximum cwnd of size 2. The interference range (IF_Range) is assumed equal to the Carrier Sensing Range (CS_Range).

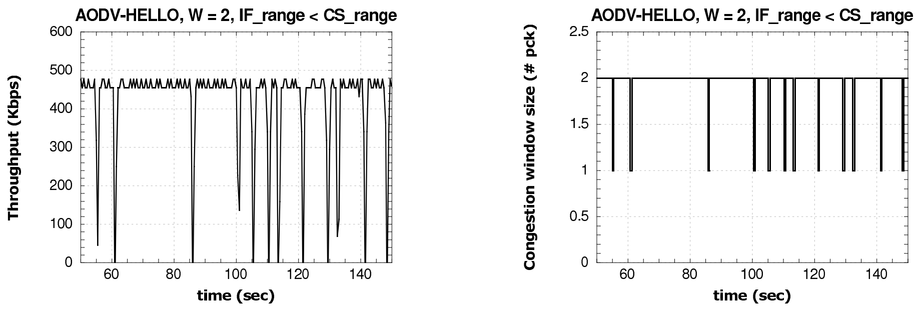


Figure 16.8: Throughput (left) and congestion window size (right) vs. time when using AODV-Hello in the 3-hop scenario with maximum cwnd of size 2. The interference range (IF_Range) is less than the Carrier Sensing Range (CS_Range).

stant, as shown in Figure 16.6. The difference in detections between the two methods can be explained as follows. In the AODV-Hello case it's sufficient to loose two broadcast packets to detect a link failure. In the AODV-LL case a link failure is detected when a unicast packet is lost. Since unicast packets are re-transmitted up to 7 times, while broadcast packets are transmitted just once, the AODV-Hello mechanism proves to detect link failures quite more frequently than AODV-LL. In conclusion, the TCP throughput whit AODV-Hello is significantly lower than that with AODV-LL. In addition, frequent false *link-failure* detections make the TCP throughput with clamped cwnd size not so different from the throughput with unclamped congestion window. Actually, the TCP throughput is limited by *false link-failures* rather than the cwnd size. I also did some simulations runs by assuming $IF\_Range < CS\_Range$, which is more realistic. As expected, I observed no difference when using AODV-LL, and a reduced number of false *link-failure* detections when using AODV-Hello (see Figure 16.8).

16.5.3 Influence of the background traffic

I also investigated the influence of interfering traffic on TCP performance. To this end, I considered the *3-hop* scenario described above and added a CBR (Continuous Bit Rate) session to it. This CBR session has *N3* as its source node and *N2* as it recipient node, and uses UDP as the transport protocol. It inject in the network a periodic traffic pattern with a bit rate equal to 192 Kbps, which correspond to the bit rate of an MP3 stream. The results obtained in this scenario (throughout referred to as 3-hop-UDP) are shown in Figure 16.9. There is no qualitative difference with the results in, except that TCP throughputs are lower and the retransmission indices greater. As in the 3-hop scenario without background traffic, the optimal cwnd size is 3. But, as above, there are not significant differences associated with the various maximum cwnd sizes.

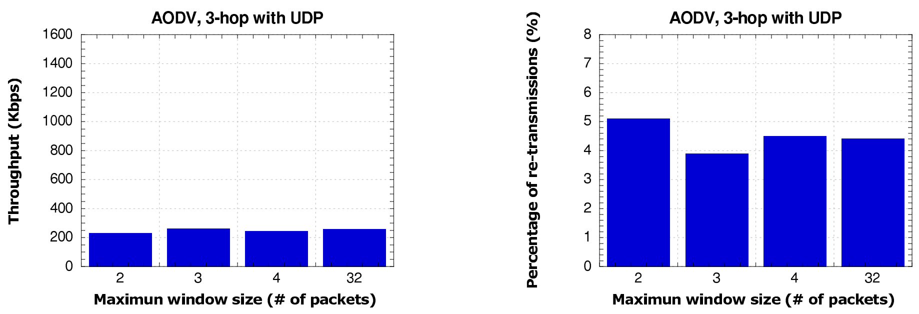


Figure 16.9: Throughput (left) and percentage of retransmitted segments (right) vs. maximum cwnd size in the 3- hop scenario with background periodic UDP traffic . The routing protocol is AODV.

Table 16.2: Throughput (in Kbps) vs. maximum cwnd size with OLSR.

	1hop	2hop	3hop	4hop	3hop-UPD
TCP-2	1369,13	627,3	213,4	175,1	238,9
TCP-3	1456	676,3	282,1	172,8	259,3
TCP-4	1473,6	698,4	229,9	151,8	218,7
TCP-uc	1523,9	696,5	275,5	162,55	233,3

Table 16.3: Percentage of re-transmissions vs. maximum cwnd size with OLSR.

	1hop	2hop	3hop	4hop	3hop-UPD
TCP-2	0	0	1,09	3,55	1,5
TCP-3	0	0,07	1,12	3,2	1,36
TCP-4	0	0	1,45	3,85	1,35
TCP-uc	0	0	1,87	4,8	1,8

16.5.4 Analysis with OLSR routing protocol

To conclude my analysis I also performed some experiments with OLSR as routing protocol. The results obtained, in terms of throughput and percentage of re-transmissions, are summarized in Table 16.2 and Table 16.3, respectively.

I can observe that the results with OLSR are not very different from those with AODV under the same conditions. An important issue is that the retransmission index with OLSR is always significantly lower than that with AODV. I observed that the latter results is confirmed by simulations. One possible reason for this behaviour is the different parameter values used by AODV and OLSR to manage HELLO messages. OLSR sends HELLO messages periodically every HELLO_INTERVAL and considers the information provided by a HELLO message valid for NEIGHB_HOLD_TIME seconds. Assuming default parameter values, HELLO_INTERVAL is set to 2s and NEIGHB_HOLD_TIME to 6s. Therefore, when using OLSR a node considers a link as broken if it fails to receive *three* consecutive HELLO messages from its neighbour. Instead, as described above, with AODV a node assumes a link failure when it fails to receive *two* consecutive HELLO messages. Hence, OLSR is more robust to *false link failures*. In addition, since routing protocols flush out the queue of pending transmissions when they detect a link failure, if AODV detects a larger number of link failures the fraction of segments discarded is larger as well.

16.5.5 Conclusions

TCP performance over multi-hop ad hoc networks (MANETs) have been extensively analysed in many previous studies. However, most of them are based on simulation results, and some of them takes simplistic assumptions, e.g., they do not consider the effect of the routing protocol. On the other hand, several previous studies have shown the importance of an experimental analysis when dealing with MANETs. In these first set of experiments I have used an experimental testbed based on WiFi technology, and measured the TCP performance in an indoor environment by considering two different routing protocols (i.e, AODV and OLSR).

For the sake of simplicity and, also, for better comparison of experimental and simulation results, I have limited my analysis to static networks with a chain topology and a limited number of hops. I have found some interesting results contrasting with simulations. In particular, I have found that in my testbed with a chain topology of four hops or less, the optimal performance is achieved with a maximum cwnd size equal to 3 (instead of 2, as suggested by simulation). In addition, the TCP performance with limited congestion window size is not so different from that achievable with an unclamped congestion window. I have shown that these discrepancies are due to the different protocols – or different protocol implementations – used in practice with respect to simulation tools.

16.6 TPA VS. TCP

This section reports the results of the comparison between TCP and TPA performance over different network topologies. Specifically, I first compared the performance of TCP and TPA in a chain topology with varying number of hops (see Section 16.6.1). For TCP I only reported the results obtained with a *cwnd* set to 2 (optimal value in simulative analysis) and 3 (optimal value in experimental analysis, see Section 16.5). In addition, I also reported the results obtained with an unclamped *cwnd*. Using the results obtained from this set of experiments I evaluated the optimal settings for TPA. Using the optimal setting for both TPA and TCP, I then evaluated the performance of both protocols in a more complex network topology, i.e., a cross topology (see Section 16.6.2). Finally, I evaluated the impact of nodes mobility over TCP and TPA (see Section 16.6.3). To have a complete analysis of the behaviour of TCP and TPA, I used in all the experiments both AODV and OLSR routing protocol.

16.6.1 Chain Topology

I considered a chain topology with hop count ranging from 1 to 4. In addition, I also ran experiments over a 3-hop chain topology in the presence of interfering traffic. Through these experiments I evaluated the impact of the link-layer contention on the TCP/TPA performance in a reasonably neat setup.

In all the experiments i compared the performance of TCP and TPA using different values for the maximum *cwnd* size. Previous simulation studies [34, 55] suggest that in some of my scenarios (i.e., chain topology with hop count equal to 3 or 4) the optimal value for the maximum *cwnd* size is 2, while experimental measurements from a real testbed (see Section 16.5 show that this optimal value is actually 3. Therefore, in the experiments I considered

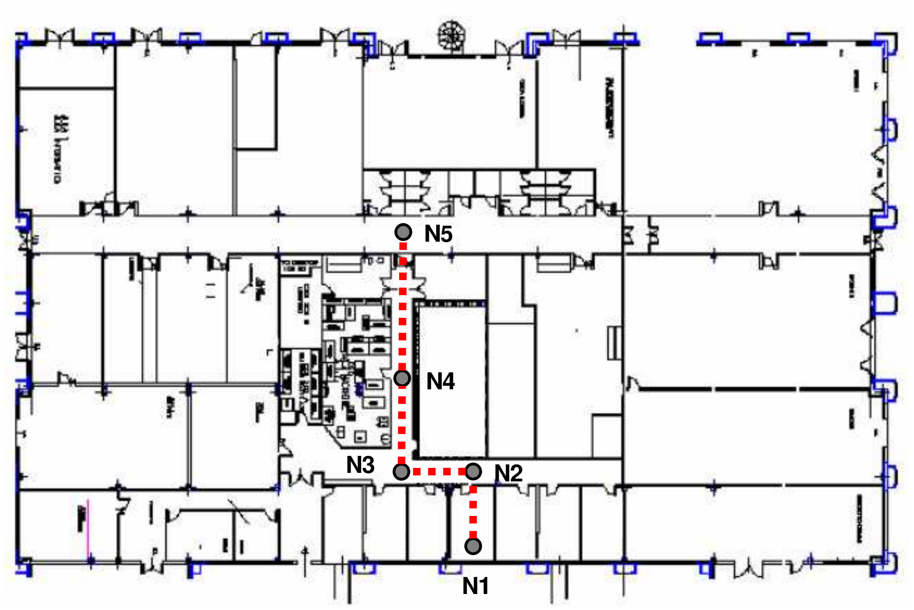


Figure 16.10: Chain Topology network.

a maximum *cwnd* equal to 2 and 3, for both TCP and TPA. As a reference, I also considered an *unclamped* (uc) value for the TCP maximum *cwnd* size. Moreover, I evaluated the performance of TPA with the modified version of the delayed ACK mechanism (see Section 15.7). In the following I will refer to TCP with maximum *cwnd* size of W as *TCP-W*, and I will refer to TPA with maximum *cwnd* set to W as *TPA-W* (or *TPA*-W* for the modified Delayed-ACK case).

Figure 16.10 shows the indoor environment where the experiments were carried out. It also shows how the nodes were positioned to form the chain topology. It is worth pointing out that this testbed was deployed in a real working environment with possibly interfering electrical appliances, people roaming around, etc. We chose such a deployment to compare TPA and TCP in a scenario that is representative of the expected environments where ad hoc networks will operate in. The experiments consisted in a file transfer of about 180s. To generate network traffic I used my *ftp-like* program. In all the experiments node *N1* was the sender, while the receiver (and the number of active nodes) depended on the specific chain length. For example, in the 3-hop scenario, node *N4* was the receiver (node *N5* was not active). We chose the transmission power of wireless cards and the distance between nodes in such a way that only adjacent nodes were within the transmission range of each other. However, since the transmission range of nodes is not a perfect circle and may vary from time to time [15], I used the `iptables` firewall to filter MAC packets and guarantee the desired topology. For example, I set up the firewall of node

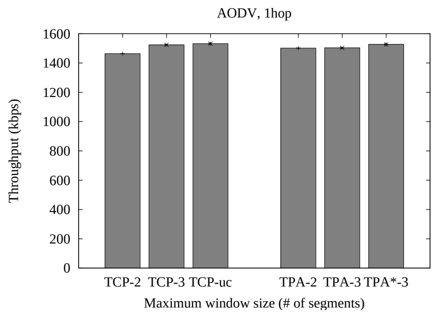


Figure 16.11: Throughput vs. window size in the 1-hop scenario.

N3 in such a way to accept only packets from node *N4* and node *N2*. We will comment on this choice later on in the performance evaluation section.

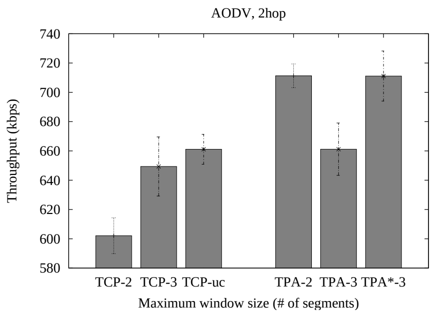
Analysis with the AODV routing protocol

In the next I report the experimental results obtained with the AODV routing protocol. Figure 16.11 through Figure 16.15 show the throughput and retransmission index of both TCP and TPA in all the scenarios I considered.

Figure 16.11 shows that in the 1-hop scenario there are not significant differences between TCP and TPA performance. This was expected since in this simple scenario problems related to link-layer contention are managed efficiently by the IEEE 802.11 MAC protocol. Specifically, in this scenario all nodes are within the transmission range of each other and can thus coordinate efficiently their transmissions. This produces a *retransmission* index equal to zero for all values of the maximum *cwnd* size parameter. Figure 16.11 also shows that *TCP-3* and *TCP-uc* slightly outperform TPA. For example, *TCP-uc* throughput is about 2% higher than *TPA-3* throughput. However, we can observe that *TPA*-3* achieves about the same throughput of *TCP-uc*. It may be worthwhile to recall here that TPA is implemented in the user space and, so, experiences a greater overhead with respect to *TCP-uc* running in the kernel space. This is basically due to a greater number of interactions between the user and the kernel spaces.

Figure 16.12 shows that in the 2-hop scenario TPA outperforms TCP both in terms of throughput and retransmission index. Specifically, *TPA-2* provides an increment in throughput of about 18% respect to *TCP-2* but retransmits about 83% less segments. *TPA-3* provides a throughput about 2% higher than that of *TCP-3*, and reduces the number of retransmitted segments of about 76%. *TPA*-3* instead, provides a higher throughput than *TCP-3* (about 9.5%) and retransmits about 72% less segments. *TPA*-3* is able to provide an higher throughput than *TPA-3*, since it reduces the number of ACKs and, hence, the contention on the wireless channel.

Figure 16.12 also shows that, as in the 2-hop scenario, TCP achieves the optimal throughput when it uses an unclamped transmission window. Specifically,



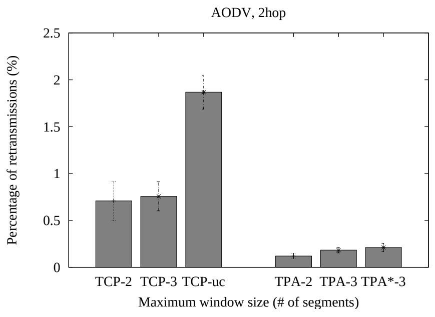
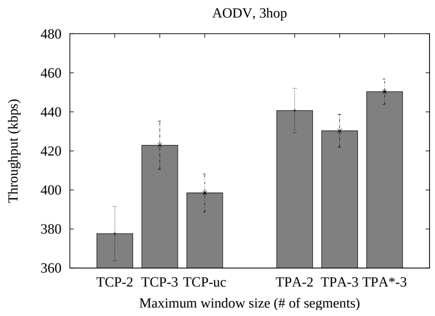


Figure 16.12: Throughput (up) and percentage of retransmitted segments (down) vs. window size in the 2-hop scenario.

TCP-uc throughput is about 2% higher than *TCP-3* throughput. This happens because in a scenario where all nodes are within the same carrier sensing range, the IEEE 802.11 MAC protocol works well. However, *TPA*-3*, that is the best TPA configuration in terms of throughput, provides a higher throughput than *TCP-uc* (about 7.5%) and retransmits about 89% less segments.

The main reason for the difference between TCP and TPA performance in this scenario where there is no interference (all nodes are in the carrier sensing of each other) resides in the mechanism of HELLO messages used by AODV to maintain local connectivity [19]. When using HELLO messages (with the default parameters value), it's sufficient to loose two consecutive HELLO packets to detect a link failure. Since HELLO packets are broadcast, and broadcast packets are neither acknowledged nor retransmitted by the MAC layer, they are vulnerable to collisions and channel errors. This results in a loss of local connectivity even in networks where all nodes are within the same carrier sensing range. This loss of local connectivity results in frequent route failures, which turn out in timeouts and retransmissions at the sender side. However, as expected, TPA is much more efficient than TCP in managing these events, as highlighted by the above results (Figure 16.12 down).

The 3-hop scenario is the first scenario where problems related to link layer contentions become evident [128]. Figure 16.13 shows that the optimal value for the TCP maximum *cwnd* size is 3 segments. This result is apparently in



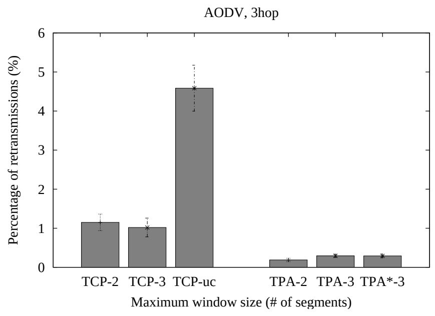
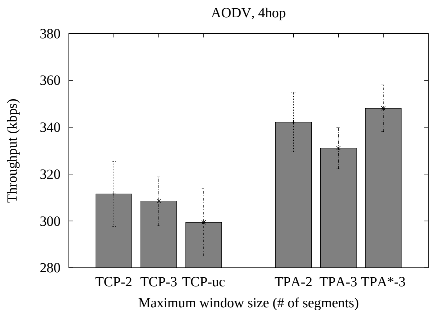


Figure 16.13: Throughput (up) and percentage of retransmitted segments (down) vs. window size in the 3-hop scenario.

contrast with previous simulation results showing that in the 3-hop scenario the TCP optimal window size is 2 segments [34]. This discrepancy has been found to reside in different behaviour between the TCP implementation in the Linux kernel, and the TCP implementation available in the ns-2 simulator [3]. In [19], by a detailed analysis of traces, I found that when the maximum *cwnd* size is set to 2 segments, the simulated TCP receiver sends back one ACK *every other* segment, while the real (i.e. Linux) TCP receiver sends back one ACK *every segment*, as if the delayed ACK mechanism were disabled in this case.

Figure 16.13 shows that TPA outperforms TCP with optimal window size both in terms of throughput and retransmission index (with all congestion window size). Specifically, *TPA-2* increases the throughput of about 17% with respect to *TCP-2*, retransmitting about 83% less segments. *TPA-3* provides a throughput 2% higher than *TCP-3*, while retransmitting about 71% less segments. Figure 16.13 also shows that *TPA*-3* is the best configuration for TPA, providing a throughput increase of about 6.5% with respect to *TCP-3* and retransmitting about 71% less segments.

In the 4-hop scenario, the link layer contention increases and then the difference in performance between the two protocols is expected to become more evident. Figure 16.14 shows that, in this scenario there is no significant difference between *TCP-2* and *TCP-3*. Also the difference between *TCP-uc* and TCP with a clamped window are not significant in terms of throughput. How-



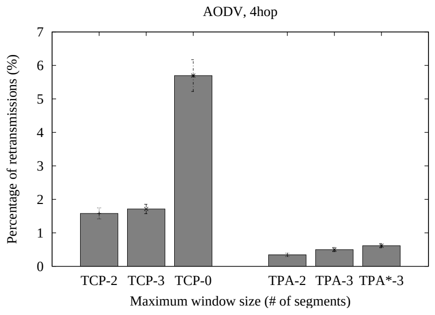


Figure 16.14: Throughput (up) and percentage of retransmitted segments (down) vs. window size in the 4-hop scenario.

ever, *TCP-2* and *TCP-3* obtain a retransmission index of about 70% lower than *TCP-uc*. Figure 16.14 also shows that TPA outperforms TCP for all the considered protocol parameters. *TPA-2* increases the throughput of about is 10% with respect to *TCP-2*, while retransmits about 78% less segments. *TPA-3* increases the throughput of about 7% with respect to *TCP-3*, and retransmits 80% less segments. Also in this scenario *TPA*-3* is the best configuration for TPA, providing an increase of about 12% in terms of throughput with respect to *TCP-2* and a decrease of about 60% in terms of retransmitted segments.

Finally, I also evaluated the performance of TCP and TPA in the presence of interfering traffic. To this end I considered the 3-hop network topology described above and added a CBR (Continuous Bit Rate) session to it. This CBR session has *N3* as its source node, and *N2* as its recipient node, and uses UDP as the transport protocol. It injects a periodic traffic pattern in the network with a bit rate equal to 192 kbps, which corresponds to the bit rate of an MP3 stream. The segment size of the UDP traffic was set to 1460 bytes. The results obtained, summarized in Figure 16.15, show that in this scenario there is no *qualitative* difference with the results obtained in the 3-hop scenario. One important observation is that the performance improvement of TPA is much more evident because the probability of link layer contentions is now greater. With respect to *TCP-2*, *TPA-2* exhibits an increment in throughput of about 28% and a decrement of the retransmission index of about 93%. *TPA-3*

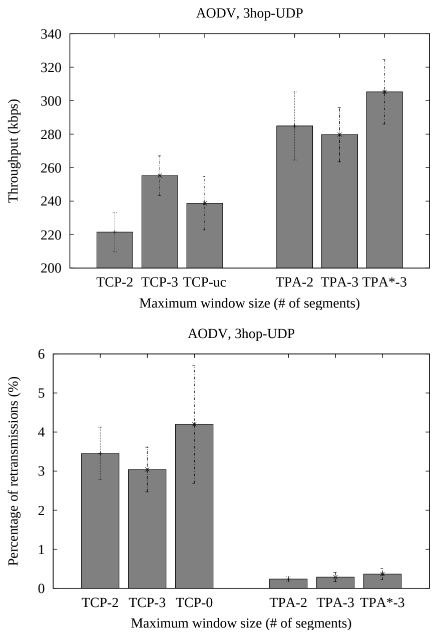


Figure 16.15: Throughput (up) and percentage of retransmitted segments (down) vs. window size in the 3-hop-UDP scenario.

provides a throughput of about 9.6% higher than TCP while retransmitting about 81% less segments respect to *TCP-3*. We can observe that also in this scenario *TPA*-3* is the best configuration for TPA, providing an increment in throughput of about 19% with respect to *TCP-3* and retransmitting about 88% less segments.

Analysis with the OLSR routing protocol

In the next I show the experimental results obtained in the same above scenarios using OLSR, instead of AODV, as the routing protocol. Table 16.4 and Table 16.5 summarize the throughput and the retransmission index, respectively. We can see that the results obtained with OLSR are similar to those obtained with AODV. As soon as problems related to link-layer contention become evident, TPA outperforms TCP both in terms of throughput and retransmission index. For example, in the 3-hop scenario *TPA-2* increases the throughput of about 16% with respect to *TCP-2* while retransmitting about 98% less segments. Throughput with *TPA-3* is 11% higher than with *TCP-3*, and retransmission are 92% less. Finally, *TPA*-3* provides a throughput about 11% higher that *TCP-uc* (that is the optimal configuration for TCP) and retransmits about 83% less segments.

Table 16.4 and Table 16.5 furthermore show that TCP with a clamped congestion window is the best configuration for TCP in terms of throughput,

Table 16.4: Throughput (in kbps) vs. maximum cwnd size with OLSR.

	1hop	2hop	3hop	4hop	3hop UDP
TCP-2	1369 ±25	619 ±14	354 ±19	231 ±18	300 ±13
TPA-2	1464 ±5	711 ±23	412 ±15	259 ±17	340 ±14
TCP-3	1456 ±16	672 ±14	357 ±18	235 ±15	320 ±13
TPA-3	1461 ±19	680 ±14	396 ±14	263 ±11	322 ±13
TCP-uc	1524 ±5	691 ±13	371 ±26	253 ±15	315 ±16
TPA*-3	1495 ±3	730 ±24	411 ±11	267 ±13	347 ±13

Table 16.5: Retransmission index vs. maximum cwnd size with OLSR.

	1hop	2hop	3hop	4hop	3hop UDP
TCP-2	0 ±0	0.02 ±0.01	0.4 ±0.15	1.3 ±0.2	0.67 ±0.24
TPA-2	0 ±0	0 ±0	0.01 ±0.006	0.09 ±0.04	0.02 ±0.02
TCP-3	0 ±0	0.02 ±0.01	0.54 ±0.18	1.38 ±0.24	0.5 ±0.19
TPA-3	0 ±0	0 ±0	0.04 ±0.016	0.15 ±0.05	0.07 ±0.01
TCP-uc	0 ±0	0 ±0	0.77 ±0.24	2 ±0.3	0.96 ±0.4
TPA*-3	0 ±0	0 ±0	0.08 ±0.04	0.16 ±0.05	0.03 ±0.001

only in the presence of background traffic. In all the other scenarios, *TCP-uc* achieves an higher throughput than *TCP-2* and *TCP-3*. This phenomenon can be explained considering the low percentage of retransmission achieved by *TCP-uc* when OLSR is used. For example, in the 4hop scenario, *TCP-uc* over OLSR retransmits about 65% less segments that over AODV. One possible reason for this difference in retransmission index is the different parameter values used by AODV and OLSR to manage HELLO messages. Specifically, by considering the default parameter values for both AODV and OLSR, OLSR assumes that a link is broken if it fails to receive *three* consecutive HELLO messages form its neighbour, while AODV assumes a link failure when if fails to receive *two* consecutive HELLO messages. This make OLSR more robust to false link failures [19], and results in a *retransmission index* closer to zero in the 2-hop scenario when using OLSR.

Finally, Table 16.4 and Table 16.5 also show that *TPA*-3* always provides the best throughput for TPA. The only exception is in the 3-hop scenario, where *TPA-2* and *TPA*-3* achieve the same throughput.

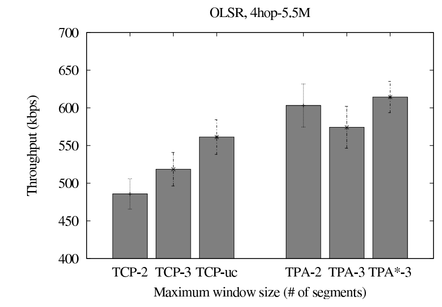
Analysis With Different Transmission Rate

I evaluated the impact of the transmission rate on both TCP and TPA performance. Specifically, I considered the 4-hop network topology with OLSR as routing protocol, and I ran experiments using 5.5 Mbps and 11 Mbps as transmission rate. The obtained results show that also with transmission rates different that 2 Mbps, TPA is able to outperform TCP in terms of both throughput and retransmision index.

Figures 16.16 and 16.17 shows the results obtained with a transmission rate equal to 5.5 Mbps and 11 Mbps respectively. When a transmission rate equal to 5.5 Mbps is used, TPA outperforms TCP with optimal window size both in terms of throughput and retransmission index with all congestion window size. Specifically, *TPA-2* increases the throughput of about 24% with respect to *TCP-2*, retransmitting about 91% less segments. *TPA-3* provides a throughput about 11% higher than *TCP-3*, while retransmits about 82% less segments. Finally, *TPA*-3* (the best TPA configuration), provides a throughput of about 9.4% higher than *TCP-uc* (optimal TCP configuration), retransmitting about 94% less segments. When a transmission rate equal to 11 Mbps is used, the difference in throughput between TCP and TPA is less evident. Specifically, *TPA-2* and *TPA-3* prodive about the same throughput that *TCP-3* (the optimal TCP configuration), while retransmit about 80% less segments. However TPA in its optimal configuration still to outperform TCP in its optimal configuration. Specifically, *TPA*-3* (the optimal TPA configuration) obtains an increment in throughput of about 4% respect to *TCP-3*, retransmitting about 80% less segments.

Trace Analysis

The aim of this section is to show the different behaviour of TPA and TCP in the presence of an ACK *inhibition*, i.e., when the route between receiver and



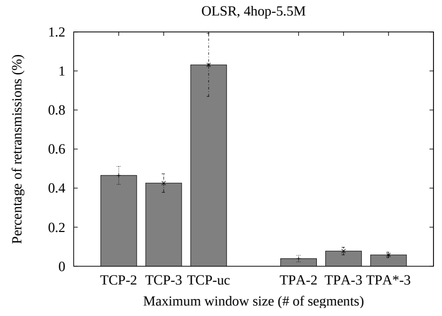
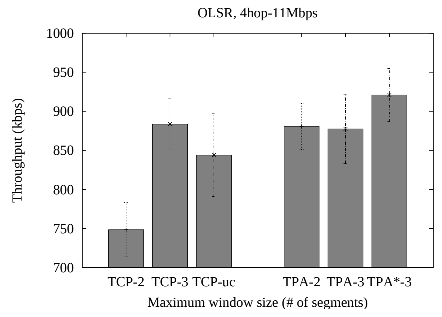


Figure 16.16: Throughput (up) and percentage of retransmitted segments (down) vs. window size in the 4-hop with 5.5Mbps transmission rate



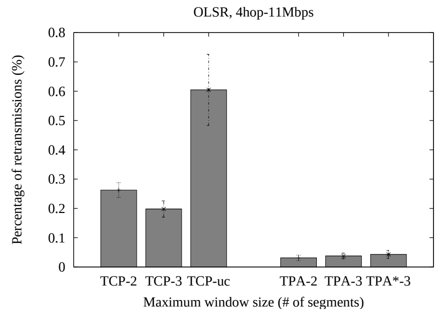


Figure 16.17: Throughput (up) and percentage of retransmitted segments (down) vs. window size in the 4-hop with 11Mbps transmission rate

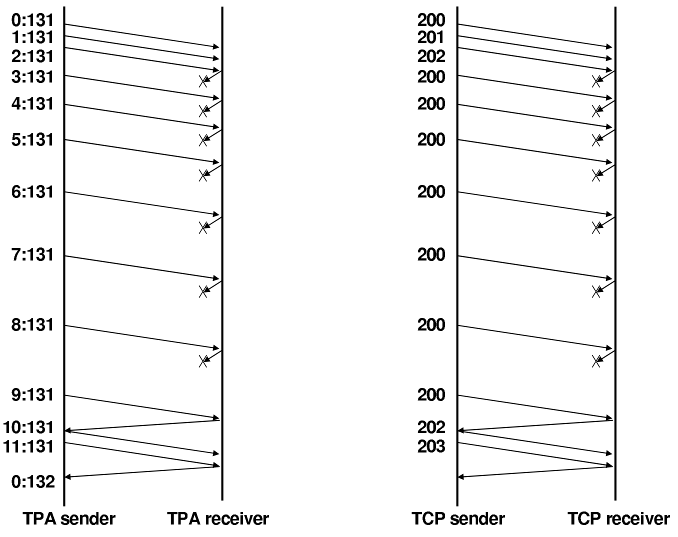


Figure 16.18: Impact of an ACK inhibition.

sender is broken, while the route between sender and receiver is still available. This can help us to understand why TPA outperforms TCP. To this end I refer to a portion of the TPA trace file obtained in one replica of the 3-hop OLSR experiment.

Figure 16.18-left shows the behaviour of TPA after about 39.557s from the start of the experiment. Numbers on the left-hand side should be read as $<$ *position_in_the_block* : *block_number* $>$. At this time the TPA sender is transmitting segments belonging to the block 131. The route between node N4 (TPA receiver) and node N1 (TPA sender) is broken while the route between node N1 and N4 is active. This implies that TPA data segments can reach node N4 while ACKs are dropped by the routing protocol and never reach node N1. Upon timers expiration for segment 0, TPA enters the *congested* state and starts sending segments with a $cwnd_{max}$ parameter set to one (segments 3 to 9 are sent at each timer expiration). Those segments are successfully received by the destination that continues to send ACKs to the sender node. However, ACKs are discarded by the routing protocol. At time 50.92s, the routing protocol recovers the route to node N1. At this point the TPA receiver, on reception of segments 9, sends back an ACK that reaches the TPA sender and notifies it that all segments belonging to block 131 have been successfully received by the destination. Upon reception of the above ACK, TPA leaves the *congested* state and sends segments 10 and 11. Then, upon reception of the ACK for segment 11, TPA transmits segments belonging to a new block.

Figure 16.18-right shows the behaviour that TCP would have had in the same conditions. In this case, numbers on the left-hand side should be read as

$< segment\_sequence\_number >$. For the sake of simplicity I will refer to TCP sequence number in terms of packets instead of bytes. As I can see from the sequence shown in Figure 16.18-right, upon timer expiration for segment 200 the TCP sender retransmits the same segment and continues to do so upon recursive timeouts.

In such cases, in which data segment reach the destination, but ACK segments do not reach the sender, TPA is not stuck at retransmitting the same segment, as legacy TCP is. Therefore, the destination continues to receive new data segments. In other words, during ACK inhibitions, TPA is able to exploit the unidirectional route available between the sender and the destination, while legacy TCP is not. This results in a performance improvement of TPA, compared with TCP behaviour.

Summary of the main results

Before proceeding with the analysis in more complex scenarios, it is worth summarizing the main results obtained so far in a chain topology. Specifically we have shown that:

- the throughput of TPA (in its best configuration) is always higher than the throughput of TCP (in its best configuration), unless in the 1-hop case. Specifically, the TPA throughput is between 5% and 19% higher than the TCP throughput. It is worth recalling that TPA is currently implemented in the user space, and this causes a non-negligible reduction of the TPA throughput. We can reasonably expect the difference between TPA and TCP in the one-hop case to disappear if also TPA were implemented in the kernel.

- TPA (in its best configuration) retransmits between 64% and 94% less segments than TCP (in its best configuration) does. This results in lower energy consumption and lower congestion in the network.

- *TPA*-3* is generally the best configuration for TPA, since it reduces the number of ACK in transit in the network, while *TPA-3* always performs bad than *TPA-2* and *TPA*-3*.

- The best configuration for TCP varies with network topology and routing protocols. However, in the presence of background traffics, *TCP-3* seem to be the best configuration for TCP. In the remaining set of experiments, we only considered *TCP-3* and *TPA*-3*.

- In my experiments, the use of a clamped *cwnd* for TCP doesn't provide significant advantages in terms of throughput. This is because, as observed in [19], when the HELLO messages are used to maintain the local connectivity, the TCP throughput is limited by false link-failure rather than the *cwnd* size.

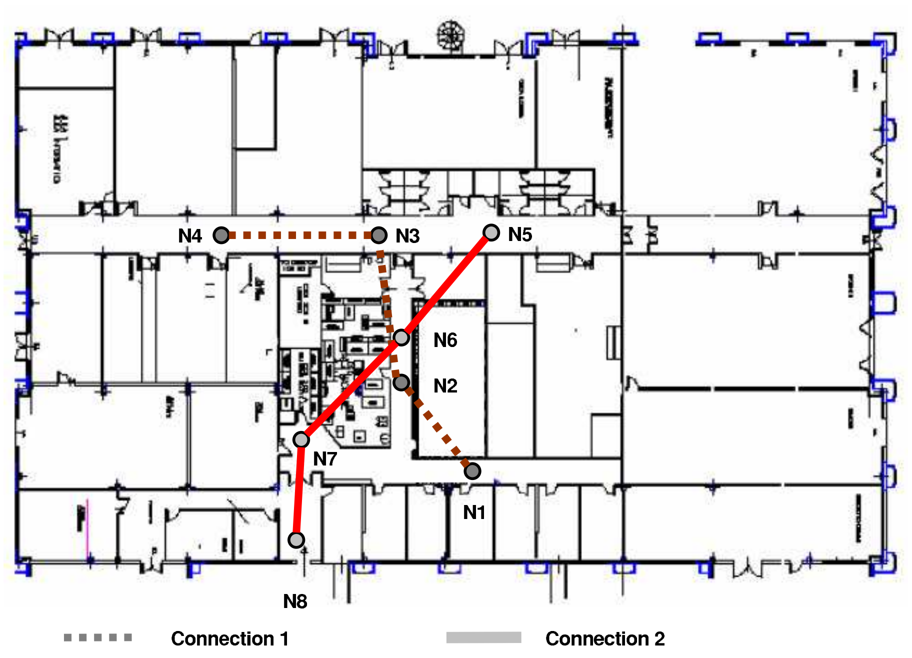


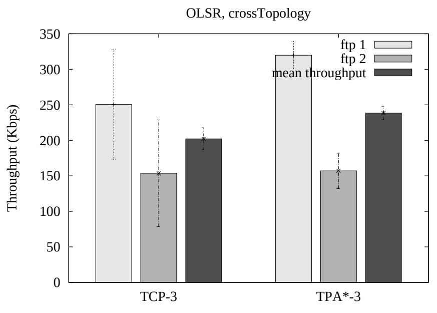
Figure 16.19: Cross Topology network.

16.6.2 Cross Topology

Experiments presented so far allowed us to understand the TPA performance in a fairly simple scenario. This is essential to clearly understand its behaviour, and tuning its parameters. In the next sections, I consider more complex scenarios, starting from the well-known cross topology [46, 55]. Recall that the protocols configurations I compare hereafter are *TCP-3* and *TPA*-3*.

Figure 16.19 shows the nodes position in the indoor environment I used to carry out my experiments. In this scenario there are two TCP/TPA connections, the first one (referred to as connection 1) from node *N1* (sender) to node *N4* (receiver) and the second one (referred to as connection 2) from node *N8* (sender) to node *N5* (receiver). Each connection is three hops long. As in the single-chain case, I positioned nodes of each chain so that only adjacent nodes of a chain were in the transmission range of each other.

In a real environment is hard or even impossible to build a cross topology like that used in simulative analysis. For example, in the cross topology used in Section 17.4, nodes belonging to different connections cannot communicate with each other, with the exception of the middle node. In my testbed, instead, nodes belonging to different connections can communicate with each other. For example, node *N3* is within the transmission range of node *N5*, and node *N6* is within the transmission range of node *N2*. Consequently, to avoid temporary spurious paths, I re-enforced the paths along the intended chains through `iptables`. However, I make attention to put the source nodes and the



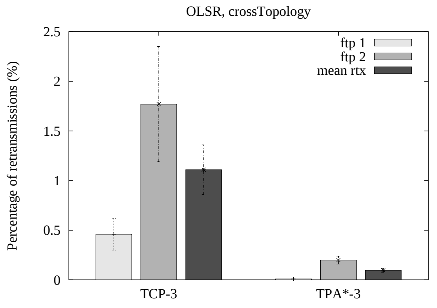


Figure 16.20: Throughput (up) and percentage of retransmitted segments (down) in the cross topology with OLSR.

destination nodes outside of their transmission range. Specifically, node *N8* is outside of the transmission range of node *N1*, while node *N4* is outside of the transmission range of node *N5*.

To generate network traffic I used two *ftp-like* connections as above. I started both connections at the same time and I made them last for 240 seconds. As performance metrics I present both the *throughput* and the *retransmission index* of the two connections. In addition, I also show the *mean throughput*, i.e., the mean between the throughput of the two connections, and the *mean retransmission index*, i.e., the mean between the retransmission indices of the two connections.

Figure 16.20 shows the performance indices achieved over OLSR. Also in this scenario TPA significantly outperforms TCP both in terms of throughput and retransmission index. Specifically, *TPA*-3* increases the *mean throughput* of about 18% with respect to *TCP-3* and reduces the *mean retransmission index* of about 73%. When AODV is used, the TPA performance improvement is less evident. However, Figure 16.21 shows that *TPA*-3* still increases the *mean throughput* of about 5% with respect to *TCP-3*, and reduces the *mean retransmission index* of about 60%.

I can observe that both TCP and TPA suffer a severe unfairness between the two connections. Specifically, connection 1 obtains always a higher throughput than connection 2. In my setup, nodes placement plays a role in the result-

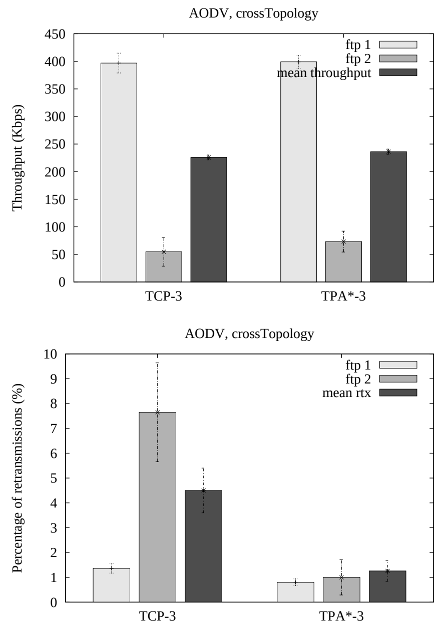


Figure 16.21: Throughput (up) and percentage of retransmitted segments (down) in the cross topology with AODV.

ing unfairness. However, unfairness of transport protocols for ad hoc networks is a well-known problem [46, 127]. Addressing such issues in TPA is still an ongoing work. I present some preliminary design choices and results in Section 17. However, it is worth pointing out that *both* connections achieve higher throughput and lower retransmission indices when TPA is used, even though connection 1 benefits more than connection 2. This means that TPA is able to better exploit the available bandwidth in case of cross topologies.

16.6.3 Roaming Node

To complete my experimental evaluation of TPA I consider the impact of nodes mobility on the performance of both TPA and TCP. To this end, I replicated the *Roaming Node* scenario defined in [80].

The *Roaming Node* scenario (Figure 16.22) consists of four nodes. Three of them are stationary (*N1*, *N2*, and *N3*) and one is mobile (*N4*). Nodes *N1*, *N2* and *N3* form a static two hop chain network, i.e., nodes *N1* and *N3* are out of the transmission range of each other. The mobility pattern followed by node *N4* is as follows. At time 0 node *N4* is in position P1. After 40 seconds it starts to move toward positions P2, where it pauses for 40 seconds. Then it moves towards position P3, where it pauses for 40 seconds. Finally, it goes back to positions P2 and P1, pausing for 40 seconds in each one. The time needed

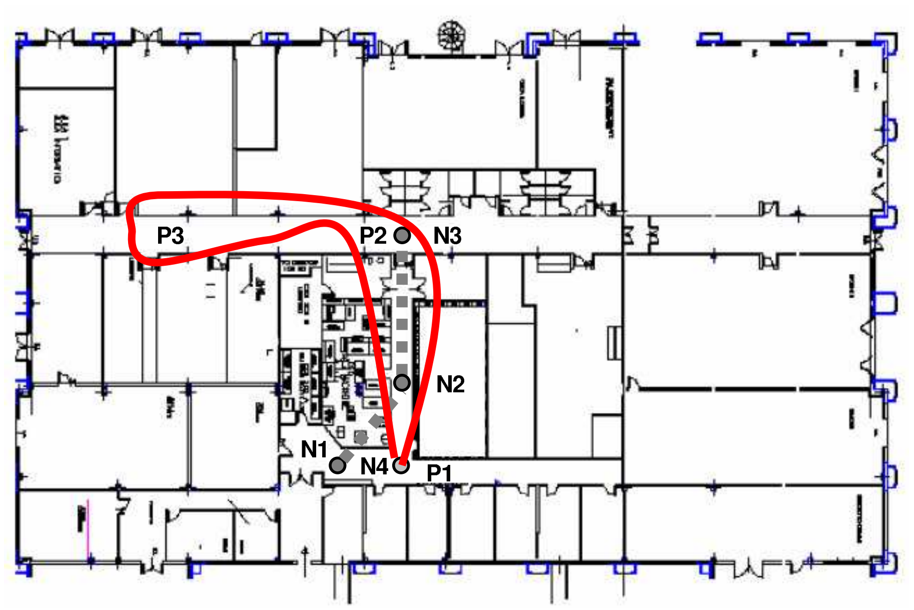


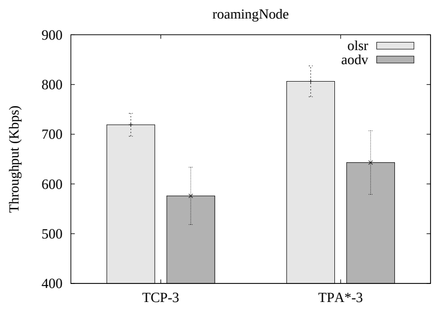
Figure 16.22: Roaming Node Scenario.

to move from a position to the next position is about 20 seconds. Thus, the experiments lasted for about 280 seconds. A single ftp-like connection spanned between nodes N1 and N4 for the whole experiments duration. The length of the path between nodes *N1* and *N4* varied during the experiment. Specifically, in positions P1, P2 and P3, node *N4* could reach node *N1* in 1 hop, 2 hops and 3 hops, respectively. It should be noted that in this set of experiments, to allow routes to be dynamically recomputed I did not enforced paths through `iptables`. I replicated the experiments 5 times, and computed the average values over the replicas.

Figure 16.23 shows the throughput and retransmission index of TCP and TPA over both OLSR and AODV. As I can see, also in this scenario TPA significantly outperforms TCP both in terms of throughput and retransmission index. Specifically, in the OLSR case, *TPA*-3* increases the *throughput* of about 12% with respect to *TCP-3* and reduces the retransmission index of about 73%. When AODV is used *TPA*-3* increases the throughput of about 12% with respect to *TCP-3* and reduces the retransmission index of about 26%.

Figure 16.23 also shows that both TCP and TPA work better over OLSR than over AODV. Specifically, the throughput in the AODV case decreases of about 25% with respect to the OLSR case. This may appear strange, since OLSR suffers from a non negligible *re-route* time, due the rules OLSR uses to generate and disseminate the information about network topology (Topology Control messages, see [37] for more details).

The authors of [91] have analysed the impact of this delay on the perfor-



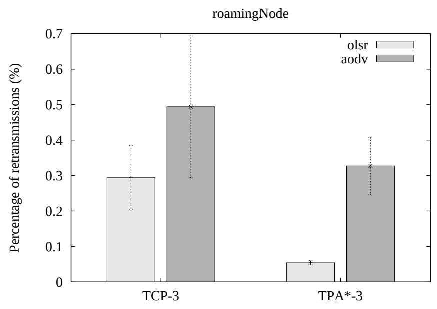


Figure 16.23: Throughput (left) and percentage of retransmitted segments (right) in the roaming node scenario.

mance of TCP. Despite this drawback, OLSR is able to provide a more stable networking environment in my experiments. This can be explained as follows. To positioning nodes I used the `ping` utility. For example, to position node *N3* I used the `ping` utility to guarantee that node *N3* can reach node *N2* but not node *N1*. To find the position P3, instead, I used the `ping` utility to guarantee that in position P4 node *N4* can only ping node *N3*. In this way, I enforced that unicast traffic were routed through the intended hops in the chain. However, as noted in [80], even if two nodes cannot ping each other, the HELLO or RREQ messages (i.e., broadcast messages) can be sporadically received. The difference in size between HELLO and data packets plays an important role in this phenomenon (HELLO packets are smaller that data packets, and hence they are less prone to bit error). In addiction every node in my network may experiences different external condition (like signal to noise level). This may produce some *unidirectional* links.

The presence of spurious HELLO packets affects the behaviour of AODV and OLSR in a different way. As soon as AODV receives a HELLO or RREQ message, it assumes the message sender is reachable, and starts sending *unicast* traffic to it. However, as noted above, receiving a HELLO or a RREQ message does not necessarily means that it is possible to send *unicast* traffic towards the message sender (in [80] this is called the *communication gray zone* problem). On the other hand, in OLSR a link is deemed valid only if both endpoints

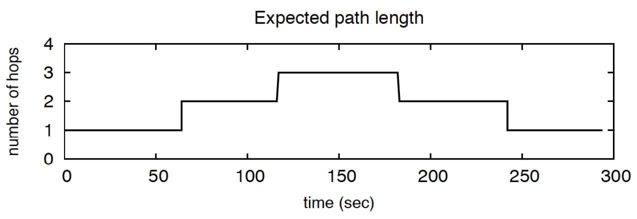


Figure 16.24: Ideal path length during my experiments.

announce each other in their respective neighbour list [37]. It is easy to show that this is a stronger check on the links' validity, which grants more stable routes. In other words, AODV tries to use links as soon as it receives a single broadcast message. Due to wireless links variability, this can make routes too unstable, eventually resulting in low throughput at the transport level.

Based on these remarks, I further investigate the behaviourr of TCP and TPA just in the OLSR case. Figure 16.24 shows the ideal path length between the sender (*N4*) and the receiver (*N1*) that the routing protocol is expected to compute during my experiments. Figure 16.25 shows the real path length (upper plot) and the throughput (lower plot) achieved by TCP (over OLSR) during a particular replica of my experiment (other replicas show a similar behaviour). A number of hops equal to zero means that node N4 has no entries for node N1 in its routing table, and is thus not able to communicate with it. Figure 16.26 shows the same performance figures in the case of TPA. First of all, note that the path calculated by OLSR is comparable in both cases, and is slightly better in the TCP case, as shown by the additional drop after 150s in Figure 16.26 (upper plot). As far as the throughput, TCP performs close to TPA only when the sender and the receiver are 1-hop away, i.e., before 50s and after 250s. When they are 2-hops away, TCP is able to carry on some transfer, as shown by the plateaux around 100s and between 200s and 250s. However, in these time frames TPA is able to achieve a quite more stable throughput. Finally, when the sender and the receiver are 3-hops away, TCP is completely unable to carry on any transfer, while TPA still achieves a stable behaviour in terms of throughput. Therefore, TPA is able to reconfigure after route changes, and carry on data transfer more efficiently than TCP.

16.7 Conclusions

In this chapter I presented an evaluation of TPA protocol in a *real* ad hoc testbed. I have investigated the impact of different protocol parameters on the throughput and the number of segments used to sustain the throughput. I have run experiments on different topologies, different routing protocols, and also in mobile scenarios. Finally, I have also run experiments with higher transmission rate (5.5 Mbps and 11 Mbps). In all the cases I have investigated TPA is able to improve the performance of TCP. Specifically, TPA delivers greater throughput

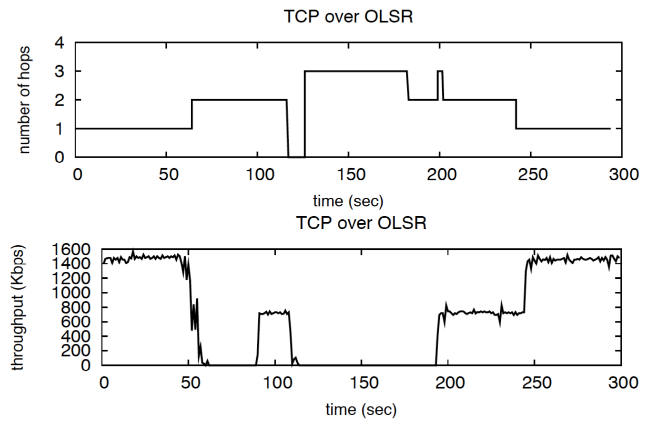


Figure 16.25: Path length and instantaneous throughput of TCP over OLSR in the Roaming Node Scenario.

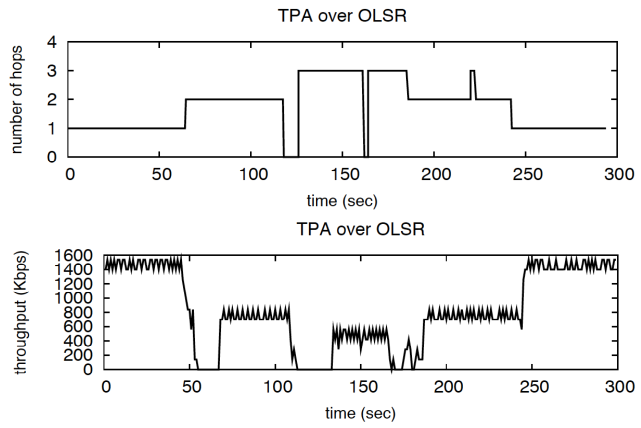


Figure 16.26: Path length and instantaneous throughput of TPA over OLSR in the Roaming Node Scenario.

with respect to TCP (up to 19%), while reducing *at the same time* the number of retransmissions (up to 94%).

The experiments presented in this chapter are tailored to multi-hop ad hoc networks in which groups of users set up a stand-alone network and exchange data in a p2p fashion. The results I have obtained motivate us to further investigate the TPA performance also in different setups. For example, it would be intresting to compare TPA performance with that of different TCP variants, like TCP Sack and TCP Vegas. In addiction, it would be extremely interesting to understand how TPA works in mixed wireless/wired scenarios. In these cases two options could be compared, namely rely on slight TPA modifications to make it work with unmodified TCP endpoints, or using an Indirect-TCP approach, and thus envisioning a first TPA trunk between the wireless node and the gateway to the wired network, and a standard TCP trunk in the wired network. This naturally leads to investigate the viability of TPA in mesh network environments. Another area still to be deeply investigated is how TPA can address fairness issues different and more complex scenarios with respect to those considered in this chapter.

Chapter 17

Simulative Analysis of TPA

17.1 Introduction

The previous Chapter reports the experimental results of TPA analysis over simple network topologies. To complete the study of TPA protocol, I also analysed TPA performance over more complex network topologies, like the cross topology, the parallel topology, and the grid topology. In addiction, I also studied TPA performance over a random generated network topology, and over a mobile scenario. Since it is difficult to build this kind of networks in a real environment, I used the ns-2 simulator [3] to perform my analysis. This also permitted us to take into consideration the unfairness problem. In a real tested, is hard or even impossible to set up connections with exactly the same peculiarity, since different nodes may experience different level of external noise. The difference between connections may make the experimental study of the fairness problem harder. In a simulative environment, instead, is possible to set up connections with exactly the same characteristic. This assists the study of the fairness problem.

Several previous works have analysed the TCP unfairness problem over MANETs (see, for example, [46, 127]). Channel capture, hidden and exposed terminal conditions, and the binary exponential backoff of the IEEE 802.11 MAC are the main causes of TCP unfairness. Moreover, TCP's congestion control mechanism exacerbates the problem. Specifically, as noted in [46], TCP's window-based congestion control mechanism leads to segments' burst on ACK reception. This produces an increased channel contention and reduces the chance of neighbour nodes to access the channel. In [46], ElRakabawy et al. proposed a modified version of the *Adaptive Pacing* mechanism available for the Internet, and called it TCP-AP. They show that TCP-AP can significantly mitigate the unfairness problem encountered by TCP. Even if TPA is able to provide a higher fairness than TCP (as shown in the next sections), it still suffers of a severe unfairness. Consequently, I decided to implement the Adaptive Pacing algorithm proposed in [46] in TPA, and I called this version of TPA TPA-AP. This section reports the results of a comparison between

TCP-AP and TPA-AP. As far as TCP-AP, I used the code implemented by the authors of [46].

To compare TCP and TPA performance, I used the same configuration settings used in the experimental analysis. Specifically, I considered for both TCP and TPA a maximum *cwnd* of 2 and 3. In addiction, I also considered an *unclamped* value the TCP maximum *cwnd* size, and I evaluated the performance of TPA with the modified version of the delayed ACK mechanism. For TCP-AP instead, I used an unclamped value for the maximum *cwnd* size, as suggested in [46], while for TPA-AP I used the same configuration parameters used for TPA. As routing protocol, I used the AODV-UU implementation [1] of the AODV routing protocol. To maintain local connectivity I used the HELLO messages mechanism. I set the *segment payload size* of both protocols to 1460 bytes.

This Chapter is organized as follow. Section 17.2 describes the performance metrics used to compare TCP and TPA performance. Section 17.3 reports the description of the TCP-AP proposal. Section 17.4 to 17.7 reports the results of the analysis of TPA and TCP over the analysed scenarios. Finally, Section 17.8 concludes the Chapter.

17.2 Performance Measures

In my analysis I considered the following two performance measures:

- *Aggregate Throughput* index, i.e., the sum of the throughput achieved by each connection.
- *Mean Re-transmission* index, i.e., the percentage of segments re-transmitted by all the TCP/TPA senders.

The *throughput* was measured as reported in 16.3. The *Mean re-transmission* index (Rtx) was obtained as:

$$rtx = \frac{\sum_{i=1}^{n} pktRtxSrc_i}{\sum_{i=1}^{n} pktRcvDest_i}$$

where $pktRtxSrc_i$ is the number of segments retransmitted by the source i, $pktRcvDest_i$ is the number of nonduplicated segments successfully received by the destination i, and n is the number of connections active in the network.

Several previous works have shown a severe unfairness between TCP connections over MANETs (see, for example, [46, 127]). As a consequence, the *aggregate throughput* is non sufficient to analyse TCP/TPA performance in complex scenarios with multiple connections. To solve this problem, I also used the Jain's fairness index [65] to study the performance of TPA and TCP. This index is defined as:

$$F(x) = \frac{\left[\sum_{i=1}^{n} x_i\right]^2}{n \times \sum_{i=1}^{n} x_i^2},$$

where x_i is the *throughput* (as defined in 16.3) achieved by connection i, and n is the number of considered connections. In addiction, I also considered the *instantaneous fairness* index, i.e., the value of the Jan's fairness sampled every two seconds, and then averaged over the whole experiment.

17.3 TCP with Adaptive Pacing

In [46] the authors show that the TCP's congestion control mechanism exacerbates the unfairness problem. Specifically, they show that TCP's window-based congestion control mechanism leads to segments' burst on ACK reception. This produces an increased channel contention and reduces the chance of neighbour nodes to access the channel. They address this problem by introducing a modified version of the *Adaptive Pacing* mechanism available for the Internet, and called it TCP-AP. TCP-AP spreads the segments transmission according to a transmission rate that is dynamically computed. Moreover it incorporates a mechanism to identify incipient congestion and to adjust consequently the transmission rate. In more detail, TCP-AP calculates the segments transmission rate taking into account the spatial reuse constraint of IEEE 802.11 multi-hop network. The authors of [55] showed that, in a chain topology, only nodes 4 hops away from each other can transmit simultaneously. Based on this result, the authors of [46] used the *4-hop propagation delay (FHD)*, i.e. the time needed for a segment produced by node i to reach node i+4, to calculate the segment transmission rate. TCP-AP sender calculates the *FHD* using the RTT estimation and the number of hops of the connection (see [46] for the details). As anticipated, to evaluate the sender transmission rate, TCP-AP uses also an estimate of the link-layer contention. Specifically, the authors of [46] proposed the *coefficient of variation of recently measured RTT* (cov_{RTT}) as a measure of contention, i.e.:

$$cov_{RTT} = \frac{\sqrt{\frac{1}{N-1} \times \sum_{i=1}^{N} \left(RTT_i - \overline{RTT}\right)^2}}{\overline{RTT}}, \tag{17.1}$$

where N is the number of RTT samples, $\overline{RTT}$ is the mean of the samples, and RTT_i is the value of the *i-th* sample of RTT.

TCP-AP evaluates the transmission rate R as follows:

$$R = \frac{1}{\overline{FHD} \times (1 + 2cov_{RTT})}, \tag{17.2}$$

where $\overline{FHD}$ is the *exponentially weighted moving average* of *FHD* and $(1 + 2cov_{RTT})$ is the factor that takes into account the contention degree of the network. The transmission rate depends on the *FHD* index, and on the link-layer contention. Thus, the connection slows down when the contention increases.

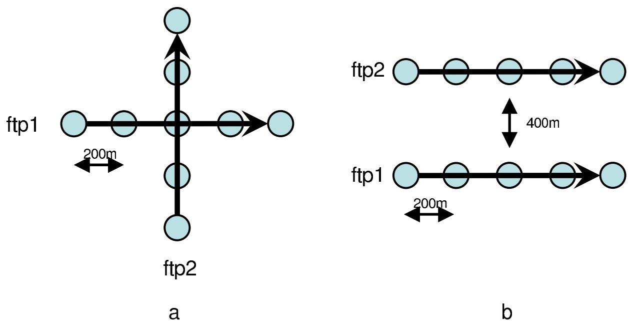


Figure 17.1: Cross and Parallel topology.

Table 17.1: Cross Topology

	Agg. Thr (kbps)	Fair. (%)	Ist. Fair. (%)	Mean Rtx (%)
TCP-2	292 ±1.4	94.6 ±3	63.6 ±1.7	5.13 ±0.49
TPA-2	301.7 ±1.3	98.9 ±0.7	69.6 ±1.3	5.23 ±0.24
TCP-3	282.5 ±2	95 ±4	69.9 ±1.9	8.3 ±0.6
TPA-3	297 ±1.6	99.7 ±0.2	72.4 ±1.25	7.35 ±0.26
TCP-uc	280.7 ±1.7	99.3 ±0.4	70.8 ±1.13	11.4 ±0.43
TPA*-3	309.5 ±1.3	99.3 ±0.36	73.3 ±1.24	7.22 ±0.3

17.4 Cross and Parallel Topology

I analysed TPA and TCP performance over the cross topology (Figure 17.1a) and the parallel topology (Figure 17.1b). These are two of the reference topologies considered in [46]. In both scenarios, the distance between adjacent nodes was set to 200 meters, so as to make each connection 4-hops long. In the parallel topology I set the distance between the two chains to 400 meters. This way, nodes of connection 1 (connection 2) were out of the transmission range of nodes of connection 2 (connection 1), but inside the carrier sensing range of connection 2 (connection 1). In both topologies I considered two *ftp* flows, each starting at time 20 and lasting for 500 seconds. I replicated the cross and parallel topology experiments for 10 times, and I evaluated the 90% confidence intervals for the measured performance metrics.

Table 17.1 shows the results of the experiments in the Cross Topology sce-

Table 17.2: Cross Topology with Adaptive Pacing enabled

	Agg. Thr. (kbps)	Fair. (%)	Ist. Fair (%)	Mean Rtx (%)
TCP-AP	153 ±2.5	99.7 ±0.14	92.3 ±0.1	6.34 ±0.36
TPA-AP-2	254.7 ±2.3	99.3 ±0.7	87.3 ±2.2	2.46 ±0.14
TPA-AP-3	250.6 ±2.7	99.1 ±0.8	86.8 ±1.9	3.3 ±0.16
TPA*-AP-3	250 ±1.4	98.2 ±0.98	88.4 ±1.5	5.8 ±0.28

nario. Also in this scenario, TPA outperforms TCP in terms of throughput, percentage of retransmission and fairness. For example, *TPA*-3* (the best configuration for TPA in terms of *aggregate throughput*), achieves an increment in throughput of about 6% respect to *TCP-2* (the best configuration for TCP in terms of *aggregate throughput*). In addiction, *TPA*-3* achieves an increment in *fairness* of about 5%, and an increment in *instantaneous fairness* of about 16%. However, this increment in fairness and throughput is paid with an increment of the retransmission index of about 40%.

Table 17.2 shows the results of the experiments in the Cross Topology scenario when the Adaptive Pacing mechanism is used. As I can see, the Adaptive Pacing mechanism is able to improve the *instantaneous fairness* of both protocols. For example, TCP-AP obtains an increment of *instantaneous fairness* of about 46% with respect to *TCP-2*. Such greater fairness is paid with a 47% reduction of the *Aggregate throughput*. *TPA-AP-2*, instead, obtains an increment in *instantaneous fairness* of about 25% with respect to TPA-AP. Also TPA pays the increment in fairness with a reduction of the *mean throughput* of about 15%.

Table 17.2 also shows that TPA-AP outperforms TCP-AP in terms of aggregate throughput and percentage of retransmission. Specifically, *TPA-AP-2* (the best configuration for TPA-AP in terms of aggregate throughput) achieves an increment in throughput of about 66%, respect to TCP-AP, while retransmits about 61% less segments. However, this big improvement in terms of throughput is paid by TPA with a reduction of the *instantaneous fairness* index. For example, while the *fairness* index is about the same, *TPA-AP-2* obtains an *instantaneous fairness* index of about 5% lower than TCP-AP.

I analysed the obtained results to understand why TPA-AP obtains a lower *instantaneous fairness* index than TCP-AP. The reason for this is the greater accuracy TPA achieves in estimating the RTT of a connection when the Delayed ACK mechanism is used. Specifically, TPA evaluates the RTT with reference to the segment generating the ACK segment at the receiver, achieving a careful estimate of the real RTT of the connection. For example, in the case depicted

Table 17.3: Cross Topology. Impact of the K parameter on TPA protocol

	$K = 1$	$K = 2$	$K = 3$
Agg. Thr. (kbps)	254.7 ±2.3	208.7 ±3.8	171.2 ±1.1
Mean Rtx (%)	2.46 ±0.14	1.83 ±0.14	1.05 ±0.11
Fair. (%)	99.3 ±0.7	99.7 ±0.18	99.8 ±0.08
Ist. Fair. (%)	87.3 ±2.2	93.7 ±1.09	96.2 ±0.3

in Figure 16.18, on reception of the ACK for segment 11, TPA uses the send time of segment 11 to evaluate the RTT. On the other hand, TCP calculates the RTT with reference to the first unacknowledged segment sent in the current transmission window. For example, in the case depicted in Figure 16.18, on reception of the ACK for segment 203, TCP uses the send time of segment 202 to evaluate the RTT. Consequently, when the delayed ACK mechanism is used, the RTT calculated by TCP includes also the additional time due to delayed ACK. Thus, TCP-AP overestimates the *FHD* and tends to work with a too low transmission rate, eventually resulting in lower throughput with respect to TPA-AP.

I tried to add an additional parameter to formula 17.1, in order to slowing down the pacing timer of TPA-AP. Specifically, I added the K parameter, as reported in formula 17.3, and I evaluated its impact on TPA-AP performance.

$$cov_{RTT} = \frac{\sqrt{\frac{1}{N-1} \times \sum_{i=1}^{N} \left(\mathbf{K} \times RTT_i - \overline{RTT}\right)^2}}{\overline{RTT}}, \tag{17.3}$$

Table 17.3 shows the performance of TPA when a K parameter equal to 1, 2 and 3 is used. For simplicity, only the results obtained with a *cwnd* equal to 2 are reported. The other TPA-AP configurations obtains similar results. Table 17.3 shows that raising the value of K improves the *instantaneous fairness* of TPA-AP. This improvemet in fairness is paid with a reduction of the *aggregate throughput*. For a value of K equal to 2, TPA-AP improves the *instantaneous fairness* of TCP-AP of about 1.5%, maintaining a throughput of about 36% higher and retransmitting about 71% less segments. Summarizing, when a K parameter equal to 2 is used, TPA-AP significant improves TCP-AP performance in terms of *throughput*, *retransmission index* and *instantaneous fairness*. In the following, I will used a K parameter equal to 2 to compare TCP-AP and TPA-AP performance.

Table 17.4 shows the results obtained over the parallel topology when the Adaptive Pacing mechanism is not used. Also in this scenario TPA outperforms TCP in terms of throughput, percentage of retransmission and fairness. Specifically, *TPA*-3* (the best configuration for TPA in terms of throughput),

Table 17.4: Parallel Topology

	Agg. Thr. (kbps)	Fair. (%)	Ist. Fair (%)	Mean Rtx (%)
TCP-2	292.2 ±1.5	83.1 ±11	54.9 ±1.5	2.9 ±0.5
TPA-2	298.9 ±2.1	97 ±2	58.1 ±1.03	2.4 ±0.15
TCP-3	283.3 ±1.6	95.7 ±3	57.9 ±1.44	4.5 ±0.5
TPA-3	294.7 ±2.1	98.1 ±1.6	60 ±1	3.4 ±0.19
TCP-uc	279.5 ±2.5	82.3 ±13	56 ±2.4	6.4 ±0.7
TPA*-3	308.2 ±1.9	97.7 ±1	58.8 ±0.62	3.14 ±0.16

achieves an increment in *aggregate throughput* of about 5%, respect to *TCP-2* (the best configuration for TCP in terms of throughput). In addiction, *TPA*-3* achieves an increment in *fairness* of about 17%, and an increment in *instantaneous fairness* of about 7%. However, also in this case, the increment in fairness and throughput is paid by TPA with an increment of the retransmission index of about 8%.

Table 17.5 shows the results of the experiments in the Cross Topology scenario when the Adaptive Pacing mechanism is used. As I can see, also in this scenario the Adaptive Pacing mechanism is able to improve the *instantaneous fairness* of both protocols. In this scenario, TPA-AP outperforms TCP-AP in terms of throughput, and percentage of retransmission, while achieves a lower *instantaneous fairness* index. Specifically, *TPA-2* (the best configuration for TPA in terms of throughput) achieves an increment in throughput of about 39%, respect to TCP-AP while retransmits 77% less segments. However, while the fairness index is about the same, TPA obtains a *instantaneous fairness* index of about 3% lower respect to TCP-AP. This because TPA-AP, even if a K parameter equal to 2 is used, still to be more aggressive than TCP-AP. However, the reduction of *instantaneous fairness* is small respect to the big increment of *aggregate throughput.*

17.5 Grid Topology

This section reports the results of the simulative analysis of TPA over the grid topology (Figure 17.2). This topology consists on a 5 x 5 grid, where the distance between horizontal and vertical adjacent nodes was set to 200 meters. Over this topology, I set up 6 *ftp* flows, each of wich is 4-hops long. In this

Table 17.5: Parallel Topology with Adaptive Pacing enabled

	Agg. Thr. (kbps)	Fair. (%)	Ist. Fair (%)	Mean Rtx (%)
TCP-AP	155.5 ±4.2	99.8 ±0.1	90.8 ±0.6	7.3 ±0.4
TPA-AP-2	215.3 ±0.9	99.5 ±0.35	87.8 ±1.2	1.7 ±0.16
TPA-AP-3	208.3 ±2.5	99.6 ±0.5	87.7 ±1.6	2 ±0.2
TPA*-AP-3	193.2 ±3.7	97.5 ±1.5	88.4 ±2.2	2.5 ±0.12

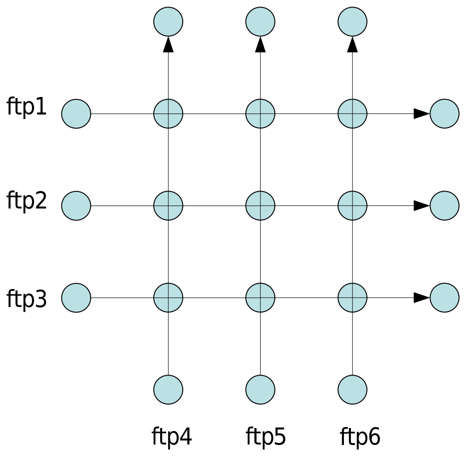


Figure 17.2: Grid topology.

scenario, each ftp flow starts at time 20 and lasts for 500 seconds. I replicated the grid topology experiments for 10 times, and I evaluated the 90% confidence intervals of the measured performance metrics.

Tables 17.6 and 17.7 show the performance of TCP and TPA with and without the Adaptive pacing mechanism over the grid topology. When the adaptive pacing mechanism is not enabled, TPA outperforms TCP in terms of *aggregate throughput*, *fairness* and *instantaneous fairness*. For examples, *TPA-2* achieves an increment in *aggregate throughput* of about 3.7%, with respect to *TCP-2*. In addiction, *TPA-2* obtains an increment in *fairness* of about 43%, and an increment in *instantaneous fairness* of about 34%. However, as in the case of the cross and parallel topologies, this improvement in throughput is paid with an increment of the retransmission index. For example, *TPA-2* retransmits about 22% more segments than *TCP-2*.

Table 17.7 shows that the adaptive pacing mechanism can significantly im-

Table 17.6: Grid Topology

	Agg. Thr. (kbps)	Fair. (%)	Ist. Fair. (%)	Mean. Rtx. (%)
TCP-2	270.1 ±3	62.6 ±5	35.4 ±2	12.3 ±1
TPA-2	280 ±2.9	89.2 ±1.8	47.5 ±0.4	15 ±0.34
TCP-3	263.9 ±4.2	66.5 ±8.6	35.9 ±2	13.7 ±0.8
TPA-3	278 ±2	89.8 ±1.73	48.1 ±0.6	18.9 ±0.2
TCP-uc	268 ±2.7	65.3 ±8	34.4 ±2.6	15.9 ±1.3
TPA*-3	283 ±2	88.2 ±2	47.5 ±0.6	16.4 ±0.4

prove the *instantaneous fairness* of both TCP and TPA also in the grid topology. This increment is paid by TCP and TPA with a reduction of throughput. However, also in this case TPA-AP outperforms TCP-AP. Specifically, *TPA-2* (the best configuration in terms of *instantaneous fairness*) achieves an increment in throughput of about 18%, while retransmits 74% less segments. The *instantaneous fairness* is about the same while TPA achieves a fairness index about 2% lower that TCP.

Table 17.7: Grid Topology with Adaptive Pacing enabled

	Agg. Thr. (kbps)	Fair. (%)	Ist. Fair. (%)	Mean. Rtx. (%)
TCP-AP	202.8 ±2.6	98.6 ±0.9	86.1 ±1.5	28.6 ±1
TPA-AP-2	239 ±2.3	96.1 ±0.9	85.8 ±1	7.5 ±0.4
TPA-AP-3	241 ±1.1	96.5 ±1.7	84.9 ±1.3	8.4 ±0.26
TPA*-AP-3	233.7 ±1.8	95.3 ±1.4	85.5 ±1.6	7.2 ±0.4

17.6 Static Random Topology

In order to test TPA performance in a more generic scenario, I considered a random topology of 50 nodes randomly distributed in a area A=$1000m \times 1000m$. After a warm-up time of 20 seconds, one or more TCP/TPA connections were established over each of which an *ftp* file transfer was conducted for 500 seconds. Specifically, I considered an number of connection simultaneously active in the network equal to 3, 5, and 10. I generate randomly 10 scenarios and I calculated my performance metrics as the mean over the considered scenarios. To achieve a more accurate measure of my performance metrics, I replicated the experiments carried out using a specific scenario 10 times, and I evaluated the mean values for the measured performance metrics. I select the source destination pairs randomly, and I used the same source destination pairs for all the considered scenarios.

Table 17.8 shows the results obtained over the static random scenario, when the Adaptive Pacing mechanism is not used. This table only reports the *aggregate throughput* and the *instantaneous fairness* metrics. This because the *Mean Retransmission* index of both protocols is about the same, while the trend of the *Fairness* and the *Instantaneous Fairness* indexes is comparable. Table 17.8 reports the results for the optimal configuration of both TCP and TPA. The optimal configuration of both TCP and TPA depends on the metric used as reference metric. If the *aggregate throughput* is chosen as the reference metric, the optimal configuration for TCP is *TCP-uc*, while the optimal configuration for TPA is *TPA*-3*. If I used the *instantaneous fairness* as the reference metric, the optimal configuration for TCP and TPA is *TCP-2* and *TPA-2* respectively. Table 17.8 shows both these optimal configurations. $TCP^{\dagger}$ and $TPA^{\dagger}$ are the optimal setting for TCP and TPA when the *aggregate throughput* is chosen as the reference metric, while $TCP^{\ddagger}$ and $TPA^{\ddagger}$ are the optimal setting for TCP and TPA when the *Instantaneous Fairness* is chosen as the reference metric.

Table 17.8 shows that TCP is always able to provide an higher *aggregate throughput* respect to TPA, while TPA always provides an higher *Instantaneous Fairness*. For example, when 5 connections are simultaneously active in the network, $TPA^{\dagger}$ obtains an increment of the *instantaneous fairness* index of about 15% respect to $TCP^{\dagger}$. This increment in fairness is paid by TPA with a reduction of the *aggregate throughput* index of about 1.5%. $TPA^{\ddagger}$, instead, obtains an increment of the *instantaneous fairness* index of about 10% respect to $TCP^{\ddagger}$. This increment in fairness is paid with a reduction of the *aggregate throughput* index of about 3.6%.

When the Adaptive Pacing is used, the optimal configuration of TPA-AP is always *TPA-AP-2*, considering both the *aggregate throughput* and the *Instantaneous fairness* as reference metric. Consequently, table 17.9 only reports the results of *TPA-AP-2*. Table 17.9 shows that when the Adaptive Pacing is used, TPA-AP always outperforms TCP-AP in terms of throughput and percentage of retransmission. In addiction TPA also achieves an higher *instantaneous fairness* index. For example, when 3 connections are simultaneously active in the network, TPA achieves an increment in throughput of about 14% respect to

Table 17.8: Static Random Topology

		Number of connections		
		3	5	10
$TCP^{\dagger}$-uc	A. Thr.	885.1	1126	1303
	I. Fair.	46.8	40.3	28.9
$TPA^{\dagger *}$-3	A. Thr.	827.2	1108	1195
	I. Fair.	56.7	46.5	34
$TCP^{\ddagger}$-2	A. Thr.	831.3	1121	1226
	I. Fair.	52.98	42.6	31.3
$TPA^{\ddagger}$-2	A. Thr.	801	1080	1173
	I. Fair.	58.3	46.9	34.6

Table 17.9: Static Random Topology with the Adaptive Pacing enabled

		Number of connections		
		3	5	10
TCP-AP	A. Thr.	415.9	546.7	558.8
	I. Fair.	71.4	66.7	59
	M. Rtx	7.4	9.5	13.4
TPA-AP-2	A. Thr.	476.6	621.8	641.4
	I. Fair.	75.3	68.9	57.9
	M. Rtx	1.7	2.2	3.5

TCP, while retransmits about 76% less segments. In addiction, TPA obtains an *instantaneous fairness* index of about 5.6% higher respect to TCP.

17.7 Mobile Scenario

The network I simulated consisted of 50 nodes moving over a 1000m x 1000m field. To perform my analysis, I utilized 50 different mobility patterns based on the random waypoint model, each of wich have been generated using the code [107] available for the ns2 simulator presented in [90]. This code produces a perfect sample of the node mobility state, and so permits to start a simulation in steady state. To mimic high node mobility, I used a mean node speed of 5 m/s, and a delta node speed of 4 m/s. This means that nodes sample numeric speed from a uniform distribution on the interval from 1 to 9 m/s. I also used a pause time of 10 seconds. After a warm-up time of 20 seconds, 5 TCP/TPA connections were established over each of which an *ftp* file transfer was conducted for 500 seconds. I calculated my performance metrics as the mean over the considered patterns. To avoid a to high grow of the RTO due to long route recomputation delay, I fixed the *Maximum RTO* parameter for both TCP and TPA to 2 seconds.

In the mobile scenario, TPA obtains optimal performance in the *TPA*-3*

Table 17.10: Mobile Scenario

	Agg. Thr. (kbps)	Fair. (%)	Ist. Fair. (%)	Mean. Rtx. (%)
TCP-uc	864.8	75	38.2	5.8
TCP-2	837.4	77	43.9	4.8
TPA*-3	871.7	78.5	45.3	3.6

Table 17.11: Mobile Scenario with the Adaptive Pacing enabled

	Agg. Thr. (kbps)	Fair. (%)	Ist. Fair. (%)	Mean. Rtx. (%)
TCP-AP	433.2	94	69.8	16
TPA-AP*-3	536.2	91.2	67.1	3.9
TPA-AP-2	523.7	92.8	68.4	3.4

configuration. This in terms of both *aggregate throughput* and *instantaneous fairness*. The optimal TCP configuration, instead, depends on the performance metric chosen as reference metric. When the *aggregate throughput* is chosen as the reference metric, *TCP-uc* is the optimal TCP configuration, while if the *Instantaneous Fairness* is chosen as the reference metric, *TCP-2* is the optimal TCP configuration. Table 17.10 reports the simulative results relative to the optimal configuration of both TCP an TPA in the mobile scenario. It shows that *TPA*-3* obtains an increment in throughput of about 1% respect to *TCP-uc*. In addiction, *TPA*-3* also increments the *instantaneous fairness* index of about 18.6%, and reduces the *retransmission* index of about 38% respect to *TCP-uc*. *TPA*-3* also outperforms *TCP-2*, achieving an increment in throughput of about 4% and retransmitting about 25% less segments. *TPA*-3* also improve the *instantaneous fairness* index of about 3.2%.

I evaluated the impact of the adaptive pacing mechanism also in this scenario. Table 17.11 reports the results relative to TCP-AP, *TPA-AP*-3* (the optimal TPA configuration in terms of throughput), and *TPA-AP-2* (the optimal TPA configuration in terms of *instantaneous fairness*). *TPA-AP*-3* increments TCP-AP throughput of about 23.8%, restransmitting about 75.6% less segments. However, this increment in throughput is paid by TPA with a lower *instantaneous fairness* index. Specifically *TPA-AP*-3* achieves a *instantaneous fairness* index of about 3.9% lower than TCP-AP. *TPA-AP-2*, instead, increments the *aggregate throughput* of about 20.9%, retransmitting about 78.7% less segments respect to TCP-AP. However, in this case, *TPA-AP-2* is less aggressive than *TPA-AP*-3*, and provides a *instantaneous fairness* index only 2% lower than TCP-AP.

17.8 Conclusions

In this Chapter I presented a simulative evaluation of the TPA protocol. Using the ns2 simulator I have investigated TPA performance over simple network topologies, like the cross and the parallel topology, and over more complex network topologies, like the grid topology, a random generated topology, and a dinamic topology (50 nodes moving in a 1000 x 1000 area). In this scenarios TPA outperforms TCP both in terms of aggregate throughput and fairness. Specifically, TPA delivers greater throughput with respect to TCP (up to 6%), while increments *at the same time* the fairness index (up to 35%).

This Chapter also presented some results about unfairness mitigation in TPA. Specifically, I implemented in TPA the Adaptive Pacing mechanism, a popular proposal for improving TCP fairness, and I compared the performance of TCP with Adaptive Pacing and TPA with Adaptive Pacing. To perform my analysis I used the same scenarios used in the previous simulative analysis. In all the scenarios I have investigated TPA is able to improve the performance of TCP. Specifically, with the Adaptive Pacing enabled, TPA is able to increase the TCP throughput up to 39%, granting *at the same time* an increment in fairness up to 5.6%. In addiction, TPA is able to reduce the number of retransmitted segment up to 78.7%.

Chapter 18

Conclusion

This thesis Part described the main specificity of MANETs that condition TCP behaviour and reported the major proposals aimed to improve TCP's performance in such environment. In addiction it also proposed a novel transport protocol for Ad hoc Networks, named TPA, that is specifically tailored to the characteristics of the MANET environment. The TPA protocol provides a reliable, connection-oriented type of service, and includes several innovations with respect to the legacy TCP protocol. Unlike TCP, TPA is able to manage efficiently route failures and route changes that may arise due to nodes' mobility. In addition, the TPA congestion control mechanism is designed by taking into account the real nature of congestion phenomena in MANETs. Finally, TPA implements a novel retransmission policy to reduce the number of useless retransmissions and, hence, energy consumption.

This thesis Part presented an evaluation of TPA protocol in a *real* ad hoc testbed. Specifically, I have investigated the impact of different protocol parameters on the throughput and the number of segments used to sustain the throughput. I have run experiments on different topologies, different routing protocols, and also in mobile scenarios. I have found that TPA throughput is between 5% and 19% greater than the TCP throughput, and, furthermore, TPA retransmits between 64% and 94% less data segments.

This thesis Part also presented a simulative analysis of TPA. Specifically, using the ns-2 simulator, I have analysed TPA performance over simple network topologies, like the cross and the paralles topologies, and over more complex network topologies, like the grid topology and the random topology. Finally, I also analysed TPA performance over a highly dinamic environment, where 50 nodes move over a 1000 x 1000 area. In these scenarious, TPA is able to achieve an increment in throughput up to 6%, and an increment in fairness up to 34% respect to TCP. To conlcude TPA study, this thesis Part also reported some results about unfairness mitigation in TPA. Specifically, I integrated in TPA the Adaptive Pacing mechanism, a popular proposal for improving TCP fairness [46], and I compared the performance of TPA with Adaptive Pacing and TCP with Adaptive Pacing over the previous scenarios. With the adaptive pacing mechanism enabled, TPA is able to increase the TCP throughput up to 39% granting an increment in fairness up to 5.6% respect to TCP. In addiction, TPA is able to reduce the number of retransmitted segment up to 78.7%. Also in this case I used the ns-2 simulator to peform my analysis.

The experiments presented in this thesis Part are tailored to multi-hop ad hoc networks in which groups of users set up a stand-alone network and exchange data in a p2p fashion. The results I have obtained motivate us to further investigate the TPA performance also in different setups. For example, it would be intresting to compare TPA performance with that of different TCP variants, like TCP Sack and TCP Vegas, or with the performance of the various proposal available in literature to improve TCP peformance over MANETs. In addiction, it would be extremely interesting to understand how TPA works in mixed wireless/wired scenarios. In these cases two options could be compared, namely rely on slight TPA modifications to make it work with unmodified TCP endpoints, or using an Indirect-TCP approach, and thus envisioning a first TPA trunk between the wireless node and the gateway to the wired network, and a standard TCP trunk in the wired network. This naturally leads to investigate the viability of TPA in mesh network environments. Another area still to be deeply investigated is how TPA can address fairness issues different and more complex scenarios with respect to those considered in this chapter.

Part IV

Summary and Conclusion

In this thesis I have presented a practical architecture to logically extend traditional wired LANs using multi-hop ad hoc networking technologies. Specifically, using proxy ARP servers and basic properties of the longest-matching rules used by standard IP routing, I implemented a set of mechanisms to interconnect Ad hoc Networks to Internet. The proposed architecture provides ad hoc node self-configuration and both Intranet and Internet connectivity in a way that is transparent to the wired nodes, i.e., without requiring changes in the pre-existing wired LAN. The protocol changes are quite limited and restricted to the gateway nodes.

I have prototyped the proposed architecture to test its functionalities. The shown experimental results indicate that: i) the network performance of Internet access in static configurations can be significantly enhanced (in some cases I have more than doubled the measured throughputs) by properly setting the OLSR protocol parameters such as to improve route stability; and ii) the continuity of TCP sessions during node mobility is achieved without requiring additional overheads.

I believe that there are several related aspects that are worth being further investigated in future work.

- The gateway selection procedure implicitly relies on the ad hoc routing protocol. In the case of OLSR, it is accomplished using shortest-path criteria. However, in a multi-homing scenario, several gateways can exist, which may be implemented using different technologies and may have different capabilities. Thus, there could be many benefits in designing cooperative heuristics to select gateways such as to obtain load balancing within the ad hoc network, or more efficient handovers.

- In this work I have considered basic IP services, i.e., unicast routing and dynamic address allocation. However, more sophisticated functionalities, such as multicast and QoS management, have been developed for the Internet. Therefore, my proposed architecture should be extended to facilitate the integration of these additional capabilities.

- The address allocation scheme described in this thesis allows the exploitation of DHCP servers to assign IP addresses that are topologically correct in the entire extended LAN. However, it is needed a detailed evaluation of the efficiency of my proposal and a comparative study with other auto-configuration schemes. In addition, I intend to explore how to extend my solution to deal with the typical problems that may arise due to node mobility, such as message losses, and network partitions and merging.

In this thesis I have also proposed a novel transport protocol for Ad hoc Networks, named TPA, that is specifically tailored to the characteristics of the MANET environment. This proposal is motivated by the evidence that the TCP protocol exhibits poor performance in a MANET environment. The ultimate reason for this is that MANETs behave in a significantly different way with respect to traditional wired networks, like the Internet, for which the TCP protocol was originally conceived.

The TPA protocol provides a reliable, connection-oriented type of service, and includes several innovations with respect to the legacy TCP protocol. Unlike TCP, TPA is able to manage efficiently route failures and route changes that may arise due to nodes' mobility. In addition, the TPA congestion control mechanism is designed by taking into account the real nature of congestion phenomena in MANETs. Finally, TPA implements a novel retransmission policy to reduce the number of useless retransmissions and, hence, energy consumption.

This thesis presented an evaluation of TPA protocol in a *real* ad hoc testbed. Specifically, I have investigated the impact of different protocol parameters on the throughput and the number of segments used to sustain the throughput. I have run experiments on different topologies, different routing protocols, and also in mobile scenarios. I have found that TPA throughput is between 5% and 19% greater than the TCP throughput, and, furthermore, TPA retransmits between 64% and 94% less data segments.

This thesis also presented a simulative analysis of TPA. Specifically, using the ns-2 simulator, I have analysed TPA performance over simple network topologies, like the cross and the paralles topologies, and over more complex network topologies, like the grid topology and the random topology. Finally, I also analysed TPA performance over a highly dinamic environment, where 50 nodes move over a 1000 x 1000 area. In these scenarious, TPA is able to achieve an increment in throughput up to 6%, and an increment in fairness up to 34% respect to TCP. To conlcude TPA study, this thesis also reported some results about unfairness mitigation in TPA. Specifically, I integrated in TPA the Adaptive Pacing mechanism, a popular proposal for improving TCP fairness [46], and I compared the performance of TPA with Adaptive Pacing and TCP with Adaptive Pacing over the previous scenarios. With the adaptive pacing mechanism enabled, TPA is able to increase the TCP throughput up to 39% granting an increment in fairness up to 5.6% respect to TCP. In addiction, TPA is able to reduce the number of retransmitted segment up to 78.7%. Also in this case I used the ns-2 simulator to peform my analysis.

During my Ph.D. I investigated TPA performance over multi-hop ad hoc networks in which groups of users set up a stand-alone network and exchange data in a p2p fashion. The results I have obtained motivate me to further investigate the TPA performance also in different setups. For example, it would be intresting to compare TPA performance with that of different TCP variants, like TCP Sack and TCP Vegas, or with the performance of the various proposal available in literature to improve TCP peformance over MANETs. In addiction, it would be extremely interesting to understand how TPA works in mixed wireless/wired scenarios, like that I have presented in the first part of this thesis. In this case two options could be compared, namely rely on slight TPA modifications to make it works with unmodified TCP endpoints, or using an Indirect-TCP approach, and thus envisioning a first TPA trunk between the wireless node and the gateway to the wired network, and a standard TCP trunk in the wired network. This naturally leads to investigate the viability of TPA in mesh network environments. Another area still to be deeply investigated is how TPA can address fairness issues in different and more complex scenarios with respect to those considered in this thesis.

Bibliography

[1] AODV-UU, AODV Linux Implementation, University of Uppsala.

[2] Ieee. standard 802.11-1999, part 11: Wireless lan medium access control (mac) and physical layer (phy) specications. *In The Institute of Electrical and Electronics Engineers.* 1999.

[3] The Network Simulator - ns-2 (version 2.30).

[4] Local and Metropolitan Area Network – Specific Requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *The Institute of Electrical and Electronics Engineer*, (802.11), aug 1999. Piscataway, NJ.

[5] Wireless LAN Medium Access Control (MAC)and Physical Layer (PHY) Specification/Amendment 2: Higher-speed Physical Layer (PHY) in the 2.4 GHz band. *The Institute of Electrical and Electronics Engineer*, (802.11b), November 2001. Piscataway, NJ.

[6] A. Acharya, A. Misra, and S. Bansal. A Label-switching Packet Forwarding Architecture for Multi-hop Wireless LANs. In *Proc. of ACM WoWMoM 2002*, pages 33–40, Atlanta, Georgia, USA, September, 28 2002.

[7] C. Ahlund and A. Zaslavsky. Integration of Ad Hoc Network and IP Network Capabilities for Mobile Hosts. In *Proc. of ICT 2003*, volume 3, pages 482–489. IEEE Computer Society Press., Tahiti, February 23 – March 1 2003.

[8] A. Ahuja, S. Agarwal, J. Sing, and R. Shorey. Performance of TCP over Different Routing Protocols in Mobile Ad Hoc Networks. In *Proceedings of the IEEE Vehicular Technology Conference (VTC 2000)*, pages 2315–2319. IEEE Computer Society Press, May 2000.

[9] M. Allman, H. Balakrishman, and S. Floyd. Enhancing tcps loss recovery using limited transmit. RFC 3042, IETF Network Working Group, January 2001.

[10] M. Allman, V. Paxson, and W. Stevens. Tcp congestion control. RFC 2581, April 1999.

[11] E. Altman and T. Jimenez. Novel Delayed ACK Techniques for improving TCP Performance in Multi-hop Wireless Networks. In *Proceedings of the IFIP International Conference on Personal Wireless Communications (PWC 2003)*, number 2775, pages 237–250, Venice, Italy, September 23–25 2003. LNCS.

[12] V. Anantharaman, S.-J. Park, K. Sundaresan, and R. Sivakumar. TCP Performance over Mobile Ad-hoc Networks: A Quantitative Study. *Wireless Communications and Mobile Computing Journal (WCMC)*, 4(2):203–222, 2004. Special Issue on Performance Evaluation of Wireless Networks.

[13] G. Anastasi, E. Ancillotti, and A. Passarella. Tpa: A transport protocol for ad hoc networks. In *ISCC '05: Proceedings of the 10th IEEE Symposium on Computers and Communications (ISCC'05)*, pages 51–56, Washington, DC, USA, 2005. IEEE Computer Society.

[14] G. Anastasi, E. Borgia, M. Conti, and E. Gregori. Wi-Fi in Ad Hoc Mode: A Measurement Study. In *Proc. of IEEE PerCom 2004*, pages 145–154, Orlando, FL, March, 14–17 2004.

[15] G. Anastasi, E. Borgia, M. Conti, E. Gregori, and A. Passarella. Understanding the Real Behavior of Mote and 802.11 Ad hoc Networks: an Experimental Approach. *Pervasive and Mobile Computing*, 1:237–256, July 2005. Special Issue on Performance Evaluation of Wireless Networks.

[16] Giuseppe Anastasi, Emilio Ancillotti, Marco Conti, and Andrea Passarella. Experimental analysis of a transport protocol for ad hoc networks (tpa). In *PE-WASUN '06: Proceedings of the 3rd ACM international workshop on Performance evaluation of wireless ad hoc, sensor and ubiquitous networks*, pages 9–16, New York, NY, USA, 2006. ACM Press.

[17] E. Ancillotti, G. Anastasi, M. Conti, and A. Passarella. A comprehensive study of tpa: a transport protocol for ad hoc networks. *submitted to The Computer Journal.* Special Issue on Performance Evaluation of Wireless Networks.

[18] E. Ancillotti, G. Anastasi, M. Conti, and A. Passarella. *Design, Implementation and Measurements of a Transport Protocol for Ad Hoc Networks.* chapter in MobileMAN (M. Conti, Editor), Sprinter. to appear.

[19] E. Ancillotti, G. Anastasi, M. Conti, and A. Passarella. *Experimental Analysis of TCP Performance in Static Multi-hop Ad Hoc Networks.* chapter 6 in Mobile Ad Hoc Networks: from Theory to Reality, Nova Science Publisher. (M. Conti, J. Crowcroft, and A. Passarella, Editors).

[20] E. Ancillotti, R. Bruno, M. Conti, E. Gregori, and A. Pinizzotto. *A MobileMAN Approach for the Interconnection of Heterogeneous Ad Hoc Networks to the Internet.* chapter in MobileMAN (M. Conti, Editor), Springer. to appear.

[21] E. Ancillotti, R. Bruno, M. Conti, E. Gregori, and A. Pinizzotto. *Implementation and Experimentation of a Layer-2 Architecture for Interconnecting Heterogeneous Ad Hoc Network to the Internet.* chapter in Mobile Ad Hoc Networks: from Theory to Reality, (M. Conti, J. Crowcroft, A. Passarella, Editors), Nova Science Publisher. to appear.

[22] E. Ancillotti, R. Bruno, M. Conti, E. Gregori, and A. Pinizzotto. Experimenting a Layer 2-based Approach to Internet Connectivity for Ad Hoc Networks. In *IEEE ICPS Workshop on Multi-hop Ad hoc Networks (REALMAN 2005)*, Santorini (Greece), July 14 2005.

[23] E. Ancillotti, R. Bruno, M. Conti, E. Gregori, and A. Pinizzotto. A Layer-2 Architecture for Interconnecting Multi-hop Hybrid Ad Hoc Networks to the Internet. In *in Proceedings of WONS 2006*, pages 87–96, Les Menuires, France, January, 18–20 2006.

[24] E. Ancillotti, R. Bruno, M. Conti, E. Gregori, and A. Pinizzotto. A layer-2 framework for interconnecting ad hoc networks to fixed internet: Test-bed implementation and experimental evaluation. *submitted to The Computer Journal*, 2007.

[25] E.M. Belding-Royer and C.-K. Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communication Magazine*, pages 46–55, April 1999.

[26] M. Benzaid, P. Minet, K. Al Agha, C. Adjih, and G. Allard. Integration of Mobile-IP and OLSR for a Universal Mobility. *Wireless Networks*, 10(4):377–388, July 2004.

[27] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, S. Floyd D. Estrin, V. Jacobson, G. Minshall, G. Minshal, C. Partridge, K. Ramakrishnan L. Peterson, S. Shenker, J. Wroclawski, and L. Z. Recommendations on queue management and congestion avoidance in the internet. RFC 2309, IETF Network Working Group, April 1998.

[28] R. Braden. equirements for internet hosts – communication layers. RFC 1122, October 1989.

[29] L. Brakmo and L. Peterson. Tcp vegas: End to end congestion avoidance on a global internet. *IEEE Journal on Selected Areas in Communication*, 13(8):1465–1480, October 1995.

[30] R. Brannstrom, C. Ahlund, and A Zaslavsky. Maintaining Gateway Connectivity in Multi-hop Ad hoc Networks. In *Proc. of IEEE LCN 2005*, pages 682–689, Sydney, Australia, November 15–17 2005.

[31] J. Broch, D.A. Maltz, and D.B. Johnson. Supporting hierarchy and heterogenous interfaces in multi-hop wireless ad hoc networks. In *Proc. of I-SPAN'99*, pages 370–375, Perth, Australia, June, 23–25 1999.

[32] S. Carl-Mitchell and J.S. Quarterman. Using ARP to Implement Transparent Subnet Gateways. RFC 1027, October 1987.

[33] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash. A Feedback Based Scheme for Improving TCP Performance in Ad Hoc Wireless Networks. *IEEE Personal Communication Magazine*, 8(1):34–39, February 2001. Special Issue on Ad Hoc Networks.

[34] K. Chen, Y. Xue, S. Shah, and K. Nahrstedt. Understanding Bandwidth-Delay Product in Mobile Ad Hoc Networks. *Computer Communications*, 27:923–934, July 2004. Special Issue on Performance Evaluation of Wireless Networks.

[35] D. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17:114, June 1989.

[36] J. P. Clark. Window and acknowledgment strategy in tcp. RFC 813, IETF Network Working Group, July 1982.

[37] T. Clausen and P. Jaquet. Optimized Link State Routing Protocol (OLSR). RFC 3626, October 2003.

[38] M. Conti, E. Gregori, and G. Maselli. Improving the performability of data transfer in mobile ad hoc networks. In *IEEE Sensor and Ad Hoc Communications and Networks (SECON 2005)*, pages 153–163, Sept. 26–29 2005.

[39] C. D. A. Cordeiro, S. R. Das, and D. P. Agrawal. Copas: dynamic contention-balancing to enhance the performance of tcp over multi-hop wireless networks. *Computer Communications and Networks, 2002. In proceedings of IC3N'02*, pages 382–387, 2002.

[40] M. Crovella and A. Bestavros. Self Similarity in World Wide Web Trafc: Evidence and Possible Causes. *IEEE/ACM Transactions on Networking*, 5:835–846, 1997.

[41] Darpa. Transmission control protocol. ietf network working group,. RFC 793, April 1981.

[42] R. de Oliveira and T. Braun. A Dynamic Adaptive Acknowledgment Strategy for TCP over Multi-hop Wireless Networks. In *Proceedings of IEEE Infocom 2005*, volume 3, pages 1863–1874, Miami, USA, March 2005. IEEE Computer Society Press.

[43] R. Draves, J. Padhye, and B. Zill. The architecture of the Link Quality Source Routing Protocol. Technical Report MSR-TR-2004-57, Microsoft Research, 2004.

[44] R. Droms. Dynamic Host Configuration Protocol. RFC 2131, March 1997.

[45] T.D. Dyer and R.V. Boppana. A Comparison of TCP Performance over Three Routing Protocols for Mobile Ad Hoc Networks. In *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc)*. ACM Press, October 2001.

[46] S. M. ElRakabawy, A.Klemm, and C. Lindemann. TCP with Adaptive Pacing for Multihop Wireless Networks. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 288–299, Urbana-Champaign, IL, USA, May 2005. ACM Press.

[47] P.E. Engelstad and G. Egeland. NAT-based Internet Connectivity for On Demand MANETs. In *Proc. of WONS 2004*, pages 4050–4056, Madonna di Campiglio, Italy, January, 18–23 2004.

[48] P.E. Engelstad, A. Tønnesen, A. Hafslund, and G. Egeland. Internet Connectivity for Multi-Homed Proactive Ad Hoc Networks. In *Proc. of IEEE ICC'2004*, volume 7, pages 4050–4056, Paris, France, June, 20–24 2004.

[49] Z. Fan. IPv6 stateless address autoconguration in ad hoc networks. In *Proc. of PWC03*, pages 665–678, Venice, Italy, September, 23–25 2003. Springer-Verlag. Lecture Notes in Computer Science.

[50] S. Floyd and T. Henderson. The newreno modication to tcps fast recovery algorithm. RFC 2582, IETF Network Working Group, April 1999.

[51] S. Floyd, T. Henderson, and A. Gurtov. The newreno modication to tcps fast recovery algorithm. RFC 3782, IETF Network Working Group, April 2004.

[52] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, August 1993.

[53] 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 the IEEE International Conference on Network Protocols (ICNP 2000)*, Paris, France, November 2002. IEEE Computer Society Press.

[54] Z. Fu, X. Meng, and S. Lu. How Bad TCP Can Perform in Mobile Ad Hoc Networks. In *Proceedings of the IEEE Symposium on Computers and Communications (ISCC 2002)*, pages 298–303, Taormina-Giardini Naxos (Italy), July 2002. IEEE Computer Society Press.

[55] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla. The Impact of Multi-hop Wireless Channel on TCP Throughput and Loss. In *Proceedings of IEEE INFOCOM 2003*, San Francisco (California), March 30–April 3 2003. IEEE Computer Society Press.

[56] T. Goff, N. B. Abu-Ghazaleh, D. S. Phatak, and R. Kahvecioglu. Preemptive routing in ad hoc networks. In *Proc. ACM MOBICOM*, pages 43–52, Rome, Italy, 2001.

[57] A. Gupta, I. Wormsbecker, and C. Williamson. Experimental evaluation of tcp performance in multi-hop wireless ad hoc networks. In *Proceedings of IEEE/ACM MASCOTS*, pages 3–11, Volendam, Netherlands, October 2004.

[58] 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.

[59] Ahmad A. Hanbali, E. Altman, and P. Nain. A survey of tcp over ad hoc networks. *Communications Surveys & Tutorials, IEEE*, 7(3):22–36, 2005.

[60] Qi He, L. Cai, X. Shen, and P. Ho. Improving TCP Performance over Wireless Ad Hoc Networks with Busy Tone Assisted Scheme. *EURASIP Journal on Wireless Communications and Networking*, 2006. Article ID 51610, 11 pages.

[61] G. Holland and N. Vaidya. Impact of routing and link layers on tcp performance in mobile ad hoc networks. In *In Proceedings of ACM/IEEE Wireless Communication Networks Conference (IEEE WCNC 1999)*, New Orleans, USA, September 1999.

[62] G. Holland and N. Vaidya. Analysis of TCP Performance over Mobile Ad Hoc Networks. *Wireless Networks*, 8:275–288, 2002.

[63] V. Jacobson. Congestion avoidance and control. In *In Proceedings of ACM SIGCOMM*, pages 314–329, Stanford, CA, August 1988.

[64] V. Jacobson, R. Braden, and D. Borman. Tcp extensions for high performance. RFC 1323, IETF Network Working Group, May 1992.

[65] R. Jain, D. Chiu, and W. Hawe. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Systems. DEC Technical Report DEC-TR-301, 1984.

[66] C. Jelger, T. Noel, and A. Frey. Gateway and Address Autoconguration for IPv6 Ad Hoc Networks. Internet Draft, 2004.

[67] L. B. Jiang and S. C. Liew. Proportional fairness in wireless LANs and ad hoc networks. In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 3, pages 1551–1556, Long Beach, CA, USA, 13-17 March 2005. IEEE Computer Society Press.

[68] D.B. Johnson, D.A. Maltz, and Y.-C. Hu. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR). Internet Draft, July 19 2004.

[69] U. Jönsson, F. Alriksson, T. Larsson, P. Johansson, and G.Q. Maguire Jr. MIPMANET - Mobile IP for Mobile Ad Hoc Networks. In *Proc. of MobiHoc 2000*, pages 75–85, Boston, MA, USA, August, 11 2000.

[70] P. Karn and C. Partridge. Improving round-trip time estimates in reliable transport protocols. In *Computer Communication Review*, volume 17, pages 2–7, August 1987.

[71] V. Kawadia and P. Kumar. Experimental investigation into tcp performance over wireless multihop networks. In *Proc. of ACM SigCom 2005 Workshops*, Philadelphia (PA), August 22–25 2005.

[72] D. Kim, C. Toh, and Y. Choi. Tcp-bus: Improving tcp performance in wireless ad hoc networks. *J. Commun. and Net.*, 3(2):17586, June 2001.

[73] Fabius Klemm, Zhenqiang Ye, Srikanth V. Krishnamurthy, and Satish K. Tripathi. Improving tcp performance in ad hoc networks using signal strength based link management. *Ad Hoc Networks*, 3(2):175–191, 2005.

[74] S. Kopparty, S. V. Krishnamurthy, M. Faloutsos, and S. K. Tripathi. Split tcp for mobile ad hoc networks. In *Proc. IEEE GLOBECOM*, Taipei, Taiwan, Nov. 2002.

[75] S. J. Lee and M. Gerla. Split multipath routing with maximally disjoint paths in ad hoc networks. In *Proceedings of IEEE ICC01*, June 2001.

[76] J. Li, C. Blake, D. De Couto, H. Lee, and R. Morris. Capacity of Ad Hoc Wireless Networks. In *Proc. ACM/IEEE International Conference in Mobile Computing and Networking (MobiCom 2001)*, Rome, Italy, July 2001. IEEE Computer Society Press.

[77] D. Libes. Implementing software timers. *C Users Journal*, November 1990.

[78] H. Lim, K. Xu, and M. Gerla. Tcp performance over multipath routing in mobile ad hoc networks. In *in Proceedings of the IEEE International Conference on Communications (ICC)*, May 2003.

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

[80] H. Lundgren, E. Nordstrom, and C. Tschudin. Coping with Communication Gray Zones in IEEE 802.11b based Ad hoc Networks. In *Proceedings of the 5th ACM international workshop on Wireless mobile multimedia*, pages 49–55, Atlanta, Georgia, 2002. ACM Press.

[81] M. Marina and S. Das. On-demand multipath distance vector routing in ad hoc networks. In *Proceedings of IEEE International Conference on Network Protocols (ICNP)01*, Nov. 2001.

[82] M. Mathi, J. Mahdavi, and S. F. A. Romanow. Tcp selective acknowledgement options. RFC 2018, IETF Network Working Group, October 1996.

[83] J. Monks, P. Sinha, and V. Bharghavan. Limitations of tcp-elfn for ad hoc networks. In *Proc. Mobile and Multimedia Commun.*, Tokyo, Japan, October 2000.

[84] G. Montenegro. Reverse Tunneling for Mobile IP. RFC 2344, May 1998.

[85] K. Nahm, A. Helmy, and C.-C.Jay Kuo. TCP over Multi-hop 802.11 Networks: issues and Performance Enhancement. In *Proceedings of ACM MobiHoc*, pages 277–287, Urbana-Champaign, IL, USA, June 2005. ACM Press.

[86] S. Nesargi and R. Prakash. MANETconf: Conguration of Hosts in a Mobile Ad Hoc NEtwork. In *Proc. of INFOCOM 2002*, pages 1059–1068, New York, NY, June, 23–27 2002. IEEE Computer Society Press.

[87] Ping Chung Ng and Soung Chang Liew. Re-routing instability in ieee 802.11 multi-hop ad-hoc networks. *Ad Hoc and Sensor Wireless Networks*, 1:01–25. 2005.

[88] Ni, S. and Tseng, Y. and Chen, Y. and Sheu, J. The Broadcast Storm Problem in a Mobile Ad Hoc Network, 2002.

[89] R. Ogier, F. Templin, and M. Lewis. Topology Dissemination Based on Reverse-Path Forwarding (TBRPF). RFC 3684, February 2004.

[90] Santashil PalChaudhuri, Jean-Yves Le Boudec, and Milan Vojnovic. Perfect simulations for random trip mobility models. In *ANSS '05: Proceedings of the 38th annual Symposium on Simulation*, pages 72–79, Washington, DC, USA, 2005. IEEE Computer Society.

[91] S. Papanastasiou, L. M. Mackenzie, M. Ould-Khaoua, and V. Charissis. On the interaction of tcp and routing protocols in manets. In *in proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006)*, pages 62–69, February 19–22 2006.

[92] S. Papanastasiou and M. Ould-Khaoua. TCP Congestion Window Evolution and Spatial Reuse in MANETs. *Journal of Wireless Communications and Mobile Computing*, 4(6):669–682, Sept. 2004.

[93] S. Papanastasiou, M. Ould-Khaoua, and L. MacKenzie. *Handbook of Algorithms and Wireless Networking and Mobile Computing*, chapter TCP Developments in Mobile Ad Hoc Networks.

[94] V. Park and S. Corson. Temporally-ordered routing algorithm (tora). Internet Draft, draft-ietf-manet-tora-spec- 03.txt, June 2001. work in progress.

[95] V.D. Park and M.S. Corson. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In *Proceedings of IEEE INFOCOM '97*, Kobe (Japan), 1997. IEEE Computer Society Press.

[96] V. Paxson and M. Allman. Computing tcp's retransmission timer. RFC 2988, November 2000.

[97] C. Perkins. IP Encapsulation within IP. RFC 2003, October 1996.

[98] C. Perkins. Minimal Encapsulation within IP. RFC 2004, October 1996.

[99] C. Perkins. IP Mobility Support for IPv4. RFC 3344, August 2002.

[100] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561, July 2003.

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

[102] C.E. Perkins. *Mobile IP Design Principles and Practice.* January 1998.

[103] D.C. Plummer. An Ethernet Address Resolution Protocol. RFC 826, November 1982.

[104] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, September 2001.

[105] P. Ratanchandani and R. Kravets. A Hybrid Approach to Internet Connectivity for Mobile Ad hoc Networks. In *Proc. of IEEE WCNC 2003*, volume 3, pages 1522–1527. IEEE Computer Society Press., New Orleans, USA, March, 16–20 2003.

[106] Ros F. Ruiz, P. and A. Gomez-Skarmeta. Internet connectivity for mobile ad hoc networks: Solutions and challenges. *IEEE Communication Magazine*, 43.

[107] Rice University. S. PalChaudhuri. ns-2 Code for Random Trip Mobility Model.

[108] N. Spring, D. Wetherall, and D. Ely. Robust Explicit Congestion Notification (ECN) Signaling with Nonces. RFC 3540, June 2003.

[109] P. Srisuresh and M. Holdrege. IP Network Address Translator (NAT) Terminology and Considerations. RFC 2663, August 1999.

[110] W. Stevens. Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. RFC 2001, IETF Network Working Group, January 1997.

[111] W.R. Stevens. TCP/IP Illustrated. volume 1. Addison Wesley, 1994.

[112] W.R. Stevens. *UNIX Network Programming Volume 2, Interprocess Communications*. Prentice Hall PTR, 2nd edition edition, 1999.

[113] W.R. Stevens. *Advanced Programming in the UNIX Environment*. Pearson Education, 2nd edition edition, 2005.

[114] D. Sun and H. Man. ENIC - An Improved Reliable Transport Scheme for Mobile Ad Hoc Networks. In *IEEE Globecom Conference*, volume 5, pages 2852–2856, (San Antonio, TX), November 2001. IEEE Computer Society Press.

[115] Y. Sun, E.M. Belding-Royer, and C.E. Perkins. Internet Connectivity for Ad hoc Mobile Networks. *International Journal of Wireless Information Networks*, 9(2):75–88, April 2002. Special issue on "Mobile Ad hoc Networks: Standards, Research, Application".

[116] K. Sundaresan, V. Anantharaman. H. Hsieh, and R. Sivakumar. Atp: A reliable transport protocol for ad hoc networks. *IEEE transactions on mobile computing*, 4(6):588–603, November-December 2005.

[117] K. Tang and M. Gerla. Fair sharing of mac under tcp in wireless ad hoc networks. In *Proceedings of IEEE MMT'99*, Venice, Italy, Oct. 1999.

[118] C.A. Thekkath, T.D. Nguyen, E. Moy, and E.D. Lazowska. Implementing network protocols at user level. *IEEE/ACM Transactions on Networking*, 1(5):554–565, October 1993.

[119] S. Thomson and T. Narten. IPv6 stateless address autoconguration. RFC 2462, 1998.

[120] A. Tønnesen. Implementation of the OLSR specification (OLSR_UniK). Version 0.4.8, December 2004.

[121] C. Tschuding, R. Gold, O. Rensfelt, and O. Wibling. LUNAR: a Lightweight Underlay Network Ad-hoc Routing Protocol and Implementation. In *Proc. of NEW2AN'04*, St. Petersburg, Russia, February, 2–6 2004.

[122] N. Vaidya. Weak Duplicate Address Detection in Mobile Ad Hoc Networks. In *Proc. of ACM MobiHoc 2002*, pages 206–216, Lausanne, Switzerland, June, 9–11 2002. ACM Press.

[123] S. Vutukury and J. J. Garcia-Luna-Aceves. Mdva: A distance-vector multipath routing protocol. In *Proceedings of IEEE INFOCOM01*, Apr.

[124] R. Wakikawa, J. Malinen, C. Perkins, A. Nilsson, and A. Tuominen. Global Connectvity for IPv6 Mobile Ad Hoc Networks. Internet Draft, 2006.

[125] F. Wang and Y. Zhang. Improving tcp performance over mobile ad hoc networks with out-of-order detection and response. In *Proc. ACM MOBIHOC*, pages 217–25, Lausanne, Switzerland, June 2002.

[126] K. Weniger and M. Zitterbart. Address Autoconfiguration on Mobile Ad Hoc Networks: Current Approaches and Future Directions. *IEEE Network*, 18(4):6–11, July/August 2004.

[127] K. Xu and M. Gerla. Tcp unfairness in ad hoc wireless networks and a neighborhood red solution. *Wireless Networks*, 11(4):383–399, 2005.

[128] K. Xu, M. Gerla, and S. Bae. Effectiveness of rts/cts handshake in ieee 802.11 based ad hoc networks. *Ad Hoc Networks Journal*, 1(1):107–123, July 2003.

[129] Kaixin Xu, Sang Bae, Sungwook Lee, and Mario Gerla. Tcp behavior across multihop wireless networks and the wired internet. In *WOWMOM '02: Proceedings of the 5th ACM international workshop on Wireless mobile multimedia*, pages 41–48, New York, NY, USA, 2002. ACM Press.

[130] S. Xu and T. Saadawi. Does the ieee 802.11 mac protocol work well in multihop wireless ad hoc networks? *IEEE Communications Magazine*, June 2001.

[131] S. Xu and T. Saadawi. Performance evaluation of tcp algorithms in multi-hop wireless packet networks. *Wireless Communications and Mobile Computing*, 2(1):85–100, 2001.

[132] S. Xu and T. Saadawi. Revealing the problems with 802.11 medium access control protocol in multi-hop wireless ad hoc networks. *Computer Networks*, 38:531–548, March 2002.

[133] Luqing Yang, Winston K.G. Seah, and Qinghe Yin. Improving fairness among tcp flows crossing wireless ad hoc and wired networks. In *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 57–63, New York, NY, USA, 2003. ACM Press.

[134] L. Zhang, Z. Zhao, Y. Shu, L. Wang, and O. W. Yang. Load balancing of multipath source routing in ad hoc networks. In *Proceedings of IEEE ICC02*, Apr. 2002.

List of Figures

List of Tables