



**João Paulo Silva
Barraca**

**Mecanismos de Facturação Segura em Redes
Auto-Organizadas**



**João Paulo Silva
Barraca**

**Mecanismos de Facturação Segura em Redes
Auto-Organizadas**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Electrónica e Telecomunicações, realizada sob a orientação científica do Doutor Rui L. Aguiar, Professor do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

dedicatória

Para a Rosa.

Para os meus pais sem o esforço dos quais esta dissertação não seria possível.

o júri

presidente

Prof. Dr. Nuno Borges Carvalho

Professor Associado da Universidade de Aveiro

Prof. Dr. Rui Luis Andrade Aguiar

Professor Auxiliar da Universidade de Aveiro

Prof. Dr. Manuel Alberto Pereira Ricardo

Professor Associado da Faculdade de Engenharia da Universidade do Porto

agradecimentos

Os sinceros agradecimentos a todos os colegas do Instituto de Telecomunicações - Aveiro, em especial para as pessoas do HNG.

Palavras Chave

Cooperação, MANET, Taxação, Segurança, Motivação, Incentivos

Resumo

As redes ad-hoc e as redes auto-organizadas constituem uma área de investigação com grande interesse. Estas redes são uteis em cenários onde seja necessária uma rede de baixo custo, elevada adaptabilidade e reduzido tempo de criação. As redes infra-estruturadas, tendo uma gestão centralizada, estão agora a começar a adoptar os conceitos de redes auto-organizadas nas suas arquitecturas. Ao contrário dos sistemas centralizados, redes auto-organizadas requerem que todos os terminais participantes operem de acordo com o melhor interesse da rede. O facto de, em redes ad-hoc, os equipamentos possuírem recursos limitados, põe em causa este requisito levando a comportamentos egoístas. Este comportamento é espectável criando problemas nas redes auto-organizativas, ameaçando o funcionamento de uma rede inteira. Algumas propostas foram já criadas de modo a motivar a sua utilização correcta. Destas, algumas são baseadas em trocas de crédito entre utilizadores, outras prevêem a existência de entidades gestoras de créditos. Estas últimas propostas, que irão ser o foco desta dissertação, permitem a fácil integração de redes ad-hoc com redes infra-estruturadas e geridas por um operador. Este trabalho descreve o estado da arte actual e, com algum detalhe, os métodos utilizados e as soluções relevantes para esta área. São propostas duas novas soluções de taxaço para estas redes. Ambas as soluções possibilitam a integraço das redes com métodos de taxaço habituais em redes geridas por operadores. Para além disto, a motivaço à participaço é aumentada através de incentivos ao encaminhamento de pacotes. Todos os processos são criptograficamente seguros através da utilizaço de métodos standard como DSA sobre Curvas Elípticas e funções de síntese robustas. As soluções propostas são descritas analiticamente e analisadas, sendo os resultados obtidos comparados com outra proposta do estado da arte. Um exaustivo trabalho de simulaço é igualmente descrito de forma a avaliar as soluções em cenários mais complexos. Os resultados obtidos em simulaço são avaliados tendo em conta a variaço de várias métricas como mobilidade, carga na rede, protocolo de encaminhamento e protocolo de transporte. No final, a arquitectura, implementaço e resultados obtidos com uma implementaço real de uma das propostas e os seus resultados analisados.

Keywords

Cooperation, MANET, Charging, Security, Motivation, Incentives

Abstract

Self-organised and ad-hoc networks are an area with an existing large research community. These networks are much useful in scenarios requiring a rapidly deployed, low cost and highly adaptable network. Recently, infrastructure networks, which are managed in a much centralised form, are starting to introduce concepts of self-organised networks in its architecture. In opposition to centralised systems, self-organisation creates the necessity for all nodes to behave according to the best interest of the network. The fact that in many ad-hoc networks nodes have scarce resources poses some threats to this requirement. As resources decreases, such as battery or wireless bandwidth, nodes can start acting selfishly. This behaviour is known to bring damage to self-organised networks and threatens the entire network. Several proposals were made in order to promote the correct usage of the network. Some proposals are based on local information and direct credit exchange while others envision the existence of a central bank. The later solutions are further elaborated in this thesis, as they make possible integration of ad-hoc network with operator driven infrastructures. This work presents the current state-of-the-art on the area providing a detailed insight on the methods adopted by each solution presented. Two novel solutions are proposed providing charging support for integrated ad-hoc networks. Both solutions provide means of integration with standard management methods found in operator networks. Also, nodes' motivation is increased through the reward of nodes forwarding data packets. The entire process is cryptographically secure, making use of standard methods such as Elliptic Curve DSA and strong digest functions. The solutions proposed are described and analysed analytically, comparing the results with other state-of-the-art proposals. Extensive simulation work is also presented which furthers evaluates the solutions in complex scenarios. Results are obtained from these scenarios and several metrics are evaluated taking in consideration mobility, network load, routing protocol and transport protocol. The architecture and results obtained with a real implementation are finally presented and analysed.

Contents

1	Introduction	1
1.1	Self-organised Ad-hoc Networks	1
1.2	Rewarding Concepts	5
1.3	Objectives	6
1.4	Contributions	6
1.5	Disposition	7
2	Solutions for ad-hoc networks	9
2.1	Routing	9
2.1.1	Pro-active routing protocols	10
2.1.2	Reactive Routing Protocols	14
2.2	Address Auto-configuration	18
2.3	Quality of Service	21
2.3.1	QOLSR	22
2.3.2	INSIGNIA	23
2.3.3	SWAN	24
3	Cooperation in Ad-hoc Networks	27
3.1	Cooperation by punishment or reputation	30
3.1.1	CONFIDANT	30
3.1.2	Context Aware Detection	31
3.1.3	OCEAN	32
3.2	Cooperation by Credit Incentives	33
3.2.1	CASHNet	33
3.2.2	A Charging and Rewarding Scheme for Packet Forwarding in Multi-hop Cellular Networks	34
3.2.3	SPRITE	37
3.2.4	SCP	39

4	Proposed Solutions	45
4.1	PACP - Polynomial-assisted Ad-hoc Charging Protocol	45
4.2	SACP - Session Aware ad-hoc Charging Protocol	56
4.3	Overhead analysis	60
4.3.1	SCP	61
4.3.2	PACP	64
4.3.3	SACP	66
4.3.4	Overhead comparison	68
4.4	Obtained results	74
4.4.1	Simulation Modules	74
4.4.2	Simulation Environment	77
4.4.3	Network Performance	78
4.4.4	Control Overhead	83
4.4.5	Charging Rate	90
4.4.6	Rewarding Error	94
5	PACP implementation evaluation	99
5.1	Implementation details	99
5.1.1	Netfilter	100
5.1.2	Implementation modules	102
5.2	Ad-hoc testbed scenario	108
5.3	Delay	110
5.3.1	Registration	110
5.3.2	Session establishment	111
5.3.3	End to end delay	112
5.4	Overhead	115
5.5	Maximum throughput	117
5.6	Charging process	119
6	Conclusions	123

List of Figures

1.1	Scalability vs determinism in different types of networks [dressler06]	2
2.1	Flooding using Multi Point Relaying	12
2.2	SWAN Internal structure	24
3.1	Control signalling of a session establishment	35
3.2	Diagram representing the different phases of the SCP proposal	40
4.1	Ad-hoc integrated network	46
4.2	Different phases and messages of the PACP proposal	48
4.3	IPv6 packet with a PACP charging header	50
4.4	Variation of the number of proofs reported by SCP in relation to the size of the ECDSA key; assuming a MTU of 1500, MAC_s of 0 and $nhops_r$ lower than 8.	63
4.5	Variation of the number of proofs reported by PACP in relation to the size of the ECDSA key; assuming a MTU of 1500, MAC_s of 0 and $nhops_r$ lower than 8.	65
4.6	Scenario 1 - A typical hotspot with no mobility or multi-hop forwarding	69
4.7	Scenario 2 - An ad-hoc extended hotspot with no mobility but with multi-hop capabilities	69
4.8	Scenario 3 - An ad-hoc extended hotspot with mobility and multi-hop capabilities	69
4.9	Variation of marking overhead with the increase on number of hops	72
4.10	Variation of marking overhead with the increase of mobility. SACP line has fluctuations due to padding.	73
4.11	Diagram of an wireless node in NS-2 and the NSTAP module	74
4.12	Diagram of a DSR wireless node in NS-2 and the NSTAP module	76
4.13	Evolution of the average number of hops in reported proofs. Using (a) 10 TCP and (b) 10 UDP flows	80

4.14	Variation of the performance ratio of (a) TCP and (b) UDP flows with the increase of mobility.	81
4.15	Performance ratio of the different proposals over AODV and DSR using (a) TCP and (b) UDP flows	82
4.16	Performance ratio of the different proposals using (a) TCP and (b) UDP flows under various values of network load	83
4.17	Inband overhead of the different proposals using (a) TCP and (b) UDP flows with increasing mobility	84
4.18	Out of band overhead of the different proposals using (a) TCP and (b) UDP flows with increasing mobility	85
4.19	Total overhead of the different proposals using (a) TCP and (b) UDP flows with increasing mobility	86
4.20	Inband overhead of the different proposals using (a) TCP and (b) UDP using AODV and DSR	87
4.21	Total overhead of the different proposals using (a) TCP and (b) UDP using AODV and DSR	88
4.22	Inband overhead of the different proposals using (a) TCP and (b) UDP while increasing the load in the network	88
4.23	Out of band overhead of the different proposals using (a) TCP and (b) UDP while increasing the load in the network	89
4.24	Total overhead of the different proposals using (a) TCP and (b) UDP while increasing the load in the network	90
4.25	Efficiency of the charging process with increasing mobility. Sub-figure (a) relates to TCP flows charged to the receiver, while (b) relates to UDP flows charged to the sender.	93
4.26	Variation of error in charging hosts both for (a) TCP and (b) UDP with increasing load. Sub-figures (a) and (b) relate to flows charged according to the receiver, while (c) and (d) relate to flows charged according to the sender.	95
4.27	Variation of error in rewarding hosts both for (a) TCP and (b) UDP with increasing mobility.	96
4.28	Variation of error in rewarding hosts both for (a) TCP and (b) UDP in relation to network load	98
5.1	Internal structure of Daidalos ad-hoc nodes and ad-hoc Access Routers	100
5.2	Netfilter hooks, tables and chains	101

5.3	Class structure of an Ad-hoc Node (using ChargingAgent) or Access Router (using InternetworkAdapter)	102
5.4	Class structure of the Charging Manager	103
5.5	Representation of the integrated ad-hoc network testbed.	109
5.6	Map of IT-Aveiro and the location of the ad-hoc testbed. Nodes were distributed inside that room.	109
5.7	Packet capture showing the 6 nodes registering with the Charging Manager. Times are indicated as delay since last packet and not absolute. .	111
5.8	Packet capture showing the 6 nodes asking for permission to forward a flow	111
5.9	256Kbit traffic from Node2 to the Access Router. Node1 is forwarding and issues a renewal at t=100s.	113
5.10	Time consumed when decoding proofs 10000 times for various route lengths.	122

List of Tables

4.1	Symbols used on the equations of current section	61
4.2	Values used to calculate overhead in the different scenarios	70
4.3	Expected overhead on Scenario 1	71
4.4	Expected overhead on Scenario 2	71
4.5	Expected overhead on Scenario 3	72
4.6	Parameters modelling the creation of UDP flows in the simulation environment	78
5.1	Processing times obtained on a Athlon XP-M 1800+ (1533Mhz), ECDSA secp160r1, linux kernel 2.6.17, Openssl v0.9.8a	114
5.2	Delay and Jitter obtained for several numbers of forwarding nodes. Obtained with traffic UDP 64kbits CBR. Values are in milliseconds.	115
5.3	Overhead of each process measured for different packet rates. First sub-table depicts results obtained with constant packet size (512bytes) and variable packet rate. Second sub-table packet size varied (256, 512, 1024) and packet rate was maintained ($31pkts^{-1}$).	116
5.4	Overhead of each process measured for TCP. Considered both the cases where ACK packets are charged and where they are free. Also evaluated the usage of ECDSA <i>MAC</i> on charging headers (ECDSA 163bits)	117
5.5	Maximum throughput calculated for various packet sizes and rates.	118
5.6	Maximum throughput measured for various number of hops. Values are in kb/s	119
5.7	Comparison between the data received and reported in a 6 node string scenario using UDP flows.	120

Acronyms

Acronym	Description
AAAC	Authorisation, Authentication, Accounting, and Charging
AIMD	Additive Increase Multiplicative Decrease
AODV	Adhoc On-demand Distance Vector
AP	Access Point
AR	Access Router
CBR	Constant Bit Rate
CGA	Cryptographically Generated Addresses
CPU	Central Processing Unit
CSMA	Carrier Sense Multiple Access
CTS	Clear To Send
DAD	Duplicate Address Detection
DHCP6	Dynamic Host Configuration Protocol version 6
DNS	Domain Name System
DoS	Denial of Service
DSA	Digital Signature Algorithm
DSR	Dynamic Source Routing
DYMO	Dynamic Manet On-demand
ECDSA	Elliptic Curve Digital Signature Algorithm
ECN	Explicit Congestion Notification
ESP	Encapsulated Security Payload
FQMM	Flexibel QoS Model for MANETs
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
HIP	Host Identity Protocol
HIT	Host Identity Tag
IEEE	Institute of Electrical and Electronics Engineers
IMAQ	Integrated MANET QoS
IETF	Internet Engineering Task Force
INRIA	Institut National de Recherche en Informatique et en Automatique
INSIGNIA	In-band Signaling system for supporting quality of service in ad-hoc networks
Continued on next page	

continued from previous page

Acronym	Description
IP	Internet Protocol
IPSEC	Internet Protocol Security
IPv6	Internet Protocol version 6
IRDA	Infrared Data Association
LAN	Local Area Network
PRNET	Packet Radio Network
MAC	Message Authentication Code or Medium Access Control
MANET	Mobile Ad-hoc NETWORK
MN	Mobile Node
MPR	Multi Point Relay
NS-2	Network Simulator version 2
OLSR	Optimised Link State Routing
P2P	Peer to Peer
PACP	Polynomial-assisted Ad-hoc Charging Protocol
PCMCIA	Personal Computer Memory Card International Association
PDA	Personal Digital Assistant
PKI	Public Key Infrastructure
QoS	Quality of Service
QOSLSR	Quality of Service for OLSR
RAM	Random Access Memory
RFC	Request For Comments
RSVP	Resource Reservation Protocol
RTS	Request To Send
SACP	Session-aware Ad-hoc Charging Protocol
SURAN	Survivable Radio Network
SCP	Secure Charging Protocol
SWAN	Service Differentiation in Stateless Wireless Ad-hoc Networks
TBRPF	Topology Broadcast based on Reverse-Path Forwarding
TCP	Transport Control Protocol
TCP MSS	TCP Maximum Segment Size
TLV	Type Length Value
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunication System
VBR	Variable Bit Rate
VoIP	Voice over IP

Chapter 1

Introduction

1.1 Self-organised Ad-hoc Networks

From a functional point of view, organisation is of vital importance to any communication system. Without any form of organisation, the interactions between the entities composing the system are not defined and the exchange of information is not possible. Usually organisation adopts the attribution of roles to entities, each role defining a different set of services provided and consumed. A simple organisation method is the common server-client model where two roles are defined. The interactions of more complex systems often follow this paradigm. By combining, in the same network, different server and client roles, complex organisation schemes are possible to achieve. Current communication systems are designed this way. An architecture is defined where the roles of each component are clearly stated, typically following the old server-client model. Independently of the complexity of the system, usually only a small amount of self-configuration is allowed in the networks. The entire architecture is designed and implemented by engineers and equipments occupy fixed roles on the architecture. As benefit of such method the network architecture is always known to the administrators. For a small communication environment, such as a home environment, it makes sense for the hardware, software, addressing schemes and routes to be known to the administrator. The complexity of the system is low, having almost fixed topology and roles. In an environment such as a commercial operated network, the complexity of the architecture is divided into areas, functional units and individual modules. Such classification and the resulting modularity makes possible to replace components and to efficiently manage individual units. However all decisions are still performed by human beings.

This results in a respectable management overhead. When enough intelligence is put into the networks, the need for human control decreases and the level of complexity of the services provided and scalability will surely increase. Figure 1.1 depicts how the decentralisation of management increases the scalability of the systems. The drawback is reduced deterministic behaviour.

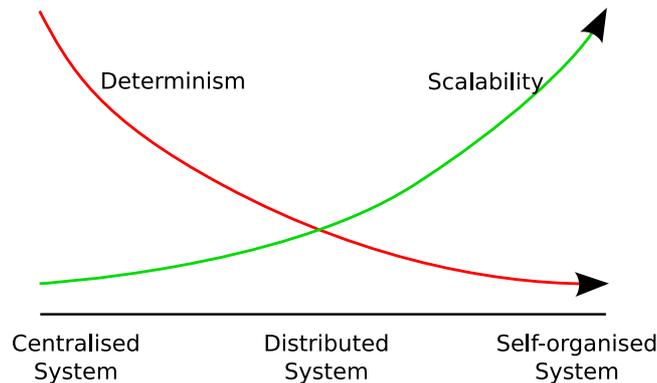


Figure 1.1: Scalability vs determinism in different types of networks [dressler06]

Nowadays, many forms of self-organisation are commonly found in ad-hoc networks: addressing can be performed dynamically; routing is able to detect topology changes and react accordingly; network policies are pushed into routers automatically based on higher layer policies; service discovery mechanisms are very useful in sharing resources between users. Although all these achievements, self-organisation is somewhat primitive and there are many situations where more intelligence and autonomic capabilities are desirable, where the network is able to identify events and where human control is simply inadequate and undesirable. The purpose of self-organised ad-hoc networks is thus to deploy a consciousness in the system, making it capable to take actions. The result will be intelligent ad-hoc networks where the human beings use the networks to enhance their interactions and where management overhead is reduced to a minimum. Current technology is still very far from this scenario.

Ad-hoc networks are a research topic getting much attention from the academic community. This is not because of the freshness of the topic (usually new topics get extra attention), but due to the potential presented by such concepts, especially with current wireless equipments. The concept of networks being able to self-organise without any human intervention or previous planning is rather old. Its origins can be traced back to 1972 and a program sponsored by the USA Department of Defence called Packet Radio Network (PRNET). Later during the following decade, this evolved into the Sur-

vivable Adaptive Radio Networks (SURAN) program. Both programs envisioned one of the most well established scenarios for ad-hoc networks: wireless packet switched communication in a battlefield, with moving elements with heterogeneous resources and roles, and the existence of an enemy which should be unable to intercept or jam communications. During the first phase of the initiative (PRNET), the mechanisms of ALOHA, CSMA and a simple distance-vector based routing protocol were combined in order to provide connectivity. The next phase (SURAN) enhanced the physical infrastructure by making radios more resilient to enemy actions, albeit smaller and energy efficient. With both these initiatives, the basic building blocks driving self-organised networks were studied: self-configuration, adaptability and failure resilience. In the early 1990s, a set of new developments slowly started focusing the research community in the direction of ad-hoc networks. Computers became lighter and powerful and the first laptops (or slightly smaller desktops) became available to the general public. Many of these equipments already had equipped a rudimentary form of communication by using infrared light. The first ideas bringing ad-hoc networks to civilian scenarios were proposed in a set of publications [perkins94], [johnson94] in conference proceedings. Finally, by the time of these publications IEEE started considering such networks and adopted the term “ad-hoc network” inside IEEE 802.11 [80211]. In the last half of the 90s, ad-hoc networks started to be a topic of much interest and rapidly a multitude of scenarios, proposals and also problems were identified. Many conferences and journals dedicated only to this type of networks were then created. By the end of the last decade, the IETF created the MANET charter which presented its terminology draft in 1997 [perkins97]. In more recent years, ad-hoc networks met even more applications and scenarios, with some important IST founded European projects like the BRAIN, MIND, DAIDALOS, AMBIENT NETWORKS and more recently WIP, considering new architectures where the concept of cooperation and self-organisation is vital.

Currently, in most ad-hoc networks it is common for mobile nodes to be heavily constrained devices. Self-organised wireless networks usually are deployed due to their low cost and most are also expected to be cheap, small and simple. They are often battery-powered and have limited CPU power, memory space and persistent storage space. The wireless interface is also very constrained specially because radios consume a lot of battery power, therefore, in order to increase the uptime, radio range is reduced to only a few ten of meters. Frequently nodes are mobile, not that they are capable of moving, but because they are carried by a person, an animal or another moving equipment. This is true for mobile users, sensors monitoring wildlife or vehicular networks.

Although all the effort dedicated to ad-hoc networks, because of the disparity of the applications, scenarios, nodes and network layers in which they can be implemented, no clear definition has ever been formulated. We can find concepts of ad-hoc networking in technologies and devices such as Bluetooth, IEEE 802.11, Sensors, Satellites and Space Probes or P2P networks. All these networks have some form of self-organisation and are somewhat dynamic, being resilient to changes in the environment. From all the concepts existing, the definition of four types of ad-hoc networks are important to this thesis: Mobile, Hybrid, Vehicular and Sensor. Many others exist, and also the definition of the given concepts may differ between some authors.

- **Mobile Ad-hoc Network (MANET):** Mobile Ad-hoc Network is the most common term associated to self-organised networks in today's literature. Also, this is the type of network having the most widely accepted definition. This was once formulated by the IETF working group: "A mobile ad hoc network is an autonomous system of mobile routers . . . connected by wireless links – the union of which form an arbitrary graph. The routers are free to move randomly and organise themselves arbitrarily; thus, the network's wireless topology may change rapidly and unpredictably. Such a network may operate in a standalone fashion, or may be connected to the larger Internet."
- **Hybrid Ad-hoc Network:** This network is an evolution from the previous one by considering that some devices have special functionalities. Although a typical MANET can be connected to a larger Internet, its supposed interconnection to be performed by standard nodes. In Hybrid networks, specially deployed nodes provide connectivity between an infrastructure network and the MANET. Such scenarios are typical for situations where an existing commercially driven wireless hotspot uses multi-hop relaying in order to increase its range.
- **Vehicular Ad-hoc Network:** Also an evolution from a typical MANET, the term Vehicular Ad-hoc Network refers to a situation where the network is comprised by automobiles cruising in a road (such as a motorway) and are able to communicate by means of MANET mechanisms. Such network may also contain sink nodes (access points) which are fixed in the road providing connectivity to the outside Internet. Both the definitions of Mobile or Hybrid network may be adequate to Vehicular. However the specificity of the mobility pattern associated with a Vehicular Network, as well as its applications makes the differentiation

worth.

- **Sensor Network:** A Sensor Network is a variety of an Ad-hoc network where nodes, either mobile or not, are stringently limited in terms of processing, battery and radio range. While in the previous cases the transport medium is IEEE 802.11, sensors frequently use other technologies such as Zigbee or Bluetooth. Nodes comprising Sensor networks possess simple processing units, have only a few KB of memory and are typically operated by standard alkaline or lithium batteries. Its application range is wide comprising scenarios where it is required to monitor locations, equipments or animals using cheap and easily deployed sensors. Proposals for Sensor networks are similar to the ones of Mobile Ad-hoc Networks, yet they must be as resource efficient as possible due to the additional constraints on the sensors.

1.2 Rewarding Concepts

Besides the adaptability to change, resources constraints and wireless interfaces, one another aspect is vital to an ad-hoc network to operate. All nodes should operate both not only as clients and servers, as in standard networks, but also as routers. Combining these functions on the same node is simple if nodes have resources, however this is not the case of ad-hoc nodes. Forwarding traffic coming from other nodes is required from the perspective of the network, yet it is undesirable from the perspective of the node forwarding. Each packet forwarded will reduce the already scarce bandwidth available and consume essential battery. CPU will also be occupied in the forwarding process and further consume battery. In order to maintain updated routing information, a routing protocol must exchange messages between routers. This will also decrease the available resources. More importantly, with the exception of routing, resources are spent not based on service utilisation by the owner but by other users. Even if battery is low and the user operating a device is not interested in generating or receiving any flow, there are still routing operations taking place. This situation will continue while other users produce traffic, the topology keeps changing or the battery is depleted. It is clear users may start acting selfishly and stop forwarding others traffic.

Forwarding nodes should have some reward in forwarding other traffic, to balance the inherent selfishness of the user. Also the reward should be attractive, motivating a correct behaviour and increasing network usefulness. This thesis addresses the issues

leading to selfish behaviour and mechanisms increasing user's motivation to participate. Many methods can be used and the most relevant to the state of the art will be described. The main focus of the work presented is charging and rewarding of users, as a mean to both integrate ad-hoc networks in commercial driven environments, and to increase the participation of users.

1.3 Objectives

This thesis aims to:

1. Study concepts associated to self-organised ad-hoc networks and methods motivating user participation.
2. Study architectures to increase the range of commercial hotspots through the usage of ad-hoc networks.
3. Analyse trust management in wireless public networks.
4. Consider the application of QoS and QoS based charging in wireless public networks.
5. Develop new solutions to secure charging in self-organised environments.
6. Implement novel solutions and validating them in laboratory environment.

1.4 Contributions

As the result of this thesis the following papers have been published:

- João Paulo Barraca, Miguel Almeida, Rafael Sarrô, Susana Sargento, Rui Aguiar, "Experimental Evaluation of an Integrated Ad-hoc Network", IST-Mobile Summit 2006, Mikonos, Greece, June 2006.
- João Paulo Barraca, Susana Sargento, Rui Aguiar, "Evaluation of MANET Charging Protocols in Hotspot Scenarios", Conferência sobre Redes de Computadores - CRC2005, Portalegre, Portugal, October 2005.

- Susana Sargento, Tânia Calçada, João Paulo Barraca, Sérgio Crisóstomo, João Girão, Marek Natkaniec, Norbert Vicari, Francisco Cuesta, Manuel Ricardo, “Mobile Ad-Hoc Networks Integration in the Daidalos Architecture”, IST-Mobile Wireless Summit 2005, Dresden, Germany, 2005.
- João Paulo Barraca, Susana Sargento, Rui Aguiar, “The Polynomial-assisted Ad-hoc Charging Protocol”, IEEE International Symposium on Computers and Communications - ISCC2005, Cartagena, Spain, 2005.
- João Paulo Barraca, Susana Sargento, Rui Aguiar, “A Lightweight and Secure Session-Aware Ad-Hoc Charging Protocol”, International Conference on Telecommunications - ICT2005, Cape Town, South Africa, 2005.

1.5 Disposition

This thesis is organised as follows:

Chapter 2 introduces the issues and solutions related to self-organised ad-hoc networks. In this chapter an overview of the proposals shaping today's self-organised networks is presented. This will cover the issues of routing, self-configuration and quality of service, which are vital for these environments.

Chapter 3 addresses the problematic of cooperation and charging in ad-hoc networks. The issues which required the creation of proposals motivating users to forward are presented. Also, some of the most representative proposals are described as they influenced the work developed for this thesis.

In Chapter 4 two new proposals are presented together with an analytical comparison with another proposal. Part of this chapter also describes and analyses the results of the extensive simulation work which was performed.

Chapter 5 describes the internals of a prototype implementation of one of the proposed solutions. Design choices of the implementation and the testing environment are also described. Also in this chapter, the results obtained in a testbed are presented and discussed.

Chapter 6 presents the main conclusions of this thesis together with some issues still requiring further work.

Chapter 2

Solutions for ad-hoc networks

Self-organised ad-hoc networks are made by the inter-operation of heterogeneous nodes using a multitude of protocols. Each different protocol provides a layer of services capable of enhancing the capabilities of the wireless network. Easily the combined capabilities will much surpass the resources handled by one single node. On this chapter, solutions providing routing, auto-configuration and Quality of Service for ad-hoc networks are discussed. Although this thesis focus on charging and cooperation, these modules are the main building blocks of the networks studied. Only after these services are functional, more complex solutions may be deployed. Also, it is almost inevitable to consider interactions between protocols at different layers. All these issues make necessary to describe a set of solutions which in some way influenced the work developed.

2.1 Routing

Routing is the function responsible for determining the best route from a source to a given destination. After route is determined, forwarding mechanisms process the packet according to the information in the routing tables. Because the own nature of ad-hoc nodes, routing protocols for MANET should be highly dynamic and robust to a myriad of problems, non existent in standard wired networks. Mobile ad-hoc environments must cope with high mobility, failures of the underlying IEEE 802.11 medium, unidirectional links, grey areas [henrik02], disruptive action or simply selfish behaviour perpetrated by malicious users. Topology may change during session lifetime, requiring the routing protocol to react and update routes between end-points.

When developing routing protocols suited for the typical environment of ad-hoc

networks, many different strategies and algorithms can be chosen. Because of the choices adopted by each solution, it is usual to classify ad-hoc routing protocols in different categories. This categorisation helps in evaluating the optimal environment for each protocol, its weakness and strengths. Moreover it helps reaching a consensus about the appropriate set of mechanisms suitable for a particular situation. One of the most important categorisation parameter is the strategy to determine the best route between two end points. Protocols can be also categorised in many other ways like the algorithm used to determine best route (link-state or distance vector), the type of traffic they forward (unicast or multicast), how (or if) they handle geographical information, the effort they make in reducing the power consumption or if they create (or not) a hierarchical structure inside the MANET. Protocols which actively monitor network changes and update routes automatically are denominated Proactive or table-driven. If a protocol only maintains routes when some flow requires it, it is denominated Reactive, or on-demand. Some protocols use both methods and are categorised as Hybrid. Reactive and Proactive are the two most popular categories and are further discussed in this thesis.

2.1.1 Pro-active routing protocols

Pro-active protocols will synchronise topology information among a set or all nodes in the ad-hoc network and build a distributed routing table. This is accomplished by exchanging periodic messages between nodes. Using the exchanged messages, each node is able to build a representation of the network topology which can be partial or total. Routes are then established using the knowledge acquired and can take in consideration additional metrics like delay, security or even price. The drawback is the control overhead produced by the constant messages exchanged even when there are no data packets to be routed. Depending on the synchronisation method, protocol overhead can stay almost constant with varying mobility and/or number of flows routed. However in most pro-active protocols, overhead increases together with the number of nodes. These control messages consume CPU time, battery and wireless resources, even when there are no active flows in the MANET. Pro-active protocols are less adequate to networks where nodes have very low capabilities like sensors, but adequate where routes should be provided with low delay and be maintained constantly.

OLSR - Optimised Link-State Routing protocol

The Optimised Link State Routing protocol is one of the most mature and popular routing protocol for MANET. It was developed at INRIA and first proposed in [clausen01] as a link state, table driven (pro-active) routing protocol for mobile wireless networks. Later it went through a development process inside the IETF MANET [manet] charter, being proposed to the Experimental RFC status in RFC 3626 [clausen03]. There are almost ten public implementations of OLSR (in July 2006), many conforming to RFC 3626 and supporting most operating systems.

Each control packet generated by OLSR can have multiple control messages providing an optimisation of the resources required for protocol operation. Messages start with a header identifying the message type. Upon reception, nodes quickly identify the type of the message and are able to forward even unknown message types. This is particularly useful when introducing a new functionality to OLSR which only a limited number of nodes support. New nodes could support the additional feature while the network would still operate because old nodes simply ignore the new message.

OLSR operation is divided in 3 main functions: neighbour sensing, optimised flooding and forwarding, and link-state messaging and route calculation. Neighbour sensing function provides information about status of links between nodes. It is implemented by the exchange of periodic *HELLO* messages at all interfaces where OLSR is active. Considering 2 nodes close by, the function may determine they are not neighbours if no packets are received by either node; the link is unidirectional if only one node receives *HELLO* packets or bidirectional if both nodes receive each others' *HELLO* packet. Such situation may seem uncommon on a perfect medium but IEEE 802.11 and the wireless medium are far from that. The precision of such process is very important to determine and maintain routes during topology changes or network start.

The optimised flooding and forwarding is one of OLSR most innovative aspects. To carry efficient flooding and determination of a forwarding route, OLSR employs the method of Multi Point Relaying in order to reduce overhead by avoiding duplicate retransmission . Using MPR, the flooding process is directed through a distribution tree both reducing the number of nodes involved in flooding and avoiding duplicated forwarding of the same packet. The size and structure of the Multi Point Relaying tree varies with network topology and number of nodes. Nodes can choose to participate or not in the forwarding tree and can choose their multi-point relays using a tunable

algorithm. To choose his MPR each node calculates the best neighbour which allows reaching any other node two hops away. Redundant nodes are detected and removed resulting in an efficient tree providing close to optimal routes. Figure 2.1 depicts an example of the optimisation resulting from flooding the network using the MPR method.

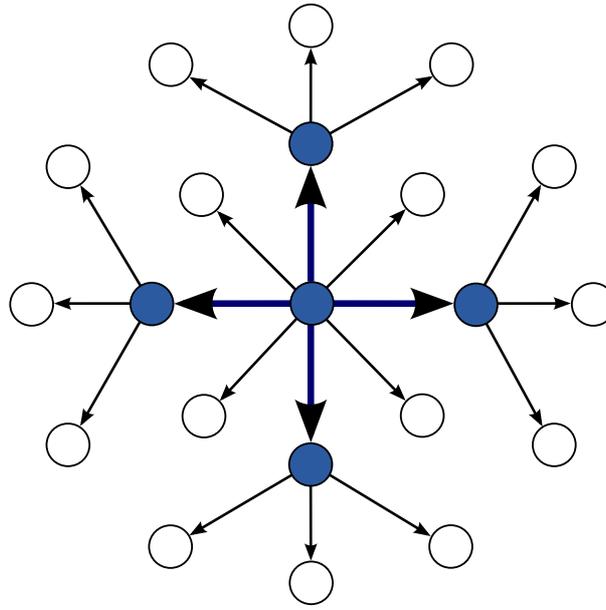


Figure 2.1: Flooding using Multi Point Relaying

Link state and routing calculation provide optimisations over traditional link state routing. In OLSR only MPR nodes generate link state messages and only MPR nodes are declared in messages. The combination of both solutions (MPR and optimised link state) is able to efficiently reduce the complexity of computing routes and the overhead of the routing protocol.

OLSR has a clever and polished design with some other aspects of relevance: supports multi-homed nodes and disseminates interface information to other nodes in the network; defines the possibility of a node to explicitly define its MPR status (i.e. participate in the MPR forwarding), allowing resource restricted nodes to be excluded from the MPR tree or carefully deployed and more powerful nodes to always be MPR nodes; connectivity is not restricted to the ad-hoc network making possible inter-operation with hosts on external networks; link hysteresis detection is supported in order to prevent using links where communication of user data packets suffers noticeable degradation (also to avoid grey areas).

Compared to other solutions, OLSR usually fails to provide the packet delivery

ratio [bhatia03] of some reactive protocols like AODV [perkins03b]. Also, because of its proactive nature, overhead in networks with high number of nodes, is much higher than the one found in reactive solutions, also making it unsuitable in low power environments, such as sensor networks where networks may be composed by hundreds of nodes. Most recent proposals related to OLSR are mainly concerned with aspects like QoS [badis03], security [hafslund04] and network auto-configuration [jelger04]. Also a new improved version of OLSR was started recently in the form of OLSRv2 [clausen06]. This new version, besides many other improvements, aims at reducing the complexity of the protocol and the overhead generated by using methods of partial topology dissemination.

TBRPF - Topology Dissemination Based on Reverse-Path Forwarding

The Topology Dissemination Based on Reverse-Path Forwarding (TBRPF) is a proactive, link-state routing protocol proposed for mobile ad-hoc networks providing multi hop routing with optimal paths. The protocol was first proposed in [bellur99] and recently reached the status of Experimental RFC 3684 [ogier04]. The protocol is divided in two different parts: the Routing Module (TRM) and the Neighbour Discovery (TND) module.

The TBRPF Neighbour Discovery module has an instance on every TBRPF enabled interface and takes care only of maintaining neighbour information updated and valid. Being a proactive protocol, every TND module periodically broadcast a routing packet (*HELLO*). This packet, besides its beacon function, carries information about interface's active neighbours and the status of the links. TBRPF considers three states for links: lost or non-existent, unidirectional and bidirectional. Using a modified version of the Dijkstra's algorithm each node is able to compute the best path to any destination required, based on the exchange of *HELLO* messages. In order to reduce the overhead of the periodic *HELLO* messages, TBRPF authors propose the usage of "differential" HELLOs. Instead of sending information about all neighbours on each *HELLO* packet, upon modification of a link, TND only sends information about changes. On a static network, overhead is greatly reduced while in a permanently moving network it behaves similarly to other proactive routing protocols.

The TBRPF Routing Module (TRM) maintains a source tree providing the shortest path to all nodes in the network. The memory and computation requirements of

storing and processing such tree puts some limits in the maximum number of nodes on a network. The TRM module periodically broadcasts the source tree of each node to their neighbours, but, as an optimisation to reduce resource consumption, only part of the tree is broadcast instead of the full tree. Additionally, a differential update of the tree is sent to denote changes in the topology promoting a quick convergence of the routes.

As in OLSR, TBRPF nodes are allowed to control the level of participation on the forwarding process, but with a different method. As presented previously in section 2.1.1, in OLSR nodes can express their willingness to forward. In TBRPF, nodes express their willingness indirectly through the size of the source tree they broadcast. A non cooperative node unwilling to forward packets will make sure other nodes do not select him as a hop and will not broadcast its tree. If a node wants to forward as much traffic as possible, it should increase the amount of the source tree it broadcasts at the same time. Although there is no certain it will be chosen, because it advertises more available destinations (and routes), in average, it will be selected more often.

2.1.2 Reactive Routing Protocols

Reactive routing protocols differ from the previous (Pro-active) as no route is pre-calculated. If no packets need to be routed, nodes have no information about the current topology or route to any destination. Information about current neighbours is typically the only information gathered. When a node wishes to send a packet, a route discovery mechanism floods the network searching for a route to the destination. When a route is established, nodes will maintain that route as long as there are data packets flowing. After the flow stops and a given timeout expires, routes are dropped. Reactive protocols are much efficient in terms of bandwidth, memory and processing requirements. All computations are simple and nodes only maintain minimal amounts of information. Moreover, the overhead in the network depends both on the number of nodes, mobility and number of flows. If there are no flows, routing overhead will be almost nonexistent. The drawbacks with these solutions are higher latency establishing routes and lower redundancy.

DSR - Dynamic Source Routing

The Dynamic Source Routing protocol is a simple, lightweight and efficient solution for routing in mobile ad-hoc networks. DSR operates on-demand and, instead of using the entries in the routing table of forwarding nodes, it uses source routing to deliver packets. The protocol is composed by two main functions: Route Discovery and Route Maintenance. When a node wants to forward a packet it should issue a *Route Request (RREQ)* message to all its neighbours. This packet is forwarded until it reaches the destination. Every time the message is forwarded, nodes add their address to the payload of the message, and the message grows with the list of all forwarding nodes. In order to avoid forwarding the same packet more than one time, every *RREQ* message carries an identifier used to distinguish different request. Also, nodes will not forward packets if their address is already in the message. With such mechanism, the loop free operation of the proposal is assured. When the *RREQ* message reaches the destination or a node which already knows the route, this node issues a *Route Reply (RREP)* message towards the origin of the message. Before issuing the *RREP* message, the destination node checks if a route to the initiator is known. If this is not the case, another Route Discovery process is required. The original *RREP* message can be piggybacked on the new *RREQ* message saving bandwidth and increasing the convergence of the mechanism. If the underlying MAC medium provides bidirectionality, such as frequently happens with IEEE 802.11, the destination node could further reduce the overhead by reversing the route list contained in the first *RREQ*. When the Route Discovery process is completed, the receiver has a list of hops willing to forward packets to the desired destination and is able to start sending data. To every data packet is added a routing header with the discovered list and sent to the network. In order to avoid the necessity of restarting the process on a packet basis, this route is cached for some time. If no *RREP* message is received after sending a *RREQ*, an exponential backoff mechanism delays further *RREQ* messages.

The information resulting from each Route Discovery mechanism is not confined to the nodes participating in the forwarding. DSR specifies nodes can listen for the traffic in the network and cache routes created by neighbours. Such mechanism can bring some problems with uni-directional links and consume additional battery from low power devices. However it is an effective manner of disseminating information about the network, making possible to react faster to topology changes. When a node moves, one of its neighbours will be a candidate to replace it in the forwarding. If this

new node listened to the Route Discovery process, it will have this information, making possible to react faster. One aspect of this cached information is *RREP* messages can be issued (as response to *RREQ* messages) by forwarding nodes if they already know the route to the destination. With this mechanism, Route Discovery is much faster, but the initiator must be able to select the best route from all routes received. Also, forwarding nodes must check if replying with cached information will create loops in the route. If this is the case, this list contained in the *RREQ* message must be merged with the cached route and the result included in the *RREP*.

One interesting aspect of DSR is the support of per hop delivery acknowledgement. This can either be performed using DSR messages, by listening to the next node forwarding the packet or by using MAC layer acknowledgement messages. If no acknowledgement is received after a timeout, the packet is retransmitted and after a number of retransmissions, a *Route Error (RERR)* message is issued towards the sender. The sender now has to restart the Route Discovery process or add another known route to data packets.

Source routing method is very effective, however it suffers from overhead. As the length of the route increases, the routing headers also increase. Using IPv6, this results in 16 bytes per packet per forwarding node. Forwarding nodes, knowing the route to other nodes in the route, may omit other nodes in between, saving this overhead in data packets. Being a relatively mature and simple solution, there are many publications supported by the mechanisms of DSR. Also, the source routing solution employed, allows for a myriad of solutions for many other problems such as charging, multipath, cooperation or quality of service. Currently, developments on the DSR protocol seem to have slowed, with the researchers focusing on solutions such as DYMO [chakeres06].

AODV - Ad-hoc On-demand Distance Vector

The Ad-hoc On-demand Distance Vector [perkins03b] routing protocol is one of the best known and well tested routing protocols for MANET environments. It functions reactively providing routes only when needed. These routes are maintained only during flows duration and are destroyed after a determined timeout.

Routes are created when an initiator sends a *Route REQuest (RREQ)* message asking to reach the desired destination. The request is efficiently forwarded through the network, avoiding loops through the usage of a sequence number. When a node, receives

a *RREQ* message and has knowledge of a route to the requested destination, it issues a *Route REPLY (RREP)* message to the origin of the *RREQ*. The initiator will thus receive multiple *RREP* messages and is capable of selecting the most suitable route based in the number of hops. When using AODV [marina02], multiple routes can be selected and some proposals [perkins03a] and [zapata05] consider extensions to support additional metrics. Nodes forwarding *RREP* messages also update their route information. If a more recent sequence number indicates a potentially shorter route, this new route replaces the existing one. If no packets of an existing route are being forwarded, the route entry is discarded after a timeout. This helps in reducing the existence of old (and frequently incorrect) information in the network and to reduce memory requisites. If topology changes during the existence of a flow, the node detecting the break will issue a *Route ERROR (RERR)* message to the sender. After a *RERR* is received, a new route discovery process must be initiated so that packets reach destination.

More than 10 public implementations are referred at AODV website [aodvweb] supporting different operating systems, architectures and IP versions. Also inter-operation between these versions has been tested [beldingroyer02] with much success. Currently, after extensive comparison publications and proved functionality, AODV has reached a stable state. New ideas and the knowledge acquired during its development are now shifted towards the creation of DYMO [chakeres06].

DYMO - Dynamic Manet On-demand Ad-hoc Routing

In 2004, the IETF MANET Charter announced on its mailing that no more effort should be spent on new routing proposals. Tens of proposals already existed and the problem of dynamic routing was well understood. Research topics focused on evaluation of the developed proposals and its usability in the various scenarios envisioned for ad-hoc networks. After the group learned from the extensive results obtained, in 2005, Ian Chakeres et al, led the definition of a new routing protocol called DYMO. It is heavily based on AODV, using the same scheme for route discovery and maintenance. Nodes still send *RREQ* messages when require routes, and responses are sent on *RREP* messages. Route breaks is also still notified using *RERR* messages. The main differences are related to simpler route maintenance, higher abstraction, routing gateways and the possibility of extensions.

In DYMO, routing messages contain a common header and data is sent on *Routing*

Elements (RE). Each element contains a type specifier, source and destination address, Length, TTL and other flags. It is not required for all nodes to support all types of *RE*, and unknown *RE* codes can be ignored. If the *RE* type defines it, unknown reception of unknown *RE* types leads to an *Unknown RE Error (UERR)* message to be sent. This feature by itself, makes DYMO more expandable and adaptable to future enhancements than AODV. Also it makes possible coexistence of nodes implementing different *RE* types.

DYMO natively supports the existence of gateways. Nodes are thus able to advertise routes to a different network using a combination of network address, mask and the Gateway flag. Gateway nodes respond to *RREQ* messages querying for routes to nodes outside the ad-hoc network. In the *RREP* message, they advertise the address they are able to reach and that they are a gateway. Besides *RREP* messages, all packets processed or created by a gateway node, must indicate the gateway status. Current work on DYMO is focused on further elaboration of the current specification and comparative studies assessment.

2.2 Address Auto-configuration

Routing protocols are only able to route packets in a network where nodes are already configured with non conflicting IP addresses. Without this step, the same address could appear repeatedly, or interfaces could be using different network masks making routing impossible. Furthermore, routing protocols frequently do not handle interconnection to outside networks (such features are usually proposed as extensions). If nodes are not aware of eventual gateways, routing will only be available inside the ad-hoc network thus making impossible scenarios related to integration with other networks.

It is clear that these auto-configuration issues are out of scope of routing protocols. However these depend on the existence of mechanisms responsible for disseminating network information and configuring nodes. DYMO, which already specifies Internet connectivity and auto-configuration, is one example of a more recent routing protocol that is already aware of such issues. Comparing with routing, at least in the number of publications available, configuration proposals are lagging behind routing proposals. The IETF MANET Working Group created the Autoconf Charter [autoconf] and started developing efforts in evaluating the requirements to address auto-configuration

in MANET. The work inside Autoconf is recent, drafts are still in very early development stages and no RFC was proposed.

Current trends propose the usage of a simple protocols broadcasting important configuration information. The most popular protocol, at least in wired networks, should be standard IPv6 address auto-configuration [thomson98]. It describes the mechanisms for automatic IPv6 address configuration based on MAC addresses and information provided on Router Advertisement messages. A Duplicate Address Detection (DAD) phase avoids the existing of conflicting nodes with same address. Other proposed methods range from network delivered addresses by DHCP6 [droms03] to more complex solutions like Cryptographically Generated Addresses (CGA) [aura05]. Information about the available gateways and the distance, in terms of number of hops, could also be delivered to nodes. This would make possible nodes to differentiate between several existing networks and select the one with less number of hops to the gateway. When considering more complex ad-hoc networks, besides information like address, mask or default route, it could be important to further disseminate information about services available to the network. Due to resource constrains not all services can be announced, but basic services required for inter-operation should be announced. Examples of such services are: set of protocols to use, location of DNS and Authentication services and details of the PKI available (if any).

Proposals like [wakikawa06] [jelger04] and [jeong06], being developed as Internet drafts, present methods for disseminating network configuration. [wakikawa06] proposes a method to propagate the network prefix inside the network by means of an Internet Gateway Discovery process, similar to the Router Discovery process of IPv6. In the proposed method, the address and network mask of the gateway, together with a sequence number and a timestamp are transmitted in a Modified Router Advertisement message. The authors envision several solutions capable of integrating the solution with various routing protocols and Mobile IPv6. No security mechanisms are specified, making this solution much vulnerable to a set of impersonation and Denial of Service (DoS) attacks: a malicious node, incorrectly advertising network information, is capable of severely disrupting network operation.

Jelger et al [jelger04], proposed a method where the gateway providing connectivity to the Internet periodically broadcasts a message denominated GW_INFO. The message is forwarded by all nodes which currently selected the gateway sending the message as its default gateway. This forms an optimised flooding method following

a directed tree sourced at each gateway. One point of interest is that this provides a method of supporting multiple gateways in the same ad-hoc and allowing nodes to choose which to use. Because a node selected a specific gateway and an upstream neighbour connecting him to the gateway, the routing protocol may decide to use a different route to the Internet. The default route will always be their upstream neighbour but it will only be used if the routing protocol does not provide a route (eg. destination is out of the ad-hoc and routing protocol is not aware of it). Besides IPv6 address configuration, it specifies an extension to GW_INFO messages which is able to inform nodes about the address of a global DNS server to use. There are no security considerations in [jelger04] allowing nodes to forge GW_INFO messages, spoofing the address of the gateway or the address of the DNS server. Recent proposals [calçada05] emerging from the IST-Daidalos project [daidalos] extend this proposal by incorporating message authentication and integrity to control messages. Nodes are then able to verify the authenticity of the GW_INFO message, solving the spoofing vulnerabilities the original proposal suffered from.

Jeon et al propose [jeong06] another solution to auto-configuration. It differs from the previous by specifying mechanisms both for IPv4 and IPv6 and to allow auto-configuration of isolated ad-hoc networks. Also it supports the existence and merge of different partitions of the network. The bases of this proposal are the mechanisms of Strong DAD and Weak DAD. The first process is used to rapidly detect a duplicate address in the local partition of the node. The later's used to detect duplicate addresses during routing and, particularly, during merge of partitions. When a node wants to join a network it starts the Strong DAD process by setting a temporary address and issuing a message requesting the address (AREQ). The message is flooded to the entire network one hop at a time. If any node has the address announced it should send an AREP message and the incoming node should restart the process. Nodes also send AERR messages when they detect an address conflict. When two or more partitions merge there is a possibility of existing duplicated addresses between nodes of the different partitions. The Weak DAD will ensure collisions do not occur by keeping, at each node, a table with addresses and so called Interface Identifiers. If there are two or more entries with the same address and different Interface Identifiers there are nodes with conflicting addresses. If a nodes' address change and active flows exist, the authors specify a method to notify active peers of the new address by issuing a specific AERR message. Most recent work also refers the use of IPSEC ESP with a shared pre-distributed key to authenticate control messages. No support for other auto-configuration param-

eters besides IPv4 and IPv6 address configuration was envisioned. Globally routeable addresses are referred as usable, but the network must provide the network mask to use. The process of choice is the standard Route Advertisement/Route Request process but performed in a multi-hop manner. In case of multiple gateways, nodes can configure multiple global addresses, but multiple Global addresses raises some issues because traditional MANET routing protocols usually only support one prefix inside each network.

2.3 Quality of Service

Quality of Service (QoS) generally aims at providing differentiation on the way packets are handled either (or both) at the sending node or at intermediate nodes. If no other definition exists, standard packets are always scheduled as Best Effort (BE). Interestingly some authors [chown03] describe situations where Less than Best Effort (LBE) traffic is also possible (and useful). Other higher priority classes may exist which are related to higher priority in queueing process. Real time classes are used by services like VoIP calls which require low delay and jitter. The IntServ model defines the existence of end-to-end reservation for the classes. Besides priority, also bandwidth, delay or jitter parameters are possible to be applied. Although more powerful than the more recent DiffServ model, IntServ also requires more state information to be kept at each node. In order to avoid routers to retain states of long terminated flows, it is required to refresh the reservation periodically. This method is very effective at the cost of additional overhead.

The protocols for wired networks provide powerful mechanisms and are well known for a long time. They provide mechanisms like end-to-end flow reservation, QoS routing or differentiated services. Wired networks are much stable and resourceful than ad-hoc networks and QoS is common from the user terminal to network provider's transport cores. In ad-hoc networks resources are scarce. Solutions like RSVP [braden97] are overkill and many other proposals following the IntServ model are unsuitable.

In ad-hoc networks, service differentiation is typically applied on a class basis providing differentiation between critical services and best effort traffic. Mainly due to the instability of the IEEE 802.11 medium and the mobility of ad-hoc nodes, it is common to consider signalling (Routing, QoS, Charging) also as a real-time service. Typically ad-hoc solutions do not aim at the complexity of traditional QoS solutions

like RSVP. The basic ideas of QoS in MANET are based on already existing solutions but often offer simplified models. Most of the current proposals allow to differentiate between a small amount of services (less than 10) while the most complex additionally allow to combine some other metrics like bandwidth and delay.

An ad-hoc network integrated with an infrastructure requires QoS mechanisms efficient enough to be deployed in mobile nodes, and powerful enough to handle wired networks requirements. Instead of performing differentiation based on services or a traffic class, each node has a user profile associated typically containing some QoS constraints. Some solutions to this problem introduce additional signalling and intelligent shaping and admission control modules like it is proposed at [crisóstomo05]. Others propose mechanisms to choose the most appropriate gateway based on additional parameters [sethom06].

2.3.1 QOLSR

Although not a standalone solution providing QoS in ad-hoc networks, the QOLSR [badis03] protocol has been actively developed inside the IETF MANET group. Because of the pluggable nature of OLSR a QoS solution was developed as an extension to standard OLSR. The QOLSR improves OLSR by adding an extension to *HELLO* and *TC* messages. This extension disseminates information about many network parameters like available bandwidth, delay, jitter, loss probability or cost. The election of MPR will be performed as in OLSR but considering the additional parameters. Bandwidth and delay parameters are the only ones which should always be included in messages and should be always available. Accordingly, each node should select its set of MPRs so that they are able to reach all 2 hop neighbours with maximum bandwidth and lowest delay. If an MPR informs its bandwidth and delay values dropped below a given threshold, the non-MPR node will restart the process of selecting a new MPR taking in consideration the updated values. Calculation of routes will be performed using one of two algorithms; QoS constrained shortest-widest path or an alternative based on Lagrangian relaxation. The first is used if only bandwidth and delay should be considered. If more parameters are to be used, the Lagrangian relaxation based algorithm can provide a number of solutions optimising the QoS constraints in use. The solution proposed is usable both in IPv4 and IPv6, only by changing the type of addresses included in *TC* and *HELLO* messages. Security and cost of links are mentioned in [clausen03] and two TLV types are already reserved for this.

2.3.2 INSIGNIA

INSIGNIA was first proposed by Ahn et al [lee98] as a lightweight, in-band implementation of the RSVP model. At the time of its publication INSIGNIA was the most complete and supported QoS proposal taking in consideration the particular requirements of ad-hoc nodes. It supports fast reservation, restoration and adaptation of flows delivering flow differentiation between Real Time and Best Effort traffic.

QoS signalling is exchanged as an in-band extension to IP packets (called INSIGNIA option) in an attempt to minimise the bandwidth due to signalling overhead. The INSIGNIA extension is composed of four 1bit fields followed by two 8bits fields taking a total of twenty bits per packet. When a node wants to perform a reservation it should mark the data packet with a Reservation message indicating the bandwidth and class values desired. Here nodes can reserve for Minimum guaranteed bandwidth (base QoS) and Maximum used bandwidth (enhanced QoS) depending on the application requirements. The packet will be forwarded along the route making soft reservations on intermediate nodes which are refreshed by data packets. If a node has no available resources to perform the reservation, the packet is still forwarded but treated as best effort and no reservation is created.

In mobile environments, such as a MANET, it is common that reservations are not possible to be performed in the moment the flow is created but only later. This is due to flow characteristics or routing changes. The opposite case is also common and INSIGNIA provides an adaptive mechanism which dynamically drops and recovers reservations without explicit signalling between or to endpoints. A feedback mechanism is also implemented allowing the destination node to inform the flow source about the state of the reservation and eventual degradation due mobility. The feedback mechanism is of vital importance during the setup of a reservation so that the sending node can have some assurances that the reservation was performed. Also, because the path or the channel parameters may change, the feedback mechanism allows sending nodes to adapt flows to maximise usability. The reason this adaptation is application driven is justified by the different requirements of applications. Real time MPEG flows, VoIP calls or bulk downloads are frequent exemples pointed out by INSIGNIA authors.

The stateful behaviour of INSIGNIA, also characteristic of the Intserv model, makes INSIGNIA not suitable for networks with high mobility or high number of nodes. Because of the not so lightweight and the adaptable behaviour of INSIGNIA, later

research focused on other proposals and models, such as the FQMM [xiao00], IMAQ [chen02a] and more recently SWAN (DS-SWAN) [domingo04].

2.3.3 SWAN

The Stateless Wireless Ad-hoc Network [ahn02] proposal is a complex QoS model providing stateless QoS differentiation in ad-hoc networks. SWAN is comprised of several distributed mechanisms glued together to form a complete QoS solution. These mechanisms are: Admission Control; Rate Control; Traffic Classification and Shaping; and MAC feedback. Figure 2.2 depicts the relations among different modules.

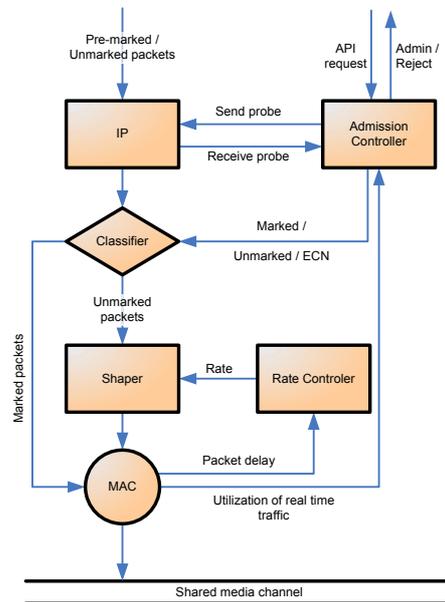


Figure 2.2: SWAN Internal structure

The Admission Controller module operates at source nodes and controls the admission of UDP real time flows only. Deciding whether a flow is allowed or blocked will depend on feedback from other mechanisms: Active probing, ECN detection and MAC feedback used.

Active probing works by sending probes from the origin to the next hop when a new real time flow is created or when renegotiation is required. Routing may also change the QoS characteristics of a path thus also invoking a probing process. Notice that the packet is forwarded and processed at each intermediate hop and not sent directly to the destination node. Each node changes the bottleneck bandwidth thus limiting the maximum rate acceptable to the lowest acceptable rate along the path.

The destination node will then send a probe reply with the available bandwidth in the path.

Rate control module shapes best effort traffic using adaptable shapers and is performed in a distributed manner at every node. Real-time traffic should not be shaped (only admitted). The algorithm used to adjust the shapers is denominated Additive Increase Multiplicative Decrease (AIMD) and was first proposed at [chiu89]. According to AIMD, best effort traffic starts at rate $|c|$ and the rate increases by $|c|$ kbits every time unit. After a few time units, if no other traffic exists, best effort traffic uses almost all available resources, just as in a normal situation. When the MAC layer reports a delay higher than a determined threshold D ms, the shaper is adjusted and rate is decreased by a multiplicative factor $F\%$.

MAC feedback is specified only for 802.11 standard, though. It works by enabling RTS/CTS negotiation for all send packets and measure the delay between the first RTS to send a data packet and the ACK received from the next hop. Collisions and delay getting access to the medium are then reported to upper layers. MAC layer also eavesdrop the radio environment and permanently calculates the occupancy of the network and available bandwidth. Forwarding nodes may detect packets are congesting the network (using MAC feedback). In this case they mark the ECN field of forwarded packets. The receiving node, upon reception of an ECN marked packet will issue a Regulate message to sender causing the Admission Control on the sending node to re-probe and re-admit the flow. If multiple flows exist simultaneously to the same destination, and no avoidance measure is considered, a regulate message will force all sources to restart the probing process. Improvements may involve insuring a delay between probes or to mark only one, randomly selected flow for congestion, instead of all flows. If the link still continues to have congestion, other is selected until congestion stops. This solution of shapers adaptable by feedback presents many problems to bursty data, due to the time shapers take to adapt. Examples of problematic applications are: Instant Messaging Service, DNS requests or even a single ping. If these are considered as best-effort they will suffer from extreme delay and packet loss until shapers reconfigure. If the shapers allow for burst of best effort traffic it could be problematic to existing real-time flows because they could experience degradation (from collisions) only corrected in the next time unit.

Classification of packets is performed at the sending node and packets are marked using the DSCP field (in IPv6). Originally SWAN only considers the existence of

2 classes: best-effort and real time; recently other authors [crisóstomo05] proposed adopting SWAN to control 4 classes in total. All classes with exception of the Real Time class will suffer shaping and shapers will be placed one after the other. The delay provided as feedback to each shaper will thus correspond to the total delay accumulated in higher order shapers. The initial feedback will correspond to the MAC delay.

Chapter 3

Cooperation in Ad-hoc Networks

The concept of a spontaneous network, composed by wireless nodes getting together at radio range, all sharing the idea of cooperating to a common purpose, and “everything simply works”, is, in almost all situations, a utopia. Real world ad-hoc networks are far from perfect and are affected by a huge amount of difficulties limiting its usage, amongst which the lack of equal interest in the node’s cooperation.

Some authors [huang04] claim that incentive and cooperation should only be deployed when networks reach a respectful maturity. Although this concept can be partially true, cooperation between nodes, in the general sense, must be vertical and exist at all levels. Three types of non-cooperating nodes are usually considered: faulty, malicious and selfish.

The first type of nodes do not follow the protocols due to some malfunction. The second are nodes operated by users who do not wish the correct behaviour of the network. Both aspects can be permanent or temporary. Faulty nodes are expected to be very common due to software or hardware faults but especially due to problems (interferences, network congestion) in accessing the 802.11 medium. Malicious nodes are expected but its number should be small as, generally, specialised tools or knowledge is required and only a rejection of service is achieved by the offending node. Selfish nodes differ from the malicious nodes because there is an intention to abuse the network resources in self advantage. One common selfish action is to simply drop packets forming a black hole. Routes are computed through the selfish node but no data packets are actually forwarded. In opposition to the others, where there is not much additional protocols can do, selfish users can be motivated to participate.

Cooperation at the MAC layer is commonly achieved by the rules composing an access protocol. The physical medium, typically IEEE 802.11b/g/a, provides a shared resource, affected by problems like packet collisions, radio interferences and hidden nodes. Because the medium is shared and IEEE 802.11 radios are half duplex, if no reservation mechanism is deployed, packet collisions will severely degrade the bandwidth available and increase round trip delays and packet losses [li02]. Thus nodes will need to manage together the wireless medium in order to achieve a better user satisfaction, though the access protocol. Radio interference is external to the ad-hoc network and there is nothing nodes can do besides changing wireless channel, or choosing routes based on additional parameters like available bandwidth, delay or noise. The later has the problem these changes must be coordinated and all nodes must change at a time. In practice it is rarely convenient, due to the technical difficulties and reduced number of channels available to 802.11b/g. The hidden node problem is inherent to the channel and must be solved by all nodes cooperating using an organised and respectful mean of communication. Typically this is minimised by using the Multiple Access with Collision Avoidance mechanism. Whenever a node wishes to send a packet, first it tries to reserve the medium by issuing a Request to Send (RTS) message. If the destination node is ready to receive data, it issues a Clear To Send (CTS) message. Other nodes listen to the message exchange and no station will send packets until this reservation expires.

At the transport level, nodes must cooperate in the discovery of routes and forwarding of packets. If nodes all belong to the same managing entity (owner or administrative domain) they will surely cooperate. Examples of these networks are sensor networks or wireless ad-hoc backhaul networks: the owner of the sensor network will want it to be efficient so that it could have accurate measures and act in real time, and a backhaul wireless network also must be a highly controlled environment with maximum efficiency. On the other hand, in a multi user environment, selfish forwarding is harder to achieve. Users usually have different behaviours, thus taking preference on a different set of applications. Because coexistence of different services or simply expectations is not as peaceful as it should be, satisfaction level may drop. If a user expects to use the ad-hoc network to download files from other nodes at X kbits/s, but only achieves $X/2$ kbits/s, because if it is forwarding Y kbits/s, it may decide to simply stop forwarding. Another situation may arise if a node is becoming really short on battery or all available CPU is required to perform a user task. Unless the network offers something to the user himself, it may become unsatisfied and eventually

start acting selfishly. This “compensation” may simply be the feeling of belonging to a particular group, such as an ad-hoc composed by a group of friends. Unfortunately this type of rewarding is not often everywhere.

Scenarios considering heterogeneous users motivated researchers to address the problem of cooperation. Several studies were performed addressing the problematic of cooperation and selfish behaviour in ad-hoc networks. Initially they started with simple methods to detect non-cooperating nodes and exclude them from the network [michiardi00], [buechegger02]. In these proposals, the concepts of trust and reputation were introduced, where the actions of users were monitored. Later research studies started applying gaming theory [fang04] in the analysis of the forwarding process. While reputation and trust was used to enforce cooperation, another set of proposals relied on virtual currency (which can be directly related to money or some profit related to the services that can be accessed, resources in the network, etc.). Virtual currency methods such as the ones proposed at [weyland04], [blazevic01], [chen04b] enable the existence of a real distributed algorithm without the necessity of a common point of trust. Nodes are able to exchange currency directly without the intervention of an additional entity. Distributed ad-hoc environments are fully supported using these proposals. Problems like credit starvation and the necessity of tamper resistant modules appear, making its deployment difficult. Also, operator driven scenarios are not well supported due to the lack of control over credit, fraud and session establishment. Examples of centralised proposals include [zhong03], [lamparter03] and [saalem06] which consider the existence of an operator managing the network. In these proposals, the operator is common and trustworthy to all participants. Such proposals are specially suited for scenarios considering operator driven environments.

In the following sections an overview of relevant work involving cooperation either by punishment or by rewarding will be presented. Proposals to promote cooperation by credit rewarding are the main focus of this thesis and will be addresses with more detail.

3.1 Cooperation by punishment or reputation

3.1.1 CONFIDANT

Buchegger and Boudec propose a solution [buchegger02] named CONFIDANT (Cooperation of Nodes: Fairness in Ad-hoc Networks) which makes selfish misbehaviour unattractive to users. The mechanisms proposed work by detecting misbehaviour and isolating the offending node from the network. CONFIDANT specifies four different entities: the monitor, the trust manager, the reputation system and the path manager.

Monitors will listen and analyse communications performed by all 1hop neighbour nodes. Case misbehaviour is found, an ALARM message is sent to the trust manager. To minimise false positives, a single misbehaviour does not trigger an alarm. Instead the node must detect violations repeatedly and will trigger the ALARM if they exceed a configured threshold. The trust manager receives ALARM messages and decides whether further routing information is accepted for processing or not depending on node trustfulness. A “friend list” is also kept at this module containing a list of nodes considered as friends and which will be notified of anomalies using ALARM messages. An ALARM message will not necessarily block a node as it may be discarded if the trust manager module assumes the sender of the ALARM message itself is not trustworthy (e.g. does not belong to the friends list). The Reputation System tracks known nodes and keeps a reputation rating about each node. If enough evidence is received indicating a node is cheating, its trust level will be decreased. Reputation lists are local to each node but can be exchanged between friends in order to fasten reputation convergence. The Path Manager module will choose routes avoiding nodes with low reputation or already excluded from the network.

CONFIDANT proposes the existence of first hand and second hand reputation. Such mechanisms are vulnerable to liars which falsely accuse other nodes of misbehaving and was further addressed in [buchegger03]. The first is collected directly by the node while the second results from ALARM messages sent by friend nodes. Only if the trust of the sending node is acceptable, and the reported information is according to eventual first hand reputation measures, the second hand table will be updated. Adding these mechanisms minimises problems related with false reputation information, which make CONFIDANT resistant against direct attacks to the reputation system.

This solution is not very adequate to low resource (mainly battery) environments

like sensor networks, since it adds more processing to each node (related to monitoring of the surrounding wireless environment).

3.1.2 Context Aware Detection

In [paul02] proposal Paul and Westhoff describe mechanisms capable of detecting node misbehaviour in a wide range of situations. More importantly, they allow identification and propagation of a specific misbehaviour of a node. It is assumed all nodes are promiscuously listening to the wireless medium and networks have medium to high concentration of nodes. A tamper resistant module obliges MAC and IP addresses to be linked and cannot be changed by user intervention. DSR should be used to provide routes, being the proposal highly linked with its the discovery and route maintenance processes.

When node A wishes to initiate communication with node B , it should discover a route to destination. Using DSR this is performed by broadcasting a *RREQ* message indicating the destination node B . Each neighbour node receiving the *RREQ* message, adds its IP address to the previous packet and rebroadcasts it. [paul02] proposes securing the route discovery process of DSR by adding a sequence number and a Hash to every *RREQ* message sent. The sequence number is randomly chosen to each new *RREQ* while the Hash is the result of a digest computed over IP_A , IP_B and the sequence number. Each forwarding node, before forwarding the message must replace the Hash by a new digest computed over the concatenation of the previous digest with its IP address. Consecutively calculation of digests will create a Hash Chain. The resulting value can be later verified using the list of IP addresses added, and the sequence number included in the packet. When B receives the *RREQ* it checks its validity by recalculating all values and comparing the result with the Hash presented. If the values match, B can be sure no node manipulated the route presented and a *RREP* message is sent. This however allows a node to add additional entries to the list; only changing previous values is prevented. In order to prevent this attack every node listens and caches every *RREQ* packet received by its neighbours N_i . After messages are processed and forwarded, neighbours analyse the response and verify if node N_i only manipulated the packet by adding its IP address and updating the Hash. Adding more than one address or adding an erroneous address is easily detected by neighbours, due to the existing mapping between MAC and IP addresses.

When an attack to the address list is detected, each node detecting the attack should inform both neighbours and *A*. This is performed by broadcasting a *SECM* message to all neighbours. Upon reception of such message, nodes compare the alarm with its cache. If they know (have in cache) the packet accused of being forged they send a *SECM* message toward *A* including both the original content (cached packet) and the forged data (packet manipulated by attacker). Upon reception of *SECM* messages, node *A* can identify if they are correlated to the same event (*RREQ*) and time frame. If this is true and a reasonable number of nodes agree on the accusation (more than 4) the node is considered a culprit and is avoided. In some situations, Paul et al consider culprit a node issuing a *SECM* if it is the only accuser. This is valid for high density environments where number of neighbours is expected to be between 8 and 10.

Dropping *RREQ* messages remains the only attack which can be performed by a forwarding node. Also, due to asymmetry in radio range, this attack cannot be easily distinguished between low card sensitivity or malfunction.

3.1.3 OCEAN

The Observation-based Cooperation Enforcement in Ad-hoc Networks (OCEAN) [bansal03] was proposed by Bansal et al, as a solution to increase cooperation a MANET. In OCEAN nodes monitor others' behaviour but in contrary to CONFIDANT, they do not exchange collected information. Because no second hand information is used, OCEAN does not suffer the problem of false accusing a node of misbehaving. In CONFIDANT this could happen either by incorrect detection or by a malicious attempt to exclude a node from the network. The drawback is the same node will have different reputation information at all neighbour nodes. Also, if a node blocks an offending neighbour, it is not guaranteed common neighbours will reach the same conclusion and the offending node could still be active in the network.

OCEAN proposes each node must classify every neighbour in a rank based on observation of messages exchanged and packets forwarded. The starting value is 0 (Neutral) which is increased by 1 at every positive action and decreased by 2 at every negative action. After the value reaches a negative threshold (-40), the neighbour is considered as being faulty. Faulty nodes are added to a black list and stay there until the expiration of a timeout. After the expected time, messages are processed and the offending node has the possibility to correct its rank. Because rank value is not set at

a different value when a node is removed from the black list, one single negative action will be enough to add it to the list until the next timeout.

This method of giving an opportunity to neighbours to redeem is important in order to proper operation of a network; OCEAN is not perfect in judging neighbours and in some conditions it can indicate a neighbour is misbehaving when that is not the case; users can change their behaviour and start cooperating if they realise they have no advantages by being selfish. Without this periodic check, nodes would be banned permanently from the network. Eventually, the entire network could be banned and no communication would be possible.

Calculating routes based on reputation is performed by adding an extension to DSR *RREQ* messages. This extension contains a list of nodes which should be avoided in the route. If a node receives a *RREQ* to a destination he knows the route, and no hop exists in the avoid list, a *RREP* message can be sent. If such node exists, the *RREQ* packet can be re-broadcasted or silently dropped depending on the avoid list and the local faulty list. Following this concept, when a *RREP* message is received, the requesting node also checks if any node in the route is present in the current faulty list. If this is true, the *RREP* message is discarded. Although the authors of OCEAN only propose the usage of DSR, it is possible to adapt similar concepts to other routing protocols such as AODV or OLSR.

3.2 Cooperation by Credit Incentives

3.2.1 CASHNet

The authors of CASHnet [weyland04] presented a solution allowing operators to deploy a pre-paid scheme in ad-hoc integrated networks. For deployment, it requires the existence of a cheat proof hardware module. This could be easily implemented as a SIM card similar to those currently used in GSM networks.

CASHnet assumes the existence of two types of virtual currencies which can be exchanged in a virtual bank. The first are called *Traffic Credits* and the second *Helper Credits*. Each node joining the network registers with the operator sending its identification, and public key. The operator will cache the public key for latter verification of reported *Traffic Credits*. After the registration is performed, node requires *Traffic Credits* so that other users allow him to send packets. It can pre-buy some currency

in the bank using real money, or it can stay silent forwarding others packets. Each sent packet contains a *Traffic Credit* token which can be collected by forwarding nodes becoming a *Helper Credit*. Later on, nodes can deliver *Helper Credits* to the bank receiving some *Traffic Credits* back. The ratio used in the exchange must be carefully chosen by the operator so that users feel rewarded and operator still gets enough profit. Packets also contain two independent message authentication codes signing the packet. One belongs to the sending node, while other belongs to the previous forwarding node.

In order to verify the authenticity of data, each node must verify two signatures upon forwarding a packet. Usually the verification operation would make this solution very resource hungry. Due to the usage of dedicated hardware (required by CASHnet), the operation would take low number of cycles in the main CPU, thus optimising the process. Also nodes must exchange and cache certificates from neighbour nodes, as well as from sender nodes. In CASHnet, certificates have low lifetimes obliging nodes to synchronise with the operator frequently. A low validity will reduce the granularity of contract expiration or of usage control. The drawback will be the higher exchange of certificates between nodes.

3.2.2 A Charging and Rewarding Scheme for Packet Forwarding in Multi-hop Cellular Networks

A Charging and Rewarding Scheme for Packet Forwarding in Multi-hop Cellular Networks was first proposed in [saalem03] and then at [saalem06] for multi-hop extended cellular networks. Such networks are heavily influenced by current cellular technologies, like GSM or UMTS, where a base station is directly connected to all terminals. The AP, being trusted by the network provider, and also by the nodes, is capable of enforcing individual user profiles and account for all traffic produced. Although the authors consider the multi-hop scenario, traffic is always obliged to cross the Access Point (AP) independently of the locations of source and destination nodes. If the traffic is only to or from the outside of the network, thus crossing the Access Point, this solution should be very efficient. However, if it is considered direct traffic between the nodes in the ad-hoc network, the sub-optimal routing will have a dramatic impact in performance. Also, in all cases, the maximum bandwidth of the entire network is limited to the bandwidth of the wireless medium available to the Access Point.

According to [saalem03], when a node wants to send a packet, first it must create

a session to the AP (also gateway) by means of a *SessionSetupRequest* message (Figure 3.1). The gateway then creates a session with the destination node, or signals another AP to start the session, in the case the destination node is outside the local ad-hoc network. The request message includes a *SessionID*, an *OldSessionID*, a *TrafficInfo* with a description of the flow and a *MAC* signing the entire packet. Each intermediate node checks the *TrafficInfo* field and decides if the flow should be accepted. In this case, the *MAC* is updated and the packet forwarded.

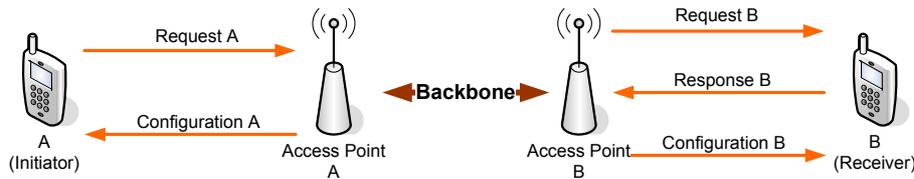


Figure 3.1: Control signalling of a session establishment

The destination node should also check the *TrafficInfo*. If the flow is accepted it will reply with a *SessionSetupResponse* message to the Access Point containing the result of the request, the *Route* and a keyed *MAC* computed over the Request message. The AP, upon reception of the *SessionSetupResponse*, then starts the charging procedure and sends a *SessionSetupConfiguration* to both the sending and destination nodes. It should be noticed that a session is really composed by two sub-sessions: the first session is established between source node and AP, and between AP and destination node. Moreover, the sending node is charged by the traffic in both directions. After the sending node receives the *SessionSetupConfiguration*, it starts sending packets to the AP. These packets have a header indicating the *SessionID* negotiated during the setup phase, a sequence number and a keyed *MAC*. The payload is also ciphered using a stream cipher, first by the sending node and then by all forwarding nodes. Ciphering the entire payload by all forwarding nodes is a bandwidth efficient method of certifying the packet was forwarded by the correct nodes. The Access Point, knowing the nodes belonging to the session will decipher the packets in the reverse order. If the packet was forwarded by a different node than the ones contained at the list established during the session setup, deciphering will result in garbage and the packet is lost.

Since the list of forwarding nodes is established during the setup phase, if the route changes a new sub-session must be created. The node detecting a change in the route should send a *BrokenSessionError* message to the sending node. Upon reception of a *BrokenSessionError*, the sending node verifies if the reporting *SessionID* is a valid one,

and restarts the sub-session. If a node receives a packet to forward and the *SessionID* is unknown, a *UnknownSessionIDError* message is sent requesting the creation of the sub-session. Recreating the sub-session is similar to the creation process, however the *SessionID* in the *SessionSetupRequest* will contain the value of the existing session. Moreover the session re-created is only the one between the source node and the AP, or the AP and the destination node, depending in which segment the route changed. During the forwarding, even if the session is valid, any node can notify the sending node he does not want to participate. This can be either because its battery is very low, or simply he does not want to participate. Such node should send a *CancellationError* message to the sender. This also triggers the re-establishment of the sub-session, but the cancelling node can now be excluded of the forwarding from the start.

Another aspect of this proposal is the support for destination acknowledgement. To acknowledge the reception of data packets, the receiving node should keep a list of all *Sequence Numbers* received. Because the values are sequential, a missing value will indicate a missing packet. After a timeout or sufficient number of losses, the receiving node should transmit to the AP the missing values. As explained at [saalem06] there is no advantage to the receiving node to not acknowledge the reception. Charging is done before packets reach the destination, this information is only used to identify which packets were forwarded and issue rewards. Although there are no proofs returned to the Access Point from the receiving node, the acknowledge mechanism allows to correctly charge users in the second sub-session. Nodes of the first sub-session are clearly identified by the Access Point without any additional mechanism. Because all traffic must cross the Access Point, operators have a great control over the packets forwarded. The cost of such control is higher bandwidth usage and higher delay, due to sub-optimal routing, for traffic with both end-points inside the ad-hoc network.

The method used to create sessions is very vulnerable to user colluding in order to gain additional reward credits. When a packet is in the setup phase, a malicious node can easily add the identifier of all of the nodes colluding instead of adding only its identifier. When this node is forwarding a data packet, instead of ciphering the payload with its secret, it will need to cipher it using the secrets of the nodes colluding. The Access Point will have no way of identifying this fake forwarding and will reward them appropriately. If all nodes colluding perform the same operation it is clear that the group will have more profit than expected by the operator. This attack poses a serious threat to the charging process and no practical solutions is available to avoid

it. The authors propose the usage of statistical analysis to detect nodes emulating the forwarding process, but this can be avoided by careful manipulation by the malicious nodes.

Other serious shortcoming of this solution is control messages are not authenticated. Any node can perform a DoS attack on the communication between two endpoints by sending a *CancellationError*, *BrokenRouteError* or *UnknownSessionIDError*. Everytime the sending node receives one of these packets, and the *SessionID* is valid, it will block all packets and re-establish the session. As the *SessionID* can be easily listened from any node neighbour to a node forwarding the flow, this attack is very easy to execute yet, very disruptive.

3.2.3 SPRITE

One of the first solutions providing charging in ad-hoc networks without direct transactions between nodes was presented by Zhong et al as SPRITE: A Simple, Cheat-Proof, Credit-Based System for Mobile Ad-Hoc Networks [zhong03]. SPRITE considers the existence of a *Credit Clearance System* where nodes can pre-buy credits. It also requires a Public Key Infrastructure available. Nodes will have a private and public keys and a certificate, issued by a trusted partner. All packets are authenticated using these keys directly by means of a MAC and nodes can verify if a packet received for forwarding is from a authorised node or not. Because nodes will only be rewarded by forwarding a valid packet from an authorised node, if the verification fails packet is silently dropped. It should be noticed, nodes are expected to have enough processing resources in order to compute key verifications in real time which will require dedicated hardware in most ad-hoc devices. However, and in contrast with concurrent proposals, SPRITE does not require a safe and trusted environment to run the SPRITE algorithm.

The authors present this solution integrated with a Source Based routing protocol, namely DSR. They also refer the possibility to charge multicast traffic. However no description of such interaction was presented in the original publication. When a node wants to send a packet, and after it has a route to the destination, it adds the full route, a sequence number and the signature of a digest calculated over the message payload, the sequence number and the route. The route is encoded by simply concatenating all IP addresses. The digest will be the start of a hash chain. When a node receives a packet to forward, it performs a number of checks. If any of these checks fails, the

packet is dropped. First the node checks if it already is in the route present in the packet. If this is false, there will be no reward for the node to forward the packet. In mobile networks, route can and will change often. If route changes, nodes will refuse to forward a packet without their address included in the route and will silently drop the packet. The undesirable result will be a higher probability loss under high mobility leading to a low performance under those scenarios. The next step is to verify if the sequence number is higher than the last sequence number received from the source node. In the final check the signature is verified assuring the packet was not manipulated. If all these steps are successful, the forwarding node will store the new sequence number and update the *Hash Chain*. Before the packet is forwarded to the next hop, each node will collect the *Hash Chain* in the packet, and store it on the nodes permanent storage medium like a flash drive or a hard disk. Finally the packet is then forwarded to the next hop in the route.

When a node is connected to a more resourceful network such like a LAN or a DSL access, it should deliver the collected digests to the Credit Clearance System. Based on the digests received, the Credit Clearance System, upon correct verification of the cryptographic keys, will charge initiators and reward the node reporting information.

According to SPRITE, session initiators are only charged when forwarding nodes deliver digests back to the *Charging Clearance System*. If they never deliver the proofs they will not be rewarded, however they will also not be charged. Although this attack is possible, it is not simple to implement. If a single node report a digest, the node will be charged. Assuring all nodes in the route are colluding, specially in a mobile environment, is not trivial. Also, the price delivered to forwarding nodes will be less than the price paid by the initiator. This is because some profit must return to the network operator owning the bank. If nodes do not collude, the system will be efficient. However, if the user initiating the connection has control or is associated with one or more nodes along the path, it may choose to not deliver proofs because it will receive less money than it will be charged of.

The solution provides an interesting application of cryptographic methods. However, because all nodes must verify, in real time, the integrity of packets, this may pose serious performance constraints. The alternatives to increased delay will result in increased overhead. The fact that all nodes store digests of all packets forwarded also poses some concerns. Considering a typical digest, size between 16 (MD5 [rivest92]) and 20 bytes (SHA1 [eastlake01]), storing one digest per packet will require significant

storage, hard to handle in PDAs or low-resource laptops. Reporting all the digest captured will also make mandatory the existence of higher bandwidth links. These issues make SPRITE only usable in a very small number of situations and on very specific scenarios.

3.2.4 SCP

The Secure Charging Protocol [lamparter03] is an AAAC protocol which improves the ideas first proposed by SPRITE [zhong03] and extends them by incorporating a more complete set of functions. SCP focus on securely retrieving accounting information for traffic flows, while retaining relevant information associated with the routes that were taken by each of the packets, in a iterative, efficient and online manner. Moreover, it operates in a partially distributed environment where nodes do not require to periodically having available a high bandwidth connection to the operator (like in SPRITE). A vital component of SCP is the existence of a centralised AAAC infrastructure under control of the operator managing the network. Besides storing all information regarding business models, users and credit, this system provides mechanisms to control the access of users to the network and the return of traffic information. In the case of SPRITE, the *Credit Clearance System* only provided functions to allow returning the traffic proofs and no active access control was deployed. In SCP, the Access Router, still owned by the operator, interfaces the infrastructure network with the ad-hoc cloud. It serves as a proxy between nodes and the central AAAC infrastructure deployed at the Core of the network and can actively authorise or block access to services after the user is authenticated into the network. Communication between nodes and the Access Router is performed using SCP, while communication inside the operator network is performed using standard protocols such as DIAMETER [calhoun03] or RADIUS [rigney00]. The usage of such protocols also makes SCP able to be simpler to introduce on existing networks where such systems should already exist.

SCP assumes each node is operated by a user and the user must have a contract with the operator managing the AAAC. From this contract results a credit profile, a pair of symmetric keys and a shared secret. The credit profile will have the guidelines for translating traffic information in real currency deduction or increase according to the business model and nodes' role. Cryptographic material is required to avoid attacks such as impersonation of a user, free-riding others' packets or faking control messages and it is an important component of SCP. Each node has a private key used to sign all

control messages and a public key, presented in a certificate signed by the operator. The shared secret is used to perform functions internal to the protocol and later described.

SCP is divided in three separate phases: Registration, Data Transfer and Accounting. These phases are depicted in Figure 3.2.

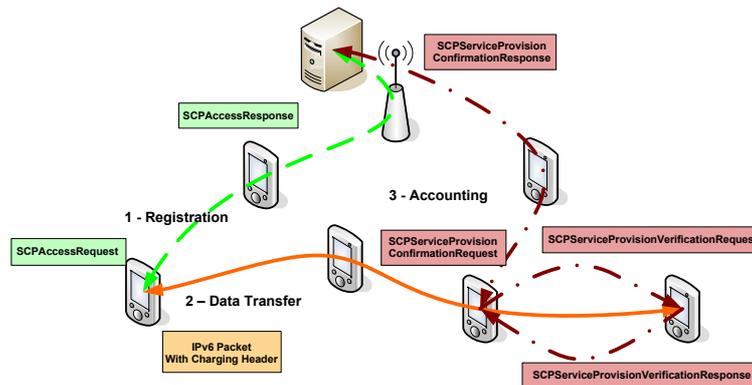


Figure 3.2: Diagram representing the different phases of the SCP proposal

The first phase is the Registration phase and occurs when a node wishes to participate on an ad-hoc network. Because no traffic without a valid signature and certificate is allowed on the network, nodes must first register with the network operator. When a node wants to join an existing network, it sends a *SCPAccessRequest* message to the local Access Router (AR) with its user identification and a common secret. The information provided is verified with the AAAC server and, case the verification succeeds, a *SCPAccessResponse* message is sent back. The response contains a private key, a signed certificate with a public key, a shared secret and some charging information. Upon reception of a successful response, the node has all required credentials to initiate a session to a destination node.

In the Data Transfer phase, an already registered node (denominated the Initiator) wishes to communicate either with a node inside the network or with a node located in a foreign network. The Initiator should intercept all packets going out of its ad-hoc enabled interfaces and add a charging header to each. Headers are required for operation of the distributed charging mechanism and nodes are obliged to add them. Packets without the charging header will be dropped by forwarding nodes. Authors of SCP only considered IPv6 and proposed adding headers as a Hop-by-Hop extension [deering98] making this proposal independent of the transport protocol used (at least at this phase). The charging header contains a *Header Type*, an optional *Route List*, a *Sequence Number*, a *Hash Chain* and a *Message Authentication Code (MAC)*.

SCP assumes the possibility of operating with Source Routing (SR) based protocols such as DSR or with other routing protocols such as AODV or OLSR. When used with the DSR protocol, it exploits the route already contained in every packet and incorporates this information in the charging header added to packets. If the routing protocol does not include a route header, SCP authors further extended [girão04] the original proposal by adding a method to iteratively build the route supporting on-demand protocols. Instead of including the full route at the sending node and because no route is added by the routing protocol, each node forwarding a packet, adds its address to the charging header before sending or forwarding it. This behaviour brings some advantages such as higher independence from the routing protocol used and lower charging overhead, especially with long routes. On the other hand, it forces nodes to fragment packets more often and to adjust upper layer protocols, such as TCP, upon each forward. The type of charging header added (with iterative route or with full route) is set using the *Header Type* field.

In order to avoid replay attacks, SCP makes use of a 32 bits long *Sequence Number*. This field is set with a random value upon initiation of a new flow and is increased upon each new packet sent. If a forwarding node detects a *Sequence Number* with a value lower than the last *Sequence Number* received in the same flow, the packet is considered as part of a replay attack and is silently dropped.

The *Hash Chain* is a digest calculated over a pseudo header using cryptographically graded digest algorithms such as MD5 or SHA-1 and has a length of 128 bits. This field is required so that the AAAC, upon reception of traffic proofs, can validate the route presented and reward the users which really forwarded the packet. When the Initiator adds the charging header, it initialises the *Hash Chain* by concatenating the *Sequence Number*, its IPv6 address, packets' *Traffic Class* and the personal *Shared Secret* and calculating the digest. The less significant 128 bits are then copied to the charging header. This can be expressed in equation 3.1 where || denotes concatenation of fields:

$$HashChain_0 = H(SeqN||IPv6Address||IPv6TC||SharedSecret) \quad (3.1)$$

Forwarding nodes process the *Hash Chain* and update it almost like the sending node. However, because adding a new digest at each forwarding node would create too much signalling overhead, the digest is calculated using an iterative manner. The *Hash*

Chain which is already present in the charging header is concatenated to the same fields used by the Initiator and a new digest is calculated. Although this new digest is completely different from the value present in the packet, because it was created using the previous value, it will be useful in authenticating forwarding nodes. The result is then added to the header replacing the previous value.

$$\text{HashChain}_n = H(\text{SeqN}||\text{IPv6Address}||\text{IPv6TC}||\text{SharedSecret}_n||\text{HashChain}_{n-1}) \quad (3.2)$$

The *MAC* field on the charging header is of variable size and contains the result of a sign operation performed by the Initiator using its private key. In SCP control messages, the ECDSA [ansix9.62] algorithm is often used to create and verify the *MAC*. When a node receives a packet, either because it is the destination node or a forwarding node, the *MAC* field must be verified, validating the data transmitted. If the correspondent public key is already cached and is still valid, it is used to verify the *MAC*. If this is not true, before the verification can be performed, the public key must be obtained by sending a *SCPCertificateRequest* message to the source IPv6 address. When a *SCPCertificateRequest* message is received, the certificate issued upon registration, is returned using a *SCPCertificateResponse* message. Upon reception of the certificate, the node is able to both verify the authenticity of the message, and the authorisation of the node to send a packet, by checking the validity of the certificate.

When a node is forwarding a packet, besides verifying the *MAC*, updating the *Hash Chain* and eventually (depending on the routing protocol used) adding its IPv6 address, it also checks the distance to the destination node. This is achieved by consulting the routing protocol itself or the routing tables in the operating system. If the forwarding node determines that the destination node is a neighbour and that it will be the next hop of the path, a snapshot of the packet is stored locally. Source and destination addresses, *Traffic Class*, *Packet Size*, *Sequence Number*, *Route* and *Hash Chain* are some of the values captured from the forwarded packet. If a second packet is forwarded for the same flow, only the *Packet Size*, *Sequence Number* and *Hash Chain* are recorded. After a number of snapshots are captured (or a pre-configured timeout) a *SCPServiceProvisionVerificationRequest* (*SPVReq*) message is assembled and sent toward the destination node. This message contains information about the session such as the session end-points, the route and the number of packets and bytes delivered,

(since the last *SPVReq* sent) related to the same flow. Upon reception of a *SPVReq*, the destination node checks the validity of both the message and session and verifies the amount of packets and bytes received. If less packets and bytes were received than present in the *SPVReq*, the internal counters reset. However if the internal counters have higher values, they are decreased for the amount present in the *SPVReq* message. In both cases, the total decrease of the counters is acknowledged using a *SCPServiceProvisionVerificationResponse* (*SPVResp*) message. The last forwarding node, which previously issued a *SPVReq* receives the *SPVResp* and builds a *SCPServiceProvisionConfirmationRequest* (*SPCReq*) towards the Access Router. This message contains information about the flow (source and destination addresses), the route and a proof representing each packet forwarded. The proof contains the values of each *Hash Chain*, *Traffic Class*, *Sequence Number* and *Packet Size* captured. A *SCPServiceProvisionConfirmationResponse* (*SPCResp*) acknowledges the reception of the proofs and is able to additionally issue an order such as allowing or blocking further communications from the Initiator. Contents of the *SPCReq* message are validated, first by verifying the *MAC* and then by computing the *Hash Chain* according to the reported route and additional fields such as each node shared secret. If the computed values for each *Hash Chain* are consistent with the values reported, the Initiator is charged while the forwarding nodes are rewarded. If the verification fails, some error or fraud attempt took place and no action is performed (besides eventually blocking the Initiator).

The authors of SCP described with detail the most appropriate business models usable in the scenarios presented. Particularly since they presented a very complete work describing what should be the relation between the credit paid to a consumer node and the credit paid to a forwarding node, yet yielding profit for the operator. A set of parameters denominated $P+$ and $P-$, representing the amount of data sent or forwarded, are used to calculate a ratio which will influence the amount of credits paid by users. This ratio reflects the amount of traffic a node helps forwarding versus the amount of traffic produced, thus revealing users behaviour.

Chapter 4

Proposed Solutions

Based on the proposals discussed in the last chapters, two proposals for secure charging protocol have been developed: the Polynomial-assisted Ad-hoc Charging Protocol and the Session-aware Ad-hoc Charging Protocol.

4.1 PACP - Polynomial-assisted Ad-hoc Charging Protocol

The Polynomial-assisted Ad-hoc Charging Protocol aims at providing charging, rewarding, access control and flow admission functionalities, in ad-hoc integrated networks. The scenarios envisioned by PACP consider a MANET interconnected to the infrastructure network managed by a network provider. Such scenario consists of an extended hotspot environment, where one (or more) ad-hoc nodes are connected to a wireless hotspot, and this interconnection capability is shared to all nodes in the ad-hoc network. Note that these scenarios have wider applications scope than this extended hotspot. Other scenarios envisioned are the ones where infrastructure networks require methods for distributed charging, rewarding and flow control mechanisms. It also copes with eventual disconnections to the infrastructure network: the ad-hoc nodes are able to store (cache) a significant amount of charging information. Upon reconnection to the infrastructure that information would then be reported. The PACP protocol is based on SCP concepts, but improves manifold over most proposals in the literature [saalem06], [zhong03], [lamparter03]: it includes the capability for flow admission without the need of sub-optimal routes, the network overhead introduced by the protocol is reduced and

the processing requirements in every node are low. Moreover, the complexity of the forwarding process is low due to the use of polynomial encoding of the intermediate nodes.

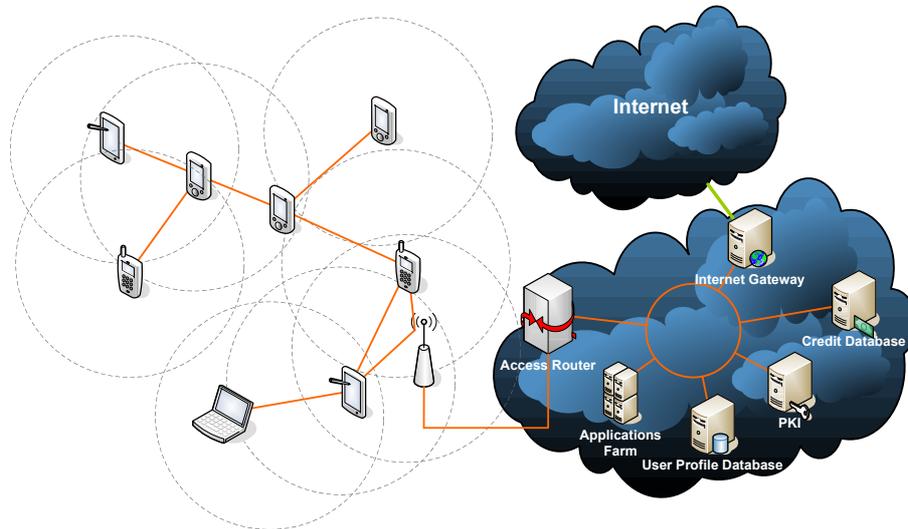


Figure 4.1: Ad-hoc integrated network

Figure 4.1 depicts the architecture of an ad-hoc network in the extended hotspot scenario. Inside the ad-hoc network, the traffic is routed through reactive or proactive ad-hoc routing protocols, like AODV, OLSR or DSR. The ad-hoc network is connected to the infrastructure network through an Access Router (AR). This element is a node (often infrastructure) that routes packets between the external networks and the ad-hoc cloud, collects the charging proofs, both for rewarding and charging, and sends them to a AAAC server in the infrastructure network. The AAAC server handles all authorisation, authentication and charging issues: it receives the proofs containing information on the senders, receivers and forwarding nodes, and processes the charging and rewarding information. Since the PACP protocol is secure, the ad-hoc nodes need to maintain cryptographic material to be able to send, forward and receive packets. This key management is provided by a Public Key Infrastructure (PKI). The ad-hoc nodes can access operator services available in the infrastructure network (located at an Application Garden), and can access any node in the Internet through the Internet Gateway. The rewarding mechanism operates under the assumption that, if the nodes between sender and receivers do not have a benefit to forward a packet, they will start acting selfishly and drop packets, instead of forwarding them. This behaviour is undesirable both for users, located far from the access point, and for network operators.

The first will be unable to communicate with other nodes, the second will have less users consuming their services. However, if nodes are rewarded for the packets they forward, they will become interested to cooperate in the ad-hoc cloud. To reward the forwarding nodes, the operator, through the AAAC, needs to have information on the paths crossed by each packet in the ad-hoc network. This is one of the main challenges of any rewarding protocol: to implicitly (with low overhead), and securely carry and transport information required for the correct charging and rewarding process. Notice that security is of major importance, since it is required to assure that the ad-hoc nodes report the correct ad-hoc paths to the AAAC. Without security issues taken in consideration this problem would be much simpler.

Briefly, the PACP works as follows (see Figure 4.2). When a node enters in the ad-hoc network it goes through a registration process to become known by the AR (and the AAAC), to be authenticated and authorised, and to receive the cryptographic material required for the charging process (Registration phase). When communicating with other nodes inside or outside the ad-hoc network, the node implicitly (through polynomial encoding) includes in every data packet the identification of the route, which will be updated at each node as the packet is forwarded. The fields containing information on the route are fixed size and cryptographically secured, so they cannot be wrongly modified by malicious nodes (Participation and Forwarding phases). The node belonging to the flows' path, one hop way from the receiver, denoted as the last forwarding node, is responsible for sending the proofs to the AR (Reporting phase). These proofs implicitly contain information on the path(s) of the flow. They are sent to the AR when the number of proofs collected at the node reaches a specific number, or when a timeout expires. Notice that, in the case of traffic with destination outside the ad-hoc network, the last forwarding node is replaced by the AR. When receiving the proofs, the AR sends them to the AAAC to verify the truthfulness of the information, through the cryptographic information contained in the proofs, and retrieves the information of the ad-hoc route (Verification phase). The AAAC is then able to correctly credit the sending (and, eventually, the receiving) nodes, as well as correctly reward the forwarding nodes. In the next sub-sections it will be detailed the security basics required for the protocol operation and the several phases of the protocol. The messages and contents depicted in 4.2 will be detailed in the following sub-sections.

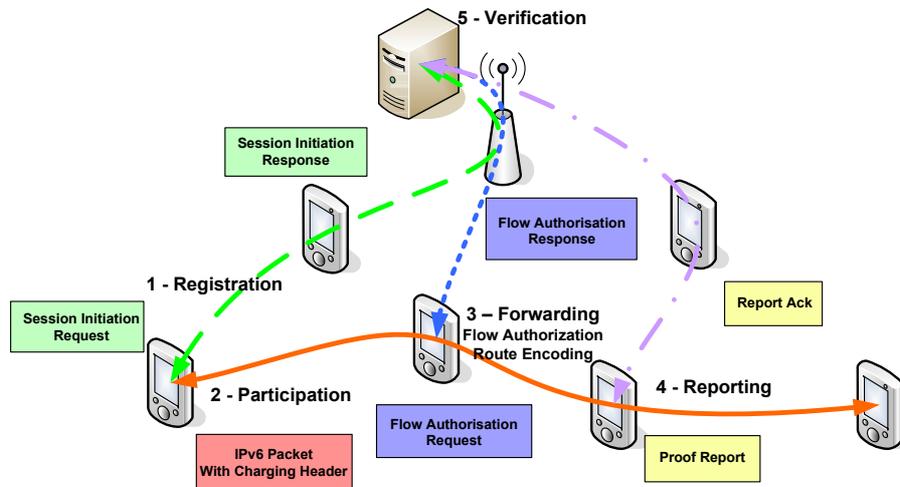


Figure 4.2: Different phases and messages of the PACP proposal

Security Assumptions

It is assumed all packets contain a Message Authentication Code (*MAC*) used to verify the integrity and ownership of the packets. The network provider has a public key, that can be used by the ad-hoc nodes, both to authenticate packets from the AR, and to protect control packets sent to the network operator.

Because of the resource limitations of nodes composing a typical ad-hoc network, the cryptographic algorithm and duration of the associated keys need to be carefully evaluated. The algorithm needs to have a very high security per bit ratio and should be easily implemented either in dedicated hardware or in software (Elliptic Curve Cryptography [koblitz87] has been the algorithm of choice when dealing with low power equipments and low bandwidth environments [ingo03]). We consider the key pair belonging to the operator to be valid for a long period. These keys should be strong enough to avoid impersonation of the trusted equipments belonging to the provider. Compromising such key can bring many problems to the operator and users.

On the other hand, the keys associated to mobile nodes are only needed during the time the node is connected to the network, and thus the key validity can be small, also leading to shorter keys. Moreover it is considered nodes should generate new keys periodically, avoiding the disclosure of too much ciphered material. The use of short keys minimises the overall overhead introduced by the message authentication code. Also, the delays associated with cryptographic verification and encryption are smaller, increasing network performance, battery life and available processing resources

of mobile nodes. All equipments participating in the ad-hoc network are operated by one or more users, each user having a user profile result of a previous agreement with the operator. The profile is composed by identification data, public and private keys, a charging profile and a traffic profile. The user profile should be fully known to the user and partially to the network operator AAAC (the operator cannot access user private key). Based on these assumptions, the PACP mechanism defines the five previously referred phases that compose the charging and rewarding process.

Registration Phase

When a mobile node joins an ad-hoc network, it needs to be authenticated and authorised in the network before being able to communicate and access the services available. The AAAC is the element responsible for this authentication and authorisation process. Communication in the registration process is performed through the AR. To start the registration phase, the ad-hoc node sends a *Session Request* message to the AR providing its authentication data and public key. The sensitive fields of this message are protected with a stream cipher. The cipher key is ciphered using the network operator public key and the entire packet signed using the nodes private key. The AR replies with a *Session Initiation* message indicating if the node is allowed to participate in the network. In the case of a positive answer, the nodes' public key is stored locally and a shared secret is generated and sent in the *Session Initiation* message with the resulting code and configuration objects. The payload of this message needs to be protected, using the same stream cipher and key used by the registering node. The configuration objects include a timestamp stating the validity of the registration. After the specified timeout, user must repeat the process. Optionally it may also generate a new pair of keys.

Generating keys periodically is important to avoid disclosure of too much cryptographic data and should be performed by all nodes. However, if nodes feel confident about their security they may opt to maintain their key during several registration processes. Moreover, each node is free to choose the key size used. More powerful nodes will be able to operate with larger keys while others will opt to use smaller key, easier to process and weaker. Because the public key is provided to the operator upon the registration process, it may force the minimum accepted key size by rejecting registration attempts with weak keys.

After the registration process is complete, the node is ready to send packets and participate in the ad-hoc network. Notice that even if the node just wants to forward or receive traffic, first it must register with the AAAC.

Participation Phase

After a successful registration process, nodes have all required information enabling them to participate in the network. When sending a packet, nodes need to include a tracking header (see Figure 4.3) in all data packets sent. Control packets are essential to network operation; because they are not related to transfer of data between nodes, they are excluded of processing in the charging protocol. No charging or rewarding will thus result from sending or forwarding these packets. Part of the header will be used as a charging and rewarding proof and will carry the identification of the nodes in the route.

The tracking header is composed by: a control code (*Code*), a sequence number (*SeqN*), a route hash (*RHash*), a hop count (*HopC*), a route identifier (*RID*), an index (X_i), a hash chain (*HashChain*) and a message authentication code (*MAC*). If the sending node is directly connected to the receiving node, the *RHash*, *HopC*, *RID* and X_i fields, which are required in the tracking header for route identification, should be omitted to reduce control overhead.

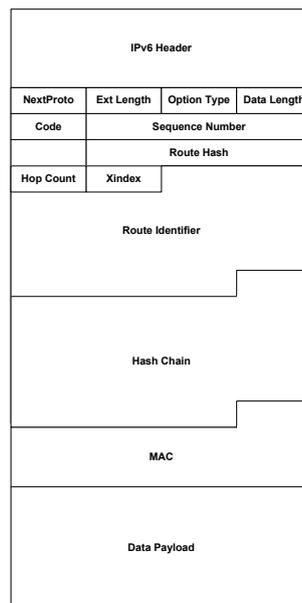


Figure 4.3: IPv6 packet with a PACP charging header

The *Code* field includes *External* and *Optimization* bits. The *External* bit, when 1, identifies the case where the endpoint is inside the ad-hoc network and the sending node is a node in an external network; the charging header is added by the AR to incoming packets. The *Optimization* bit, when 1, identifies the direct connection between sending and receiving nodes (remove of the route identification related objects). The *SeqN* field is used to avoid replay attacks and represents an identifier recently unique for each end-to-end session. It is initiated with a random value and incremented upon each packet sent. When all values are exhausted, previous sent values can be reused. The *RHash* field is a hash chain with 24 bits initially computed over the IPv6 addresses of the sending node, and interactively computed over the IPv6 address of each (and at each) forwarding node. The *HopC* represents the size of the route. *RHash* and *HopC* provide a rough mechanism to detect route changes. Although collisions can (and will) occur, this method provides some additional information helping the operator to reconstruct route information. *RID* is the field that contains the encoded route. This field represents values of a polynomial. It is updated at every node (see below) and will be used to reconstruct the path. X_i is the polynomial term and will also be used to, together with *RID*, reconstruct the path. The *HashChain* is a 128 bits value divided in 2 blocks of 64 bits. The first block is called *Charging Block* and the second one *Rewarding Block*. The sending node initialises the entire *HashChain* with the result of the *MD5* calculated over a pseudo header (concatenation of the following fields: source address, destination address, traffic class, protocol code, source port, *SeqN* and the nodes' shared secret). Other digest functions like SHA-1 can be optionally used, however all nodes must use the same function and packets without the tracking header or with an invalid MAC will be automatically discarded by forwarding nodes.

Forwarding Phase

When receiving a packet, each forwarding node needs to validate the *MAC* field, update the fields identifying the route and update the *HashChain*. Then, the packet can be forwarded. Because verification of signatures over ECDSA [ansix9.62] can be very time consuming, verification of the *MAC* field should be performed on new flows, and can be done otherwise periodically if no verification failed in the recent past. If a single verification fails, all packets should be verified and only after no verification fail occurs, nodes can start verifying packets selectively. Selective *MAC* verification allows for better performance at a cost of higher probability of forwarding fake packets

(which will result in no reward), if an attack is ongoing. Fake packets are not charged or rewarded because AAAC will detect the forgery by correlating the information returned. Forwarding nodes will update the left most bits (64 if *MD5* is considered, 80 if *SHA-1*) of the *HashChain* (corresponding to the rewarding block) with the less significant bits, resulting from the digest function, applied over a new pseudo header with the same structure as the one used by the sending node. Notice the values of this pseudo header are all the same but the shared secret, which corresponds to this forwarding nodes' shared secret. With this process, each node can assure its identity.

The route is encoded as points in a polynomial that are later reconstructed by the AAAC server. This encoding process allows the proofs to be small and with fixed size, reducing the overhead necessary to identify the route and decreasing the complexity associated to variable length packets. Making the proof size invariable of the route length is a major benefit as compared to other proposals. For example, in the case of SCP, the size of the packets is incremented by 16 bytes at each hop (length of the forwarding nodes' IPv6 address), thus requiring recalculation of the TCP checksum at every node, adjustments in the TCP MSS and IPv6 fragmentation. Other upper layer protocols may even require additional processing or just may be incompatible with SCP. Therefore, the invariability of the packet size implies a significant reduction in the operations associated with the forwarding process.

Route Encoding

Two sizes for the encoded route identifiers are defined: 64 and 128 bits. Identifiers with 64 bits are used to encode local subnet suffixes or user identification; identifiers with 128 bits can be used as globally valid identifiers or used to provide inter-operation with identity approaches like Host Identity Protocol (HIP) [moskowicz04], where the Host Identity Tag (*HIT*) is used.

IPv6 addresses as polynomials was first proposed at [dean02] to solve the IP traceback problem under Distributed Denial Of Service (*DDOS*). The idea behind the polynomial encoding is that for any polynomial $f(x)$ of degree d in the prime field $GF(p)$, with p being the smallest prime greater than $2d - 1$, it is possible to recover $f(x)$, given $f(x)$ evaluated at $d + 1$ unique points. If IP_i represents the IPv6 address of the i forwarding node, and X_i an unique packet identifier in the route R with n nodes, the AAAC Server can recover all the IPv6 addresses if it receives $Fr(x) =$

$IP_1x^{(n-1)} + IP_2x^{(n-2)} + \dots + IP_{(n-1)}x + IP_n$. Recovering this information is done by inverting a Vandermonde [press92] matrix under a prime field $GF(p)$ for a number of packets $d \geq n$. The complexity of the problem is $O(n^2)$ and it can be solved in real-time for the average route length of ad-hoc networks. The recovering process through the Vandermonde matrix is depicted in the set of inequalities 4.1.

$$\begin{bmatrix} X_1^{n-1} & \dots & X_1^3 & X_1^2 & X_1 & 1 \\ X_2^{n-1} & \dots & X_2^3 & X_2^2 & X_2 & 1 \\ X_3^{n-1} & \dots & X_3^3 & X_3^2 & X_3 & 1 \\ X_4^{n-1} & \dots & X_4^3 & X_4^2 & X_4 & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ X_n^{n-1} & \dots & X_n^3 & X_n^2 & X_n & 1 \end{bmatrix} \begin{bmatrix} IP_1 \\ IP_2 \\ IP_3 \\ IP_4 \\ \vdots \\ IP_n \end{bmatrix} = \begin{bmatrix} Fr(X_1) \\ Fr(X_2) \\ Fr(X_3) \\ Fr(X_4) \\ \vdots \\ Fr(X_n) \end{bmatrix} \quad (4.1)$$

Following the proposed approach, the sending node sets the *RID* field to 0 and X_i to a random value. Each forwarding node n computes $RID(X_i) = [(RID(X_i) * X_i + IP_n) \bmod(p)]$ and updates the *RID* field in the packet with the new value. This should be performed in chunks of 8 bits and using a p value of 257 (smaller prime larger than 2^8). The *RID* will be constructed iteratively in the forwarding nodes and the AAAC server will be then able to reconstruct the path. One restriction of this approach is that the AAAC server needs a number of packets larger or equal to the number of encoded IPv6 addresses, to be able to reconstruct the path and identify the forwarding nodes. If the number of packets traversing the same route is smaller than the number of hops, the forwarding nodes will not be rewarded for these packets (the charging process is not affected, and the sender/receiving nodes will be correctly charged). This restriction can be minimised by caching recently reconstructed routes at the AAAC server, which can be evaluated following a try and error approach. This is not considered to be a limitation of the proposal. Nodes will still be interested in cooperating by forwarding packets if they know that they will be rewarded by the majority (even if not all) of the forwarded traffic. Moreover, the simulation results that will be depicted in this thesis confirm that the percentage of times it is not possible to identify the forwarding nodes is very small, and thus nodes will still be interested in forwarding packets.

Flow Authorisation

In the registration phase, the AR includes message the objects that define the requirements for flow admission in the *Session Initiation*. Depending on the requirements imposed by the operator, traffic inside the ad-hoc network can be admitted without restrictions. Other possibility is to only perform flow admission control on internal traffic. Traffic with one of the endpoints located on an external network is expected to be admitted at the AR, requiring no additional authorisation control. Other possibility is to perform flow admission in all new flows independently of the location of flow endpoints. Although this solution is the one with most overhead and less adaptable to mobility, it ensures no non-authorized packet is ever forwarded without explicit permission.

To authorise a flow, a forwarding node sends a *Flow Authorisation Request* message to the AR specifying the source and destination endpoints (*SrcIP* and *DstIP*), Protocol (*ULProto*), source and destination ports (*SrcPort* and *DstPort*), IPv6 traffic class (*TrafCl*), and a timestamp. Also, if not available, the public key of the sending node can be requested using the *Key* flag. As in all control packets, a MAC is also added (4.2), by concatenating:

$$SrcIP||DstIP||ULProto||SrcPort||DstPort||TrafCl||Timestamp||MAC \quad (4.2)$$

The AR issues a *FlowAuthorisationResponse* message allowing or denying the flow to be forwarded. The answer sent by the AR will take into account the user profile of the endpoints, credit information reported by the AAAC server or management policies defined by the network operator. Note that, potentially, this phase could be waived in many scenarios. The response is composed of a token stating the validity of the response. After this token expires, forwarding nodes must discard the token. If the flow is active and was accepted, token should be renewed some time before it expires. This is necessary to ensure the session is not temporarily suspended while forwarding nodes renew flow tokens. If the forwarding node also requested the public key of the sending node, besides the token, the key is also provided and should be used to authenticate the packet queued to be forwarded.

Reporting Phase

Like in SCP, the last forwarding node of each packet flow is responsible for gathering the proofs present in each packet. All information in the proofs is signed in order to provide protection against changes. Therefore, the node collecting the proofs cannot manipulate the reports in an attempt to obtain some additional profit. The reports are sent periodically, or as soon as it is reached a number of stored proofs enough to fill a packet with the size of the (MTU). If the end-point of a flow is located outside the ad-hoc network, the AR collects the proofs directly and reports them to the local AAAC server. Notice that the control packets will be forwarded without additional verification by the intermediate nodes, allowing a multiple hop return channel for the collected proofs. Since all proofs are cryptographically protected, nodes will not be able to alter the proofs without compromising the rewarding process.

The proofs are reliably sent to the infrastructure network; if an acknowledgement of the proofs reception is not received, a back off timer is activated and the proofs are sent again.

Verification Phase

After receiving the proofs, the AR sends them to the AAAC server, which will verify its authenticity. If the proofs are authentic, the relevant nodes are charged and rewarded; otherwise, the proofs are discarded.

To verify the sending node (charging validation), the AAAC calculates the digest from the information reported as performed by the sending node, and compares the 64 most significant bits of the result with the 64 most significant bits from the reported *HashChain*. If the values differ, the proof is considered to be invalid and it should be immediately discarded. If the values match, the proof is stored.

When the number of reported proofs is, at least, equal to the reported route length, the identification of the forwarding nodes can be performed. In this identification process, the AAAC solves the Vandermonde matrix using the values of the RID and X_i fields and considering the number of hops specified in the route length. The obtained result will be a list of IPv6 addresses that will be repeated every $HopC$ terms. The last forwarding node needs to match the IPv6 of the node that reported the proofs. If the RID fields are corrupted, the solutions of the matrix will not be unique. For each

proof (rewarding validation), the AAAC will compute the *HashChain*, in the same way performed by the forwarding nodes, and compare the less significant values of each proof with the values calculated. If all these procedures complete successfully, the proofs are considered to be valid and the nodes are rewarded; otherwise, the proofs are discarded.

If the proofs received refer to a route for which there is not enough information to identify the forwarding nodes, the charging procedure is executed and the proof is stored for a period of time. If the timer expires the proof is discarded and no rewarding occurs. If another proof for the same route arrives at the AAAC server, they are both cached until the number of proofs stored matches the number of hops reported. When this happens, the rewarding procedure occurs, and the new route is cached. The caching of the routes at the AAAC server will increase the efficiency of the rewarding algorithm.

4.2 SACP - Session Aware ad-hoc Charging Protocol

The Session Aware Ad-hoc Charging Protocol is a charging and rewarding protocol for ad-hoc networks in scenarios with interconnection to the infrastructure network managed by a network provider. Just like the previous proposal (PACP), SACP is based on rewarding concepts of SCP, but improves over existing proposals by adding functionalities capable of providing distributed access control while minimising control overhead. It shares many design solutions with PACP, yet, some mechanisms operate differently. Also, its usage scenarios are similar to the ones presented previously for interconnected ad-hoc networks (see Figure 4.1).

Like PACP, the general SACP operation is depicted also by figure 4.2. When a node enters in the ad-hoc network, it goes through a registration process to become known by the AR (and AAAC), to be authenticated and authorised, and to receive the cryptographic material required for the charging process (Registration phase). When communicating with other nodes inside or outside the ad-hoc network, each forwarding node processes the packet and forwards it; the fields containing information on the route are cryptographically secured, so they cannot be wrongly modified along the path (Participation and Forwarding phases). However, SACP includes concepts of Automatic Route Shortening (ARS) [johnson04], with routes cached at the nodes for a specified amount of time. Therefore, the packets only carry information about the route for the charging process when the route information in the nodes is about to expire or topology changes are detected. The last forwarding node, is again responsible

for sending the proofs to the AR (Reporting phase).

These proofs contain again information on the path(s) of the flow, and are sent to the AR when the number of proofs collected in the node reaches a specific number, or when a timeout expires. Notice that, in the case of traffic with destination outside the ad-hoc network, the last forwarding node is replaced by the AR. When receiving the proofs, the AR sends them to the AAAC to verify the truthfulness of the information, through the cryptographic information contained in the proofs, and retrieves the information of the ad-hoc route (Verification phase). The AAAC is then able to correctly charge the sending (and, eventually, the receiving) nodes, as well as correctly reward the forwarding nodes. The Registration and Verification phases are similar to the same phases in PACP, as well as the security assumptions and specificities. In the next sub-sections it will be detailed the remaining phases of the mechanism which are specific to SACP.

Participation

When a node generates a packet, it checks if there is already information of this flow and its current destination. This information is denoted by a session. The information about this session is stored in the node. If this session does not exist, it creates a new one. Then, it activates a timer, the *RouteUpdateTimer* that, upon expiration, triggers the event of adding the route information, in the outgoing packets, at specified intervals. This route information is included in *RouteUpdate (RU)* messages. In order to save bandwidth, *RU* messages are sent inband in data packets. The node also sets the number of packets that may be sent, before a new *RU* message is sent. Such value is named *PacketsUntilRU* and is initialised to a user defined value. It also generates a new *Flow ID* for this session, different from 0 and from recently used values. The first data packet sent will then carry a *RU* message and will have the generated *Flow ID* set in the IPv6 header. The fields of the *RU* message are described by 4.3, concatenating:

$$Code||NHops||RouteList||SeqN||HashChain||MAC \quad (4.3)$$

Code field is used to notify nodes that the header was added by the AR and not the sending node. This bit is used to notify whether a packet is coming from the outside the ad-hoc. It is comprised of a 8bit value where only the least significant bit is currently defined. Others are reserved for future enhancements to the protocol. *NHops*

indicates the number of addresses present in the list of nodes in the path, the *RouteList*; it starts with the value 0 at the sender and is incremented by one at each hop towards the destination. The *RouteList* will be non-existent in the packet just sent, and will grow by one IPv6 address at each forwarding node. The sequence number *SeqN* is a random number used to avoid repetition attacks. The *MAC* (Message Authentication Code) field contains the signature of the data packet, created with the sending node private key. The *HashChain* will be initially created by hashing a pseudo header with data from the packet and the node itself. If the originator is located outside the ad-hoc network, *Code* field less significant bit should be set to 1 and the public key of the AR should be used to verify the *MAC*. This *MAC* is used both to assure the identity of the sending node and the integrity of data.

When sending a packet, if the *RouteUpdateTimer* expired or if *PacketsUntilRU* reaches 0, the *RouteUpdateTimer* is rescheduled, *PacketsUntilRU* is reset to the pre-configured value, a new *Flow ID* is generated, and a *RouteUpdate* message is added. The combination of a variable counter with a timer allows the refresh of the route after a certain timeout or after a certain number of packets, thus minimising errors in the rewarding mechanism.

Further packets will not carry a *RouteUpdate* header, but instead will carry the new *Flow ID* and a *PacketProof* (*PP*) header with the structure defined in 4.4:

$$Code||SeqN||HashChain||MAC \tag{4.4}$$

The *HashChain* field is constructed through the same pseudo header as before (as in PACP). This much smaller proof requires less processing from the forwarding nodes and causes less network overhead. However, it may not contain information about the “real” list of forwarding nodes: the ones rewarded will be the ones that forwarded the packet containing the *RouteUpdate* header with the correspondent *Flow ID*. Because of this issue, nodes will only be willing to forward packets with known *Flow ID* values.

Forwarding

Upon receiving a packet, each forwarding node checks if the *MAC* field is correct. If it is not, the packet is dropped. Depending on the origin of the packet, it should be using either senders’ public key, or ARs’ public key. This operation should be

performed independently of the type of inband message included. It could be ignored if no signature verification failures were detected in the recent past and the operator allows it.

If the packet has a *RouteUpdate* message, the node increases the *NHops* variable and adds its IPv6 address to the *RouteList*. In the case of a TCP packet, this operation will require adjusting TCP fields like the checksum and maximum segment size (*TCP MSS*). However, this is only required to be performed at each node when *RouteUpdate* messages are present. Then, the node concatenates the charging pseudo header constructed from the packet with the *HashChain* and calculates a new digest. The result of the new hash will be inserted in the *HashChain* field. After this process, since the forwarding node needs to remember that it is included in the route indicated by the *Flow ID*, it caches the pseudo header until a configured timeout expires or until a packet of the same session arrives with a new *Flow ID*.

If the packet carries a *PacketProof* header with a known *Flow ID*, the forwarding node just updates the *HashChain* and forwards the packet. On the other hand, if the packet only carries a *PacketProof* header but the *Flow ID* is unknown, the forwarding node decides (based on user configuration) if it should forward the packet for free. The decision will depend on users' interest in forwarding packets without being rewarded, and in the number of packets already forwarded for free. It is assumed nodes are willing to forward a very small percentage of the traffic for free, if that results in better connectivity in the network. Because packets may be forwarded by a node different from the one to be rewarded, there is an inherent error.

A situation without errors happens when all packets have the full route encoded or the network is static; the error ratio grows both with higher mobility of nodes, higher periods between *RouteUpdate* and higher values of *PacketsUntilRU*. However, the purpose of the mechanism is to minimise these situations. In either situations (packet being forwarded or not), nodes should notify the sending node (or the AR) that the route changed. This is performed using a *RouteUpdateRequest* message. This message contains information about the session source and destination, and *Flow ID*. By requesting a new *RouteUpdateMessage*, the forwarding node will insure it will be included in the charging process and will be rewarded in a near future. The effects of mobility should not create too much control overhead in the network; thus the rate of sending *RouteUpdateRequest* messages should be limited (2 per second per flow). If the sending node receives these messages at a higher rate than expected from a single node,

they should be silently dropped. Upon successful reception of a *RouteUpdateRequest* message, the node will determine if that session is still valid and, case it is true, verify if the *Flow ID* reported is the flow active for that session. If all is true, next data packet will carry a *RouteUpdateMessage* instead of a *PacketProof*.

Reporting

The last forwarding node is responsible for collecting the proofs present in the packets, either *RouteUpdate* or *PacketProof* messages. Proofs are packed in a *ProofReport* message and are sent to the AR periodically, or as soon as the number of proofs cached is enough to fill a packet with the Maximum Transmit Unit (*MTU*). The AR verifies all information and the various *HashChains* contained in the *ProofReport*, and issues a *ProofResponse* to acknowledge the reception of the proofs. Both the *ProofReport* and *ProofResponse* messages are protected by public key cryptography. Before trying to decode either message, first the *MAC* field must be verified.

4.3 Overhead analysis

Both the proposed solutions focus on reducing the network overhead of SCP. PACP also focus in avoiding the necessity of fragmentation as a packet is routed. All three proposals are very similar from a behaviour point of view. Most of the charging overhead will be due to packet marking and proof reporting which is performed by all the proposals.

Registration and flow admission, present in PACP and SACP, increases the overall overhead, but its contribution is meaningless compared with other factors. SCP lacks these mechanisms. Moreover, flow admission is an additional feature and not a required functionality. Cryptography also poses a major contribution to the overhead created by each solution. Because the same set of primitives is used in all three proposals, the relative overhead of each solution will not vary, instead the absolute overhead will.

In the following section an analysis of the overhead produced by each proposal will be described. All values obtained are calculated in relation to the amount of data sent to the network. Overhead is expressed as the percentage of additional control bytes added by each analysed charging protocol; routing and other mechanisms are ignored. In all proposals, the total overhead has two individual contributors which are referred as:

marking overhead and reporting overhead. Marking overhead is the amount of control data added to data packets (inband). Reporting overhead is the overhead produced by reporting proofs back to the AR. The total overhead will thus be the sum of both.

4.3.1 SCP

SCP adds a hop-by-hop extension to all packets sent. The size of the header, and its evolution as a packet is routed, will depend on the routing protocol used, and the mode SCP is operating.

Headers can be divided in three parts: SCP control information, hop list (route), and *MAC*. The first part has a fixed size of 27 bytes and is used by SCP in all modes. The Hop list will either contain the full path or a partial path and each entry takes 16 bytes on the header. MAC size will much depend on the size of the key used by nodes and its value will be the same as in other proposals. The marking process overheads' will vary with the number of hops, due to the variation on the size of the hop list. Equation 4.5 denotes the overhead using Full encoding while Equation 4.6 refers to Interactive encoding; A full list of the symbols used in the following equations is present in table 4.1.

Symbol	Description
$DPkt_s$	Size of data packet
$DPkt_i$	The i th data packet
$FullProof_s$	Size of a PACP Proof when end-points are not neighbours
$IPv6_s$	Size of IPv6 header
MAC_s	Size of Message Authentication Code
MTU	Maximum Transmit Unit
$nhops_r$	Number of hops in the route
$nhops_{LFNtoRN}$	Number of hops between last forwarding node and receiving node
$nhops_{LFNtoAR}$	Number of hops between last forwarding node and the access router
pad_s	Size of additional padding
$Proof_s$	Size of an individual proof
$ProofReport_s$	Size of a <i>ProofReport</i> message
$ProofReply_s$	Size of a <i>ProofReply</i> message
$RouteReq_s$	Size of a <i>RouteRequest</i> message
$SmallProof_s$	Size of a PACP Proof when end-points are neighbours
$SPCReq_s$	Size of a <i>ServiceProvisionConfirmationRequest</i> message
$SPCResp_s$	Size of a <i>ServiceProvisionConfirmationResponse</i> message
$SPVReq_s$	Size of a <i>ServiceProvisionVerificationRequest</i> message
$SPVResp_s$	Size of a <i>ServiceProvisionVerificationResponse</i> message
UDP_s	Size of a UDP header

Table 4.1: Symbols used on the equations of current section

$$SCP_{mFull} = \frac{27 + nhops_r * 16 + MAC_s + pad_s}{DPkt_s} \quad (4.5)$$

$$SCP_{mIter} = \frac{(27 + MAC_s + pad_s) * (nhops_r + 1) + \sum_{n=0}^{nhops_r} n * 16}{DPkt_s * (nhops_r + 1)} \quad (4.6)$$

Equations 4.5 and 4.6 are only true if fragmentation does not occur. Considering the initiator is sending a packet size of MTU and the number of hops ($nhops_r$) is higher than 0, more overhead is added. This overhead corresponds to additional IPv6 and Fragmentation headers. If the packet is only fragmented once (a reasonable assumption for ad-hoc networks), the overhead can be further stated as in Equation 4.7.

$$SCP_{mIterFrag} = \frac{25 + MAC_s + pad_s + \sum_{n=0}^{nhops_r} (n * 16 + IPv6_s + IPv6Frag_s)}{DPkt_s * (nhops_r + 1)} \quad (4.7)$$

In the SCP report process, a pair of $SPVReq$ and $SPVResp$ messages are exchanged between the last forwarding node and the destination node. Later $SPCReq$ and $SPCResp$ messages are exchanged between the last forwarding node and the AR. The process can thus be divided in two phases. The overhead of the reporting process will be the sum of both phases.

In the first phase the last forwarding node sends a $SPCReq$ to which the receiving node replies with a $SPCResp$ message. The size of both messages is easy to calculate because it only varies with the size of the MAC . Typically the last forwarding node is a neighbour of the receiving node, but this is not always true. If the flow terminated before the process started and topology changed, nodes could be separated (represented by $nhops_{LFNtoRN}$ and $nhops_{RNtoLFN}$). The overhead regarding the first phase of the report process is represented by equation 4.8.

$$SCP_{rep1} = \frac{SPCReq_s * (nhops_{LFNtoRN} + 1) + SPCResp_s * (nhops_{RNtoLFN} + 1)}{nproofs_{scp} \sum_{i=1} (DPkt_i * (nhops_r + 1))} \quad (4.8)$$

The number of proofs ($nproofs_{scp}$) which can be reported on a single packet, depend on the MTU and the size of the MAC . Equation 4.9 determines the value of this parameter. The values present in the equation represent field of fixed size.

$$nproofs_{SCP} = \frac{MTU - IPv6_s - UDP_s - 48 - 16 * nhops_r - MAC_s}{21} \quad (4.9)$$

Considering the default MTU in 802.11, no security ($MAC_s = 0$) and $nhops_r$ is lower than 8, $nproofs_{SCP}$ will be 60. The impact of using security will depend on the size of the key and algorithm. An ECDSA key of 160 bits produces a signature with 40bytes. The maximum number of proofs reported will thus be reduced by 2. Figure 4.4 depicts how the ECDSA key size limits the maximum number of proofs reported.

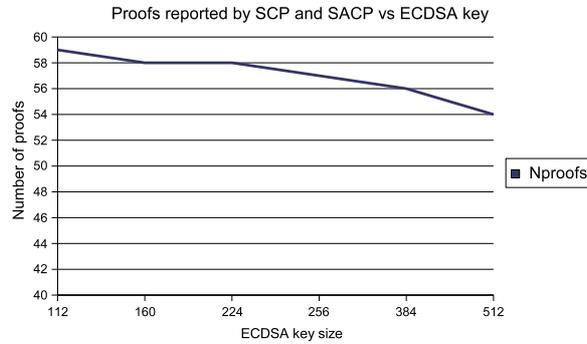


Figure 4.4: Variation of the number of proofs reported by SCP in relation to the size of the ECDSA key; assuming a MTU of 1500, MAC_s of 0 and $nhops_r$ lower than 8.

The second phase of the reporting process occurs when the last forwarding node sends a $SPVReq$ message to the Access Router. The result will be later transmitted with a $SPVResp$ message. The first will contain session information and proofs acknowledged in the first phase. Its size will be approximate to the MTU but, because the granularity imposed by the size of proofs and session, it could be lower. Each $SPVResp$ message

¹ UDP_s : Size of an UDP header, MTU : Maximum Transmit Unit (1500 bytes in 802.11)

has 54 bytes, plus *MAC*, (considering IPv6 and UDP). The overhead related to the second phase of the reporting process is stated in equation 4.10.

$$SCP_{rep2} = \frac{SPVReq_s * (nhops_{LFNtoAR} + 1) + SPVResp_s * (nhops_{LFNtoAR} + 1)}{\sum_{i=i}^{nproofs_{SCP}} DPkt_i * (nhops_r + 1)} \quad (4.10)$$

The total control overhead of SCP will be the sum of both the marking and the reporting phases. Because the overhead during the marking phase can be produced by two different methods of encoding, there are two equations (4.11 and 4.12) representing the total control overhead produced by SCP.

$$SCP_{totalIter} = SCP_{markIter} + SCP_{rep1} + SCP_{rep2} \quad (4.11)$$

$$SCP_{totalFull} = SCP_{markFull} + SCP_{rep1} + SCP_{rep2} \quad (4.12)$$

4.3.2 PACP

In the case of PACP, marking overhead is constant along the route and almost independent of the packet size or contents. It is not absolutely constant because size of ECDSA signature can vary with different security levels. The fields composing a charging header take 46 bytes on each packet, plus *MAC* and padding (equation 4.13).

$$PACP_{markFull} = \frac{46 + MAC_s + pad_s}{DPkt_s} \quad (4.13)$$

Also, PACP tries to reduce the overhead by eliminating some fields from the header in the case of a communication between neighbour nodes. Instead of the typical 46 bytes per packet (plus *MAC*) each header only takes 26 bytes (also plus *MAC* and padding). The fields removed are only those required for identifying forwarding nodes. All other functionalities are maintained. The overhead of control data in case of neighbouring nodes is stated in equation 4.14.

$$PACP_{markNeigh} = \frac{26 + MAC_s + pad_s}{DPkt_s} \quad (4.14)$$

Unlike SCP, PACP reporting phase only considers one phase where the last forwarding node reports the collected proofs to the Charging Manager. The report message is then acknowledged thus completing the report phase. Each report comprises of a set of session information (42 bytes), a list of routes and a list of proofs inside each route. Each proof is composed by the *SeqN*, the X_i , a *RID*, the *TrafficClass*, and the result of the *HashChain*. The PACP proofs are much larger than SCP or SACP proofs. This will cause the number of proofs reported per *Report* packet to be smaller. The number of proofs sent in each report, considering multiple routes per report and the usage of *MAC*, is given by equation 4.15 and some results depicted in figure 4.5. The number of routes in each packet will depend on the available space and proofs in each route. That is, when a route needs to be reported, all proofs of that route will be included while there is space in the packet. If, after all proofs are copied, there are still some free bytes, another route is included. The process repeats until adding another proof would exceed the *MTU*. If the session end-points are neighbours, each proof occupies 21 bytes ($Proof_s$). If this is not the case, each proofs takes 38 bytes.

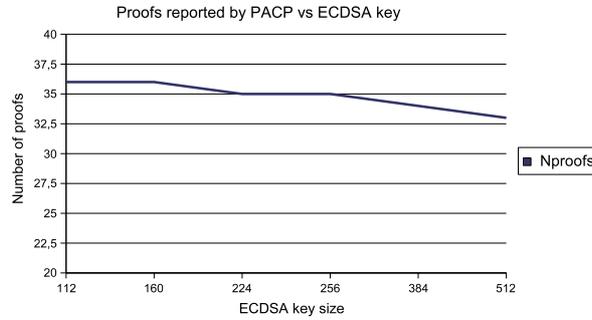


Figure 4.5: Variation of the number of proofs reported by PACP in relation to the size of the ECDSA key; assuming a *MTU* of 1500, MAC_s of 0 and $nhops_r$ lower than 8.

$$nproof_{sPACPF_{ull}}(Proof_s) = \frac{MTU - (IPv6_s + UDP_s + 42 + MAC_s + nroutes * 4)}{Proof_s} \quad (4.15)$$

The overhead resulting from PACPs' reporting phase will then be expressed by the ratio presented in equation 4.16

$$PACP_{report} = \frac{ProofReport_s * (nhops_{LFNtoAR} + 1) + ProofReply_s * (nhops_{ARtoLFN} + 1)}{nproofs_{PACP} \sum_{i=i} DPkt_i * (nhops_r + 1)} \quad (4.16)$$

PACPs' total overhead will be the sum of the overhead at each phase (equations 4.17 and 4.18). The value will depend on the distance between sender and receiver as the size of the charging proof will vary.

$$PACP = PACP_{markNeigh} + PACP_{report}(SmallProof_s) \quad (4.17)$$

$$PACP = PACP_{markFull} + PACP_{report}(FullProof_s) \quad (4.18)$$

4.3.3 SACP

SACP differs from SCP because it tries to optimise the marking overhead by reducing the need for iterative or full encoding. The route is only sent when there is a change in the topology or after a determined timeout or number of packets. Overhead will much depend on the mobility of the network and packet rate. Networks with low mobility will be much benefited from using SACP. Networks with high mobility will show to induce high rates of overhead. The definition of low or high mobility, to SACP, represent networks where route changes faster than the *RouteUpdate* period (high mobility) or are more stable (low mobility). The typical value for the *RouteUpdatePeriod* is 5 seconds. The marking overhead for the static case is represented by equation 4.19. The first ratio corresponds to the *PacketProof* message while the second to the *RouteUpdate* message.

$$SACP_{markStatic} = \frac{26 + MAC_s + pad_s}{DPkt_s} + \frac{\sum_{n=0}^{nhops_r} (1 + 16 * n + pad_s)}{PacketRate * 5 * Dpkt_s * (nhops_r + 1)} \quad (4.19)$$

In the dynamic case, it is required to consider that most data packets will contain a *RouteUpdate* message and nodes send *RouteUpdateRequest* (*RUR*) messages. There

are two probability values which must be considered. First is the probability of a node of sending a *RouteUpdateRequest* message (*RURprob*). Second is the probability of a data packet to contain a *RouteUpdate* message (*RUprob*). In the worst possible case, both probabilities will be 1, meaning all data packets will contain a *RouteUpdate* message and route changes on a packet basis. In the static case, the period between automatic *RouteUpdate* messages can also be expressed as a probability (equation 4.20) while the probability of *RUR* messages is 0. If no packets are lost, the second probability will be the sum of all nodes *RURprob* plus the probability of the *RouteUpdatePeriod* expiring. The overhead of SACP in a dynamic scenario is stated in equation 4.21.

$$RUprob_{static} = \frac{1}{PacketRate * RouteUpdatePeriod} \quad (4.20)$$

$$SACP_{markDyn} = \frac{(26 + MAC_s + pad_s) * (nhops_r + 1) + RUprob * \sum_{n=0}^{nhops_r} (1 + 16 * n)}{DPkt_s * (nhops_r + 1)} + \frac{\sum_{n=0}^{nhops_r} (RURprob * RouteReq_s * n)}{Dpkt_s * (nhops_r + 1)} \quad (4.21)$$

As in the case of PACP, SACP sends proofs to the operator without any verification of the receiving node. The number of proofs is also variable and depends both on the *MTU* and the size of the *MAC*. This value is the same as in SCP and can be calculated as stated in equation 4.9. The control overhead of SACP due to report of proofs is stated as equation 4.22.

$$SACP_{rep} = \frac{ProofReport_s * (nhops_{LFNtoAR} + 1) + ProofReply_s * (nhops_{ARtoLFN} + 1)}{\sum_{i=i}^{nproofs_{SACP}} DPkt_i * (nhops_r + 1)} \quad (4.22)$$

After all phases are correctly represented, it is now simple to state the total overhead expected from SACP. Equation 4.23 states the overhead in static networks, while equation 4.24 can be applied to the dynamic case.

$$SACP_{static} = SACP_{markStatic} + SACP_{report} \quad (4.23)$$

$$SACP_{dyn} = SACP_{markDynamic} + SACP_{report} \quad (4.24)$$

4.3.4 Overhead comparison

With the simplified equations identifying the overhead ratio of each protocol, it is possible to perform a more detailed analysis of each proposal. If the same scenarios are considered, it is also possible to compare the expected overhead. Because it was assumed the network to be reliable (no collisions or packet drops) the results obtained should be considered as the best case. Using real world equipment and the 802.11 shared medium, overhead can (and will) differ. It is however expected this analysis is still helpful in providing a rough characterisation of each proposal. Because it is difficult to characterise all possible scenarios, only 3 are evaluated in this thesis. The first scenario represents a hotspot where all traffic departs from the access point and all nodes are directly connected to the AP. Node mobility is considered to be restricted to link coverage and no multi-hop forwarding is considered. Although this scenario does not represent a MANET in its true sense, it is the most frequent scenario involving hotspots using wireless technologies.

The second scenario represents a static network where it is possible to have multi-hop communications. Nodes are not able to move and routes are static. This scenario represents a static, multi-hop network such as the internet or an hotspot with no mobility.

The last scenario represents a multi-hop capable network where routes have some probability of change. In order to facilitate calculations, the number of hops is considered to be constant. Figure 4.6, figure 4.7 and figure 4.8 represent the 3 scenarios considered.

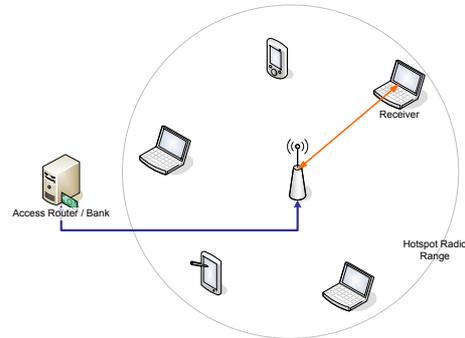


Figure 4.6: Scenario 1 - A typical hotspot with no mobility or multi-hop forwarding

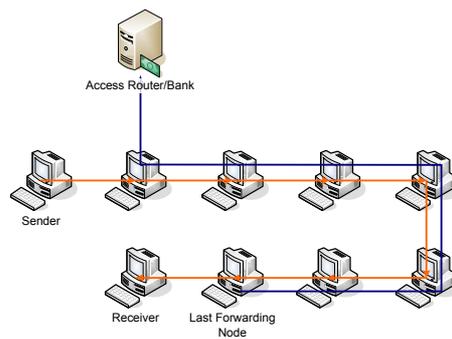


Figure 4.7: Scenario 2 - An ad-hoc extended hotspot with no mobility but with multi-hop capabilities

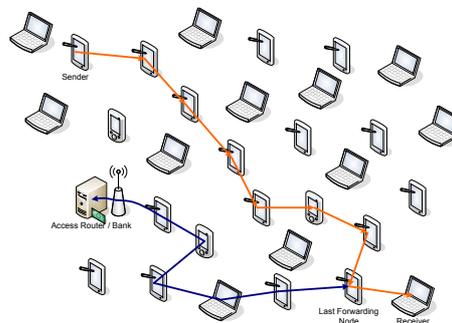


Figure 4.8: Scenario 3 - An ad-hoc extended hotspot with mobility and multi-hop capabilities

The values considered for the value of the different variables are expressed in table 4.2. The value of the padding depends on each protocol, scenario and message and is calculated depending on the other fields. Except where noted, all size values are in bytes, rates are in kbits and time is represented in seconds.

Variable	Value	Description
MTU	1500	Maximum Transmit Unit
MAC_s	40	Size of the MAC (ECDSA 163bits)
$IPv6_s$	40	Size of the IPv6 header
UDP_s	8	Size of the UDP header
$DPkt_s$	512	Data Packet Size (Payload)
$PacketRate$	20	Packet Rate (pkts/s)
SCP $SPVReq$	1479	Size of a SCP $SPVReq$ message
SCP $SPVResp$	54	Size of a SCP $SPVResp$ message
SCP $SPCReq$	74	Size of a SCP $SPCReq$ message
SCP $SPCResp$	58	Size of a SCP $SPCResp$ message
PACP $ProofReply$	58	Size of a PACP $ProofReply$ message
SACP $RURReq$	68	Size of a SACP $RURReq$ message
SACP RUP	5	Delay between automatic RUR messages ($RouteUpdatePeriod$)
Scen 1- $nhops_r$	0	Hops between endpoints
Scen 1- $nhops_{FNtoAR}$ and $nhops_{ARtoFN}$	0	Hops between LFN and AR (and vice versa)
Scen 1- PACP $ProofReport$	1495	Size of a PACP $ProofReport$ message
Scen 2,3- PACP $ProofReport$	1464	Size of a PACP $ProofReport$ message
Scen 2,3- $nhops_r$	7	Hops between flow end points
Scen 2,3- $nhops_{FNtoAR}$ and $nhops_{ARtoFN}$	5	Number of hops between LFN and AR (and vice versa)
Scen 3- $RURprob$	0.10	Probability of a route change implying a RUR message.

Table 4.2: Values used to calculate overhead in the different scenarios

Values obtained for scenario 1 are depicted in table 4.3. In this scenario only SCP should present overhead in the reporting process due to the confirmation of proofs. Other protocols have no overhead (inside the ad-hoc network) related to reporting as they do not acknowledge reception of proofs. Both PACP and SACP present the lower values for Marking and Total overhead. These should be the best solutions to provide charging on the depicted scenario. The difference between resulting overhead values is 0.59% which can be considered as irrelevant. If a real network was to be deployed, other factors such as processing requirements or easy of deployment should be considered before these values of overhead.

Table 4.4 presents the values obtained for scenario 2. Comparing with the first scenario, control overhead is higher in the second case. This is due to the increase in

Protocol	Marking	Reporting	Total
SCP Full	12.86%	0.59%	13.45%
SCP Iterative	12.86%	0.59%	13.45%
PACP	12.86%	0.00%	12.86%
SACP	12.86%	0.00%	12.86%

Table 4.3: Expected overhead on Scenario 1

the size of the route and the necessity for report proofs (Last Forwarding Node is not collocated with Access Router). In contrast with the first scenario, where the different methods of route encoding supported by SCP produce the same result, in the second scenario there is some difference. The Iterative encoding is able to reduce in 10% the overhead produced by SCP with Full route encoding. SCP with full route encoding is the proposal producing more overhead (36.77%) with a difference of more than 6% from the most efficient proposal. PACP presents less marking overhead than SCP due to its polynomial encoding. In contrast, the reporting overhead is higher because of its higher proof size resulting. The result will be fewer proofs will be sent per report packet. SACP is the most efficient proposal both at marking and reporting, achieving an overhead as low as 16.81%. SACP report overhead is similar to the values obtained for SCP. The difference exists because SACP performs no confirmation of the proofs.

Protocol	Marking	Reporting	Total
SCP Full	32.86%	3.91%	36.77%
SCP Iterative	22.86%	3.91%	26.77%
PACP	16.43%	6.97%	23.40%
SACP	12.96%	3.84%	16.81%

Table 4.4: Expected overhead on Scenario 2

It is noticeable that the increase in the number of forwarding nodes, also increases the marking overhead of some proposals. Figure 4.9 depicts how the increase influences the number of control bytes transmitted in-band on data packets. SCP (both Iterative and Full encoding) grows linearly, making this proposal less suited for networks with high number of hops. Using SCP on the Internet where the average number of hops is around 23 [frei98], SCP with Full encoding adds more than 350 bytes (plus *MAC*) to each data packet. In ad-hoc networks it is safe to consider the average number of hops to be much lower, never reaching such values. PACP shows the number of control bytes per data packet to be almost constant. The only variation existing is between neighbour and non neighbour flows. SACP is the proposal adding the fewer control bytes to each packet independently of the number of hops. It grows with the number of hops, but it does so very slowly never getting higher than PACP for reasonable values.

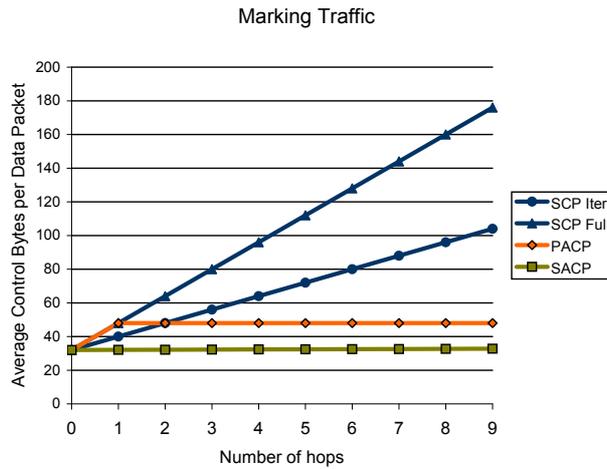


Figure 4.9: Variation of marking overhead with the increase on number of hops

Scenario 3 is similar to scenario 2 with some differences; nodes are able to move, and routes will change with a probability of 10% when a packet is to be forwarded or sent. In a real environment, such value for mobility can be considered as high. As presented in table 4.5, only SACP seems to be affected by the mobility. Its values are almost the double compared with scenario 2. Such variation is explained by the necessity of sending *RouteUpdateRequest* messages informing about the route change. Also, *RouteUpdate* messages, which will contain the full route, are added to data packets more often. The result of the two effects will be an higher marking overhead for SACP. As PACP is immune to the increase in route length, is now the most efficient proposal.

Protocol	Marking	Reporting	Total
SCP Full	32.86%	3.91%	36.77%
SCP Iterative	22.86%	3.91%	26.77%
PACP	16.43%	6.97%	23.40%
SACP	24.67%	3.84%	28.51%

Table 4.5: Expected overhead on Scenario 3

If the probability of the route changing increases, SCP and PACP will remain immune to this change. However, SACP will start generating more overhead associated to the marking process. Figure 4.10 depicts the expected variation in marking overhead according to several values for route change probability. As mobility increases SACP surpasses PACP when route change probability is around 8%, SCP with Iterative encoding around 20% and SCP with Full encoding when it is near 40%. As mobility further increases, so will SACP marking overhead, limiting SACP usability to scenarios with low mobility. Such scenarios would probably also be unusable to applications

unless they were specially designed for these scenarios. With a 30% probability of a route changing when a packet is sent or forwarded, even for low number of hops, routes would be changing on a packet basis. This would result in high routing overhead, high delay and high percentage of packet drops.

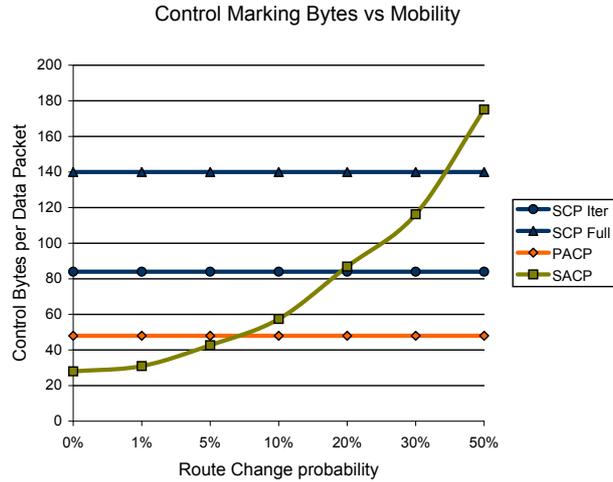


Figure 4.10: Variation of marking overhead with the increase of mobility. SACP line has fluctuations due to padding.

From the results obtained in this analysis, it is clear SACP is the protocol with better efficiency in slow (or even moderately) moving or static networks. In all scenarios it was able to always present lower values of overhead. If nodes are able to move, SCP usage should be carefully evaluated against the expected route change probability. As the probability increases, SACP will be even more inappropriate and PACP or SCP will be a better choice. Considering route length, the only proposal not affected by this issue is PACP.

4.4 Obtained results

In order to further validate the solutions proposed, a set of simulations were performed. This simulation work was done using NS-2 [ns2] version 2.27, compiled using gcc 3.3.6 [gcc] on Linux hosts.

4.4.1 Simulation Modules

Besides implementing the protocols being evaluated (SCP, SACP and PACP), a few changes had to be made to the simulation environment. The changes envisioned the possibility of intercepting data packets either received, sent or being forwarded. The developed module (*NSTAP*) is installed in the node entry-point (see figure 4.11) acting as a generic filter intercepting packets and delivering them to any agent registered into it.

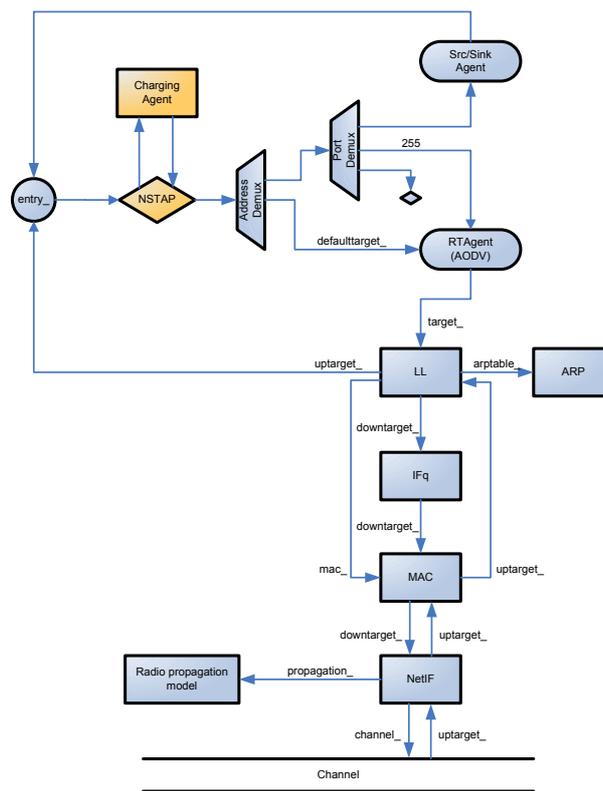


Figure 4.11: Diagram of a wireless node in NS-2 and the NSTAP module

Upon reception of the packet, the agent (charging protocol) has the opportunity to process it and issue a verdict. Four possible values for the verdict are possible:

ACCEPT, *DROP*, *FREE*, *QUEUE*. The first verdict (*ACCEPT*) orders the *NSTAP* module to let the packet proceed to other agents. Notice the payload of the packet could have been modified by the agent. The *DROP* verdict will drop the packet and write a message to the trace file. The *FREE* verdict works as the *DROP* but no message is added to the logs. This is useful to deal both with packets received and internal messages. Besides being accepted or dropped, some packets are required to be queued. The *QUEUE* verdict issues the *NSTAP* module to stop processing the packet. Later the agent is able to reschedule a concrete action to be performed on the packet (receive, forward, drop...).

Because the internal structure of a node running DSR is slightly different (see figure 4.12) from a node running AODV, another interception point had to be defined. This point was set inside DSR just after the route was found and the packet is to be sent or forwarded. Only DSR and AODV were tested using the *NSTAP*. Unless the routing protocol changes the internal structure of wireless nodes, this modules can probably be used with other protocols.

By using *NSTAP*, the interceptor agents (charging protocols) will receive more packets than standard agents. While routing messages (as well as MAC Layer packets) are filtered out, TCP and IP retransmissions are processed by *NSTAP* agents. While using charging agents, this will result in more packets being charged than the ones really received by application agents. In real world, methods for intercepting packets, such as *IP6Queue* (from Linux *Netfilter* stack), suffer from similar same issues. Because the charging protocols operate at a network level, (that is: protocols charge and reward packets in the network, opposed to messages to higher layer services) this situation is acceptable. Also, forwarding nodes should be rewarded by the packets they actually forward, even if they are retransmissions since they will spend resources also.

One important aspect when doing simulations is the Random Generator and the seed used to initialise it. NS2 uses the default seed of 12345. If the seed is not changed, running several times the same environment would result in similar results. NS-2 Random Number generator (MRG32k3a [ecuyer99]), as the majority of generators, provides the possibility of initialising the seed. One additional feature is the support for independent sub-streams. This can be seen as a bi-dimensional generator where the same seed, will also provide n independent sub-streams of random events. The method used to generate random values for the various variables was the following. When creating simulation environments, for each combination of mobility pattern, transport protocol

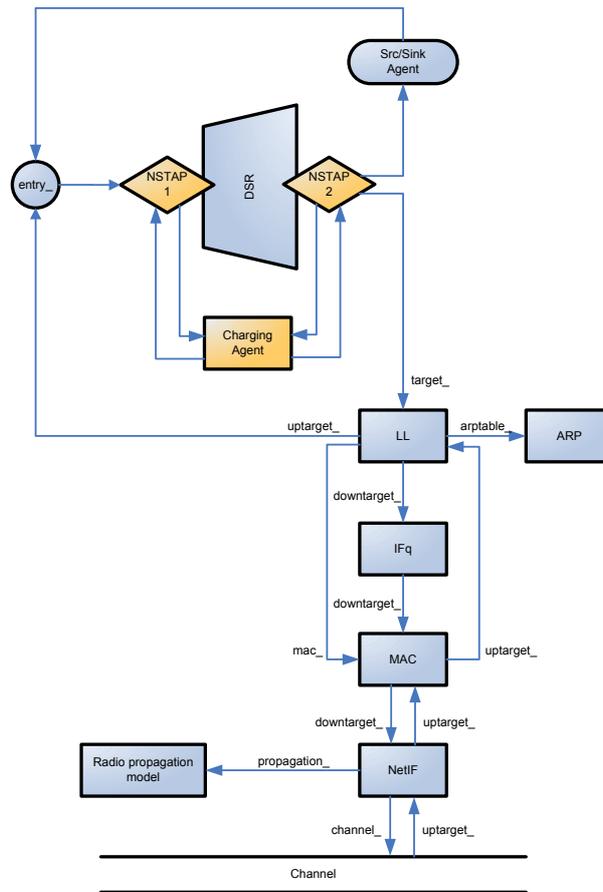


Figure 4.12: Diagram of a DSR wireless node in NS-2 and the NSTAP module

and number of flows a random seed is created. At each (n) of the 100 simulation runs of each scenario, the random number generator is seeded with this value and one generator is created per variable. Depending on the purpose of the variable, the parameters of the individual random number generators, such as value interval, average and distribution were configured. At last, the n sub-stream was selected for each variable. The result of such method is each variable of the same scenario use the same seed, while each run uses an independent sub-stream of values. The result would be similar to using the same stream and different seeds at each run. The described method was used because it was more adequate according to the way the simulations were structured. Just by analysing the seed, it was possible to rapidly identify if two simulations were from the same scenario.

4.4.2 Simulation Environment

Two set of tests were executed: one using AODV and other using DSR. In both sets the three proposals (SACP, PACP and SCP) are evaluated. Both DSR and AODV are evaluated because SCP and SACP perform differently when using DSR or other non source route based routing protocol, such as AODV.

In order to evaluate the behaviour of the proposals on real environments, it was created a scenario representing an ad-hoc extended wireless hotspot. In this environment with 1000x1000 meters, 40 nodes are deployed randomly and one is selected to be the hotspot gateway. Nodes are able to move following a Random Waypoint Model with pause time of 0s. Because variations on mobility affect the performance of the protocols, several simulations were created with different velocity intervals. In each instance the values for mobility, of each node, were uniformly distributed between 0 and either 1, 5, 10 or $30ms^{-1}$. When nodes reach the expected destination, a new destination is chosen together with the new node speed. An additional scenario was also simulated where nodes were randomly placed and static.

Traffic generated can be either direct between ad-hoc nodes or from the hotspot gateway. The number of flows from the outside (gateway) is the triple as the number of direct flows between nodes. Both TCP and UDP were tested. Flows simulated FTP (File Transfer Protocol) applications using TCP and the number of total flows generated on the network vary between 1, 2, 3, 4, 5, 10, 15, 20, 25, and 30. All TCP options were set at the default, except the ECN mechanism which was activated. In each experiment, the flows are initiated according to a Poisson process with a mean time interval between calls of $200/(number\ of\ flows)$, and each flow has an average duration exponentially distributed with average 30s. 3 classes of UDP flows were generated on the network: Constant Bit Rate (CBR) flows representing Audio and Video streams and on-off exponential (Exp) representing a Variable Bit Rate (VBR) stream. The characteristics of these UDP flows are summarised in table 4.6. The total number of UDP flows generated in each run also varied as with TCP. Individual flows were created until the total number of flows was reached. The generation of UDP flows took in consideration the probability values expressed in table 4.6.

Each simulation lasted 200s and each run was generated as follows. First a topology is generated, which includes placing nodes and creating mobility patterns. Then a routing protocol is chosen together with the transport protocol (UDP or TCP). For

Arrival (<i>s</i>)	Duration (<i>s</i>)	Packet Size (<i>bytes</i>)	Bit Rate (<i>kbs</i> ⁻¹)	ON/OFF time	Probability	Name
15	30	80	48	-	50%	Audio
30	60	1000	256	-	25%	Video
10	60	512	128	1/1	25%	VBR

Table 4.6: Parameters modelling the creation of UDP flows in the simulation environment

each value of number of flows, a connection pattern is then created. The resulting environment is then applied to all evaluated proposals, and to a plain reference without any charging protocol running. This additional run will be useful in identifying the impact of each proposal. For each combination of mobility speed, routing protocol, transport protocol and number of flows, 100 independent runs are generated and applied to all proposals (plus the reference network). The results presented are the average of such number of runs.

In the following sections it will be analysed the performance of the different proposals using several metrics. Each metric will be analysed by variation of several factors such as network load, mobility or routing protocol. Network load is an important variable because it reflects how each solutions performs under different traffic patterns and loads. It will be implemented by increasing the number of flows generated in the scenario. Mobility is important, because it stresses both the routing protocol and the charging protocol which must adapt to the new infrastructure. Other aspect is the routing protocol in use. The different proposals behave differently in AODV or in DSR. SACP and SCP can use the routing header added by DSR instead of adding a new one. This will potentially result in lower overhead for SACP and SCP when using DSR. In opposition, PACP adds its headers independently of the routing protocol, which may result in higher overhead when using DSR.

4.4.3 Network Performance

As additional data is transmitted on the networks, charging protocols will degrade the performance of the network. Considering the current scenarios, one parameter which contributes to indicate the performance is data packet delivery ratio. This ratio is calculated by the ratio between data packets received on a network with a charging proposal and the same network without any charging support. Calculating the ratio between received and sent packets could also be interesting but this method was found to be better since the influence of other aspects like mobility, number of flows, routing

and transport protocol is removed. Considering mobility, it is assured it will have some impact on packet delivery ratio. However, given the same network with same mobility pattern, it is interesting to analyse how different proposals behave.

Mobility

Mobility affects the delivery ratio of the network, independently of the existence of charging mechanisms. Higher mobility will undoubtedly stress the routing protocol generating more routing messages. Also, higher mobility will imply routes will start to shorten as it will be more difficult to maintain longer routes. Because charging proposals are affected by the route length, as this value varies, the performance of the network may also be affected. In very high mobility scenarios, the routing protocol may start to be replaced by direct delivery. Higher mobility results in higher probability of nodes reaching in close contact, delivering packets directly or just using a small amount of hops.

Figure 4.13 depict the variation in the number of hops reported in proofs. These values are not the real length of routes but the length in reported proofs which may be slightly different. If the reporting node is closer to the Access Router, proofs will be delivered easily. In the contrary, if the forwarding node is distant from the Access Router, it will be more difficult to create a stable route to deliver reports. These proofs will have less probability of being delivered during the duration of the simulation. If enough time is given, or the mobility decreases, all proofs will eventually be reported.

According to results presented (refer to figure 4.13) the number of forwarding nodes rapidly decreases as the mobility increases. TCP is unable to differentiate link failure from congestion and, independently of the existence of network congestion, mobility will badly degrade performance of TCP [holland02]. Because UDP is a simple protocol, lacking any adaptation to network conditions, it will achieve higher throughput. Also it will allow the delivery of data through a higher number of nodes. UDP packets are only dropped if queues are full and no route to destination exists. TCP packets are shaped according to TCPs' congestion control mechanism and rate will decrease if the delay or loss increase.

PACP is the proposal reporting proofs with fewer number of hops. SACP achieves slightly higher values while SCP is the proposal reporting longer routes. The reason why PACP reports lower route length is due to the higher proof size of PACP. Larger

proofs will require more reports to be issued than SCP and SACP. Because most of the flows are leaving the Access Router, the ones with shorter lengths will have higher probability of getting their reports received. On the opposite, flows with longer routes will require their report packets to be forward by a larger number of nodes and have a smaller probability of being delivered. The same principle can be applied to all proposals. This is also one effect contributing to reduce effective route length with the increase on load in networks with secure charging.

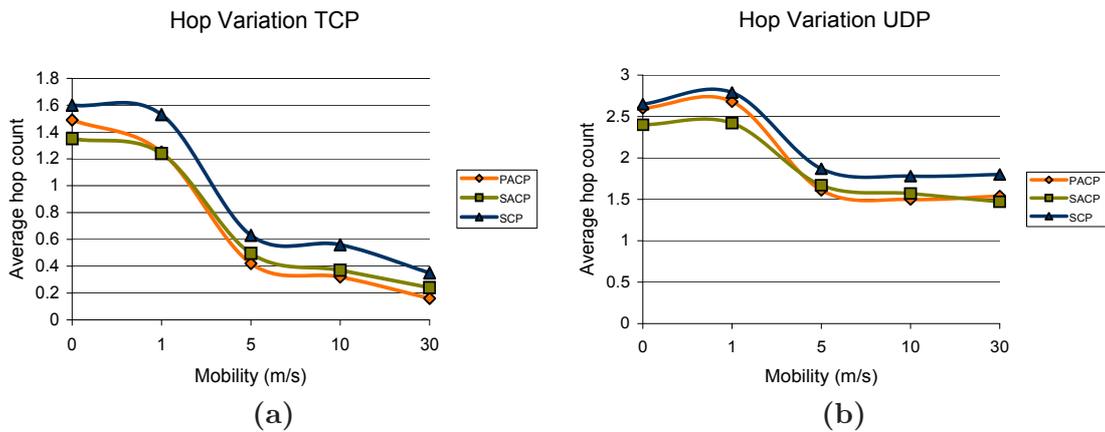


Figure 4.13: Evolution of the average number of hops in reported proofs. Using (a) 10 TCP and (b) 10 UDP flows

The impact of mobility on the performance of each proposal is depicted in 4.14. The results obtained do not try to state the impact of mobility on packet delivery but the impact of charging proposals with variable mobility. Although relative performance charts show a pronounced decrease with increasing mobility, this does not states mobility has no impact on packet delivery ratio (actually is quite the opposite). Adding charging protocols to a network with TCP flows seems to create an almost fixed penalty of 11%, even without mobility. All proposals achieve similar results with a maximum difference of only 3% when nodes are moving at $30ms^{-1}$. The fixed penalty can be related to the overhead produced by the additional control data in the network. At $30ms^{-1}$ SCP seems to differ from other proposals but it is not clear if this tendency would be maintained. UDP flows present results slightly different from TCP. The trend of all proposals is clearly defined, yet, there is a bigger difference between protocols. PACP achieves between 2 and 4% difference to SACP, with an average ratio of 93.5%. SACP is the second with an average ratio of 91%. SCP is the proposal with higher impact in performance with a difference varying between 2 and 6% of PACP. As mobility

increases, PACP and SACP show only a slight increase while SCP increases its packet delivery ratio by a higher value. This happens because SCP is highly influenced by the number of hops in the route. The shorter the route, the lesser the overhead resulting in higher performance. SACP is only slightly influenced showing also a small variation, while PACP shows a variation of only 1% between a mobility of $1ms^{-1}$ and $30ms^{-1}$.

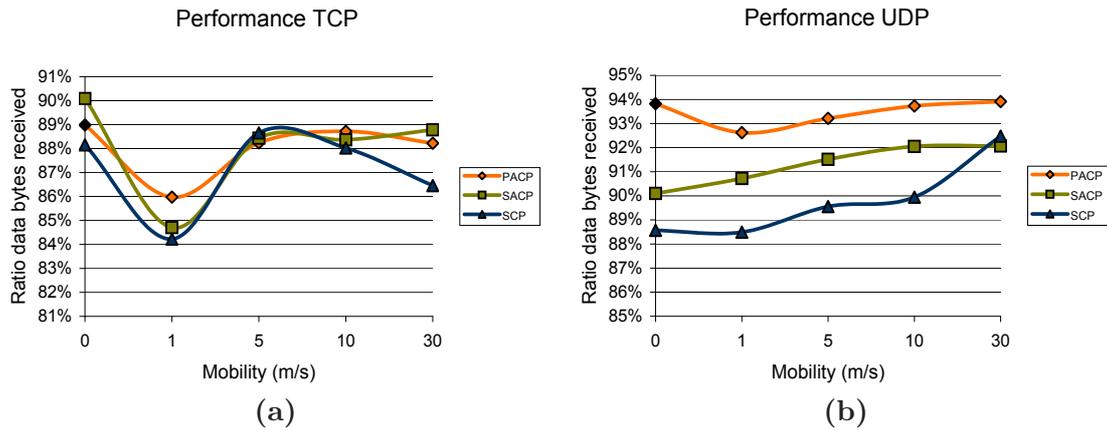


Figure 4.14: Variation of the performance ratio of (a) TCP and (b) UDP flows with the increase of mobility.

Routing protocol

Figure 4.15 depict the impact of the routing protocol on the performance of different proposals, on a static scenario. Because SACP and SCP behave differently in AODV and DSR the ratio of received packets vary. The variation is only slightly visible with TCP flows. UDP flows seem to be depend highly on the routing protocol, showing higher differences.

PACP shows a slight degradation in performance using DSR over AODV both with UDP and TCP flows. SACP shows almost the same impact with TCP flows, with only a small degradation of DSR of AODV. With UDP flows, SACP is able to achieve a ratio of packets delivered 4% higher, when using DSR. The cause of this difference is the usage of the route in *RouteUpdate* messages contained in the DSR header. SCP is the proposal showing higher dependence on the routing protocol. With TCP, SCP is able to perform around 3% better while with DSR, the gain reaches 10%. Both with TCP and UDP, SCP is the proposal with higher performance when using DSR. The reason of such increase is also the recycling of DSR headers, avoiding the inclusion of

more control data.

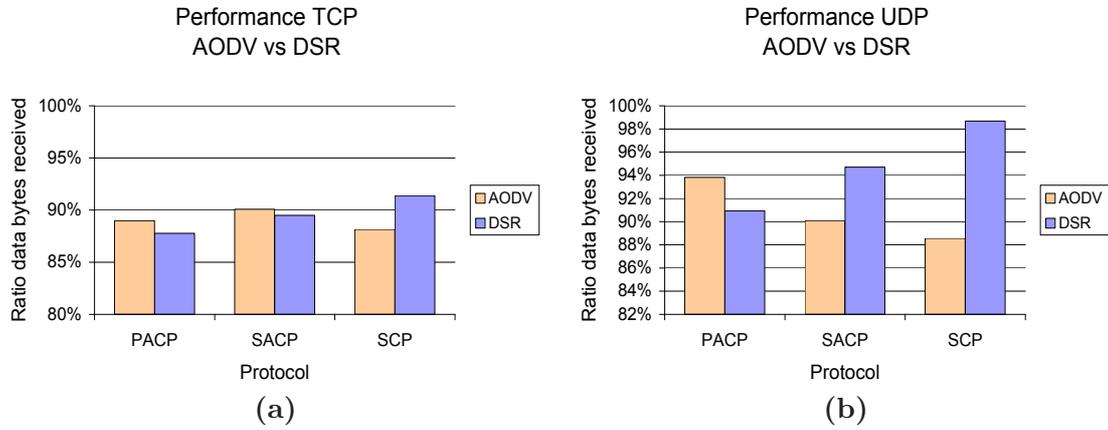


Figure 4.15: Performance ratio of the different proposals over AODV and DSR using (a) TCP and (b) UDP flows

Network load

The load on the network will influence the performance of the network due to the additional control data added. This parameter is especially important as the network becomes congested. In such scenarios, some control packets are lost requiring retransmission. Although retransmissions are required in order to reliably deliver messages to the destination, it costs bandwidth and affects the delivery of data packets.

Figure 4.16a depicts the impact of each proposal under varying network loads, when using TCP flows. The penalty of adding these charging proposals is not constant, showing a small increase with the increase in network load. The difference from a network with 1 flow and with 30 simultaneous flows is close to 10%. Individual values of each proposal are very similar showing almost no difference between them. As with other performance results obtained, UDP shows higher variation than TCP. Figure 4.16b represents the impact of the proposals under load of UDP flows. In contrast to TCP, proposals present clear differences from each other. PACP is the proposal showing less impact on performance, followed by SACP and finally SCP. It is also clear the increasing impact of performance with the increase in network load. This is due to higher concurrency in accessing the medium which results in retransmission of control messages.

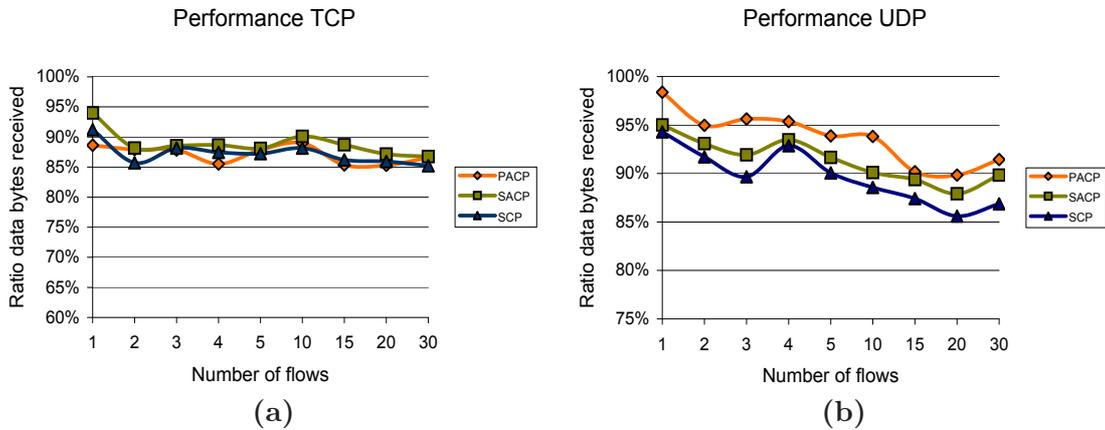


Figure 4.16: Performance ratio of the different proposals using (a) TCP and (b) UDP flows under various values of network load

4.4.4 Control Overhead

Control overhead is defined as the ratio between control bytes and data bytes received. The fewer bytes are added, the more efficient the proposal. The usage of signatures in packets is one of the major contributors to overhead, other is the protocol messages themselves. Because the cryptographic methods are very similar, in order to further isolate each solution, the usage of cryptographic methods was not simulated. Because the same amount of overhead would be added to all packets, no discernible differentiation would result. Simulating the creation and verification of ECDSA signatures would only increase the running time of simulations. As with network performance, the overhead may vary both with mobility, routing protocol and network load.

Mobility

Overhead of the reporting process should be only slightly influenced by mobility. However, marking, especially in the case of SCP and SACP, will vary with changes in the route and in its length. The higher the route length, the most overhead should be added by SACP to all packets. In the same manner, as changes in routes increases, more additional signalling is required by SACP. This additional signalling is required in order to notify the sending node of route changes. Also, every time the sending node detects a change in the route, it will issue a *RouteUpdate* message inband. SACP marking overhead is also influenced by the length of the route, increasing linearly with

the increase in the number of forwarding nodes. As stated previously in section 4.3, the increase on mobility will reduce the number of forwarding nodes which should be reflected in both SACP and SCP. The values obtained for marking overhead when using TCP flows is depicted in figure 4.17. Because PACP always adds the same amount of control data to all data packets, it presents a horizontal line without major fluctuations. In static networks, where routes are longer, SCP adds more overhead than other proposals. With the increase in mobility, the value decreases below PACP due to the reduction in the length of the route. SACP maintains the overhead almost constant with also a straight line. It is expected SACP inband overhead to increase together with mobility. However, because the route length also decreases, this is able to balance the route length. Results of the same simulation environment using UDP flows (refer to figure 4.17 are very different from the ones obtained for TCP. SCP is the proposal producing most inband overhead under all mobility values. Moreover the value increases both for SCP and the other proposals. PACP achieves a reduction of 10% over SCP while SACP is the most efficient solution. In contrary to the results obtained with TCP, the overhead of PACP and SACP is not independent of the mobility. This is because of the inexistence of contention mechanisms, more packets are dropped and never reach the destination. However, control overhead was added to these packets. The more the network is congested, the more the packets drop and higher the ratio between control bytes added and data bytes received. Because the NSTAP module accounts for packets at a lower level than applications, more packets are processed than the ones received at the application agents.

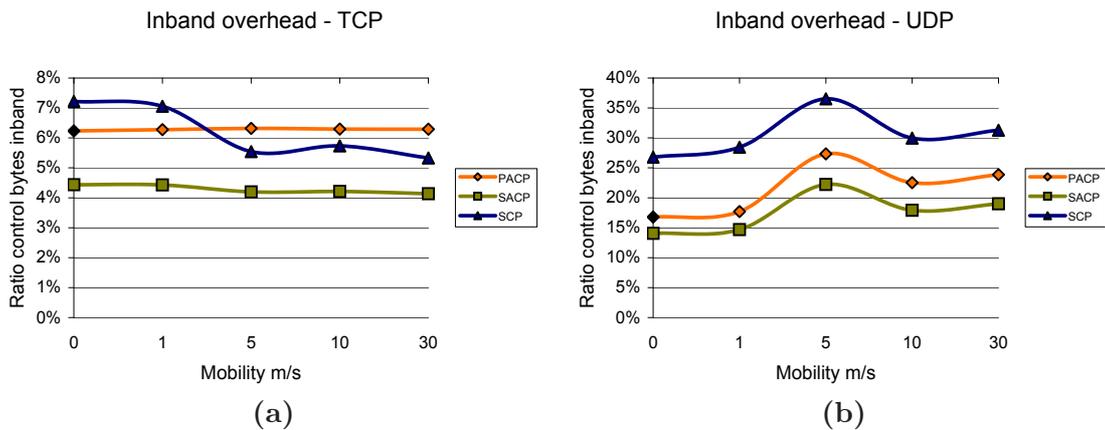


Figure 4.17: Inband overhead of the different proposals using (a) TCP and (b) UDP flows with increasing mobility

Out of band overhead is produced by the reporting process of all proposals and the route change notification of SACP. As the mobility increases more routes are created requiring more report packets to the Access Router. The results obtained for TCP and UDP are respectively depicted in figures 4.18. Result of TCP are very similar in all proposals showing a decrease in all solutions. Such decrease is due to the increase in directly delivered packets. As 2/3 of flows are from the Access Router, when delivered directly, they will not require any report to be issued inside the ad-hoc network. With UDP flows, SACP is the protocol producing more overhead maintaining a clear increase as mobility also increases. Retransmissions of report packets are one contributor to this increase which affects all proposals. SACP is additionally influenced by mobility because of the route change notification mechanism. Under high mobility ($30ms^{-1}$), PACP is most efficient solution in terms of out of band overhead.

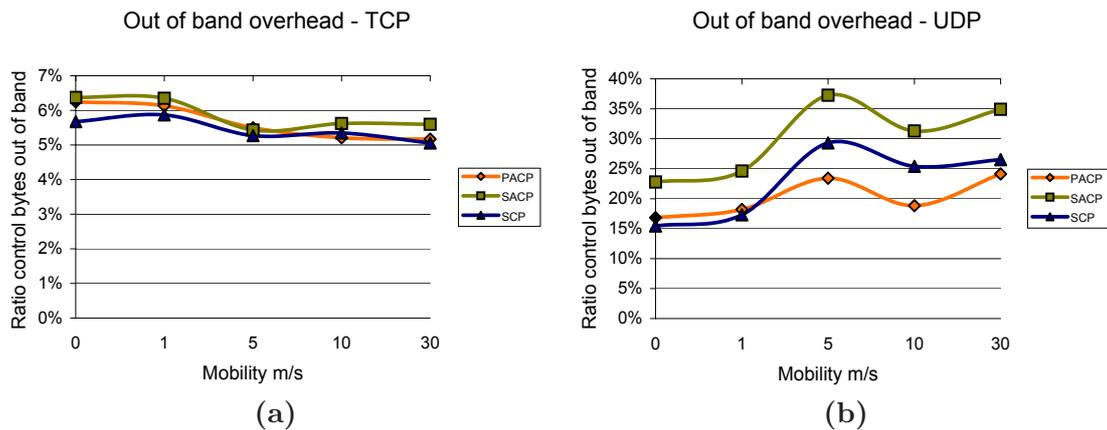


Figure 4.18: Out of band overhead of the different proposals using (a) TCP and (b) UDP flows with increasing mobility

Considering the results obtained for inband and out-of-band overhead, figures 4.19 represent the total overhead. When using TCP flows, SACP is the proposal producing less overhead with a difference less than 2% from PACP (the one producing most overhead). With UDP flows, the difference between solutions is more clear. PACP is always the most efficient solution while SCP is the less efficient. It should be noticed that the difference between them is of almost 10%. This can be claimed to compensate for the 2% difference of TCP flows, making PACP the most efficient proposal in the presented situations.

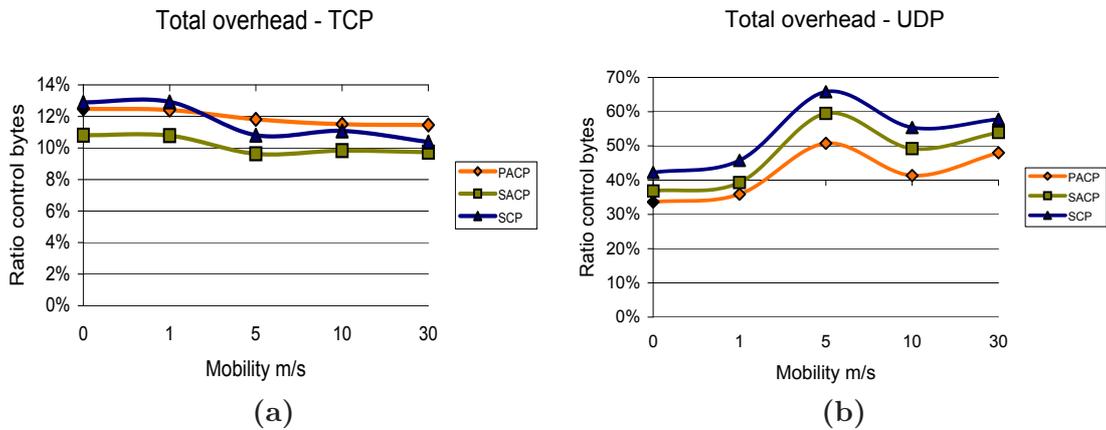


Figure 4.19: Total overhead of the different proposals using (a) TCP and (b) UDP flows with increasing mobility

Routing protocol

Changing the routing protocol from AODV to DSR will also influence overhead. SACP and SCP are optimised to source routing protocols while PACP isn't. It is thus expected the differences obtained in performance to be related to differences in overhead as well. The reporting process suffers no influence from using DSR, however the marking process is slightly different in SACP and SCP.

The results of inband overhead obtained while testing the proposals over AODV and DSR, with no node mobility, are depicted in figure 4.20. As expected PACP shows no relevant difference from changing TCP and UDP flows over either DSR or AODV. The same is not true to the other proposals. With TCP, SACP show a little decrease in overhead using DSR, while with UDP, the decrease is well pronounced. Such difference comes from the re-utilisation of DSR headers in *RouteUpdate* messages. SCP is the proposals showing the highest benefit from using DSR. As all charging headers are small size, the overhead is reduced. This does not mean there is no hop list being transmitted on packets, simply such data is added by DSR and not accounted as charging overhead.

Because of the optimisations performed by SACP and SCP, the total overhead, including inband and signaling, of these solutions with DSR is always less than with AODV. The values do not differ much when TCP flows are present. The result for UDP flows are more differentiated with a clear difference in overhead, especially for SCP. The reason for TCP showing almost no difference is due to the congestion avoidance

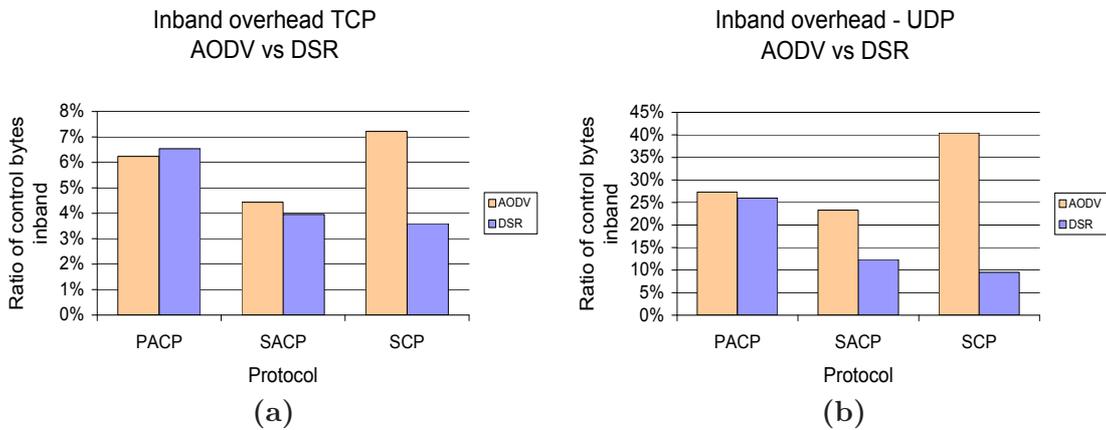


Figure 4.20: Inband overhead of the different proposals using (a) TCP and (b) UDP using AODV and DSR

mechanisms. When delivering packets to neighbours, all proposals induce the same overhead, independently of the routing protocol in use. This is because there is no necessity of sending the list of hops, which is the only thing optimised by DSR. TCP congestion avoidance, as shown in the analysis of performance, has the dramatic effect of much reducing the route length of established flows. TCP packets from flows with longer routes will suffer higher delay and packet losses thus leading TCP to reduce the packet rate of the affected flows. With small route lengths, the differences of the several proposals become smaller, resulting in the variations observed. On the contrary, UDP does not regulate flows to lower packet rates in order to avoid congestion, allowing for longer routes and clearer differentiation of each proposal. Also, as the average UDP packet is much smaller than a TCP packet, the relative impact of the charging protocols is more perceptible.

Network load

Network load has no direct effect on overhead. All proposals operate in the same manner whether the network is highly congested or without load. However, adding more flows to the network makes the available bandwidth scarcer. Packet collisions become more frequent, both requiring retransmission of TCP and control packets. Retransmission of control messages has the effect of worsening the already congested situation of the network. The implemented exponential backoff is able to minimise this effect, however it is impossible to really avoid it.

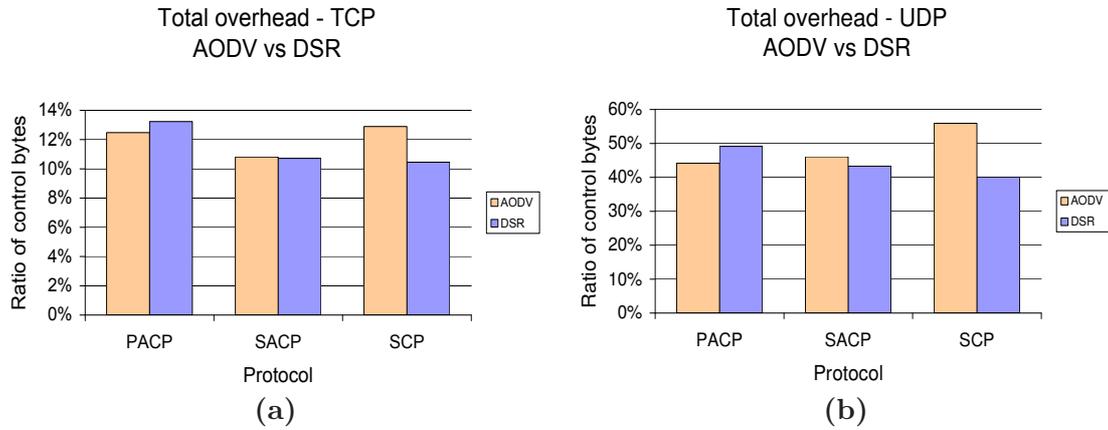


Figure 4.21: Total overhead of the different proposals using (a) TCP and (b) UDP using AODV and DSR

TCP interprets packet drops as a sign that the network is congested and starts to decrease the packet rate at which packets are sent. Packets of longer flows will have a higher probability of collide. The result is that the packet rate of longer flows will severely decrease.

As shown in figure 4.22, the inband overhead of PACP and SACP is almost constant with the increase in the network load, but not for SCP. The route shortening effect is also present in this case, apparently lowering the inband overhead of SCP. In both UDP and TCP flows, SCP always seems to add more inband data to packets than the competitors.

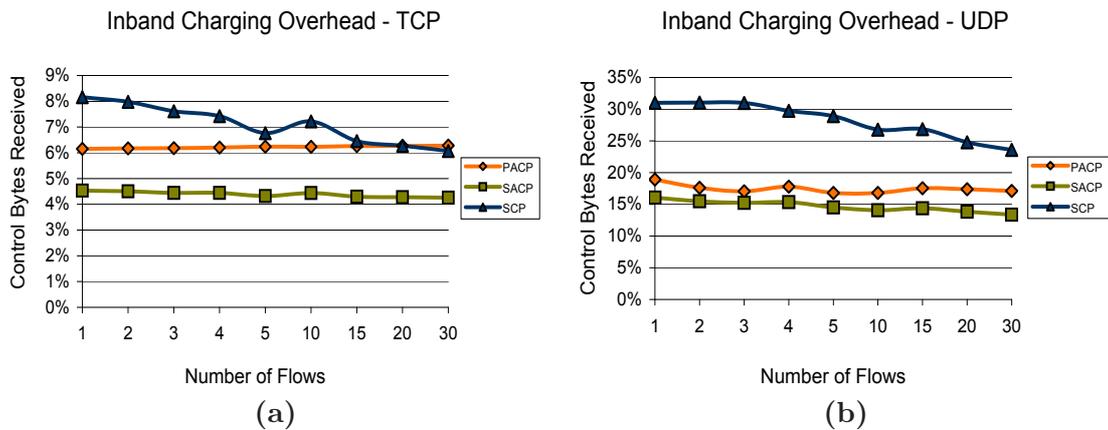


Figure 4.22: Inband overhead of the different proposals using (a) TCP and (b) UDP while increasing the load in the network

The values for out of band overhead are completely different from the previous. Besides a small tendency to decrease, overhead with TCP flows shows no substantial difference between proposals. PACP reporting process is slightly less efficient than others, which results in values higher than SCP. Even without mobility SACP frequently sends *RouteUpdate* messages. Also, fluctuations in the wireless link may force routes to be rewired by other nodes. Both mechanisms create additional out-of-band overhead, not found on other proposals. PACP overhead is almost constant and relatively low, while SCP presents the best results of overhead in relation to network load. However, as congestion increases, SCP two reporting phases is more vulnerable and may results in higher overhead figures. This tendency is clearly noticed as the number of flows increase. For values higher than 15 flows, SCP surpasses PACP in terms of overhead making PACP more suitable in networks with high congestion. This behaviour does not mean PACP is really more efficient in reporting (because it isn't). At this stage, overhead is not created because in PACP the protocol is caching proofs for later delivery. Same happens in other proposals, however with less impact. This aspect will be further discussed in the next section.

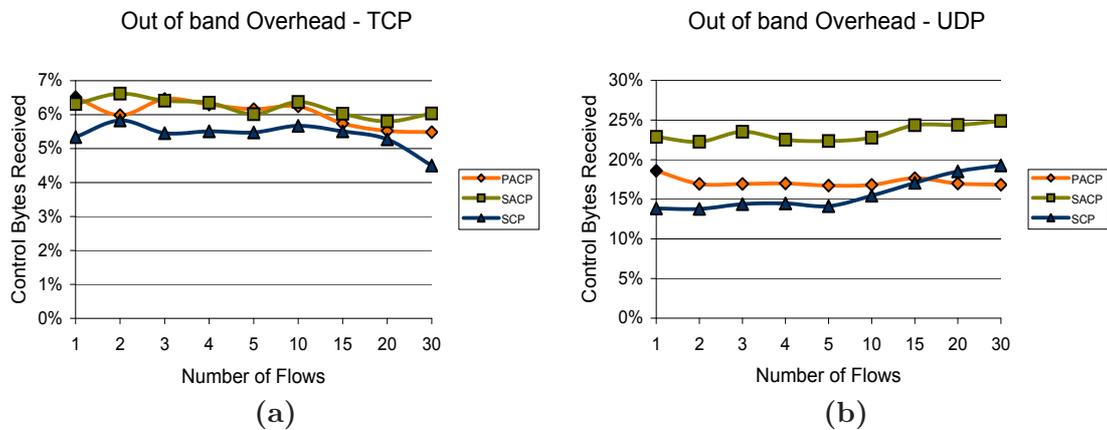


Figure 4.23: Out of band overhead of the different proposals using (a) TCP and (b) UDP while increasing the load in the network

Adding the values from both graphs, two distinct conclusions are obtained (figure 4.24). For TCP flows, there is no substantial difference between proposals. The congestion avoidance mechanisms plays a dominant role in limiting the total network congestion, also shaping the graphs displayed. UDP flows do not possess this capability and the impact is more visible. PACP is the most efficient proposal followed by SACP with just 4% more overhead. SCP becomes the less efficient with a steady difference of

10% from PACP. Interestingly, the total overhead with UDP seems to follow a tendency to decrease with the load in the network. This is related to the effective reduction on route length (this graph only compares completed sessions).

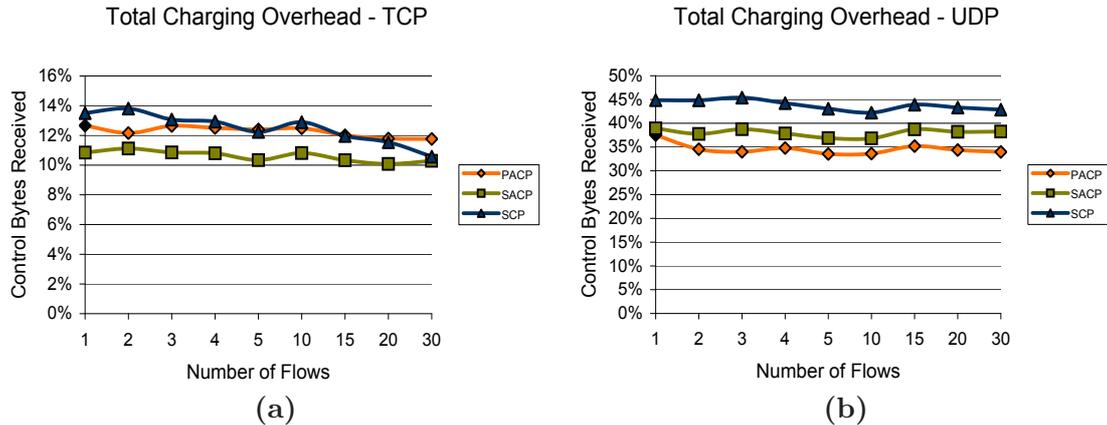


Figure 4.24: Total overhead of the different proposals using (a) TCP and (b) UDP while increasing the load in the network

4.4.5 Charging Rate

Charging rate is defined as the ratio between the number of proofs reported successfully and the total packets to be charged. While there is only one definition, it can be calculated in different forms depending on the business model.

One common method to charge users in IP networks is to charge the receiver of data. It is assumed the receiver requested information and is consuming a service, thus it should be charged. According to this business model, the charging rate should be calculated as the ratio between charged packets and the number of packets really received.

Other popular method, especially in broadband access, is to charge the sender. While services made available to the public focus on the downstream capability, operators are really more concerned about the upstream than downstream usage. This is one of the reasons why broadband is highly asymmetrical. Most of the common user generated traffic was on the downlink. Currently P2P communities start to making this concept inadequate preferring symmetric connections. However, operators are limiting the availability of symmetric connections only to enterprise environments. Having a

connection with 2Mbits of downstream bandwidth is relatively cheap while a connection with 2Mbits of upstream bandwidth is really much more expensive. In some cases besides limiting the upstream bandwidth to low values, operators also charge packets sent. According to this business model, users are charged either by the packets sent or received. The charging rate is defined as the total number of packets charged by the number of packets sent or received.

In cellular networks, this model is also applied to data packets over GPRS or UMTS. However, in the same cellular networks, the most common business model states that the initiator is always the one charged. While in cellular networks with a centralised management infrastructure this is easy to implement, in distributed packet switched networks this is more difficult to achieve. The only commonly found method is to use explicit application signalling and perform charging based on session establishment. This method is usually only found on VoIP applications.

In a MANET environment the business model poses additional concern. The network is not under direct control of an operator and may present anomalies. Example of such anomalies are congestion, unreachable destinations, flapping routes and high delay.

Congestion is related to the load in the network and its effects are an increase in packets dropped. Unreachable destinations also result in packets being drop but may be caused by many different factors. A node malfunctioning, unstable radio medium, unidirectional routes, interference or mobility are factors causing unreachable destinations. Flapping routes refers to unstable routes temporarily dropping packets and may also be caused by the previous factors. High delay can be caused by network congestion, by nodes with low processing capabilities, interferences and many other factors.

When a packet is lost and it was (or was going to be) charged, an error occurred. If the sender is paying for the packets sent, if a packet is dropped, the sender will be charged for a service never realised. If the sending node or a forwarding node, intentionally or because of a malfunction, drops all packets received, nodes would be depleted of credits without using any service. If the receiver is charged for a packet received, three things can happen. In the first situation, the packet is dropped before it reaches an accounting point (the last forwarding node in this case). Such packet is not charged or rewarded, but some nodes may have spent resources in forwarding it and will never be rewarded for that. The packet may reach the accounting point and be dropped before it reaches the receiving node. Actually it can reach the forwarding node

and be stopped by a firewall or a malfunction. The receiving user would be charged for the packet, however the message was never really received. Users which forwarded this packet would be correctly rewarded for the effort. The third case corresponds to a node receiving a packet correctly and being charged for it. This is the ideal situation.

Such problems exist at some level in all networks. The reason they are relevant in MANET is because packet losses are not an temporary exception to network behaviour, but a normal occurrence. During those occurrences the charging process will have some error associated. The most effective solution to handle this problem would be something along the lines of SPRITE: all nodes send information to the “bank”, which is then correlated. As usual, efficiency is very different from effectiveness and the overhead here required is overkill in wireless links. Because this issue is completely unrelated to the routing protocol, only the effects of mobility and network load are analysed on this thesis. The results obtained for charging rate while varying the routing protocol were obtained but omitted due to their irrelevancy.

Mobility

Mobility affects the efficiency of the charging process because of increasing packet drops. TCP flows react to the losses and limit the error by limiting packet rate. UDP flows are more vulnerable to this issue. Other aspect is that as mobility increases, it may be more difficult to deliver proofs and the charging rate will have a tendency to drop. This is particularly true for SCP and PACP. The first requires acknowledgement of the proofs. If the receiving node moves out of reach, proofs will never be able to be delivered. PACP is less efficient in reporting proofs than SCP and SACP because of the higher proof size. From the simulations executed, two business models were taken in consideration: charging sent packets and charging received packets.

Figure 4.25 depicts the results obtained for charging accuracy while varying the speed of nodes. For the sake of brevity, only the results obtained with TCP while charging packets received, and UDP while charging packets sent are represented. Results using TCP flows and charging the sender and UDP flows while charging the receiver, are extremely similar to figure 4.25a and were omitted. With the increase in mobility, as expected, the charging rate also changes. Because curves are never higher than 100%, this means the efficiency of the reporting process is having a huge influence on this figures. If packets were lost between the last forwarding node and the receiving

node, the charging rate could rise above 100%. This is also true for UDP traffic and even TCP when the sender is being charged. TCP limits the error but also makes it similar from the point of view of both endpoints.

Charging the sender for UDP flows presents very different results. The charging rate is very low and decreases with the increase in mobility. This is a result of packet losses before the packets reach the last forwarding node (packets are never charged), together with congestion making some reports to be also dropped. The first explains the low values, while congestion accounts for the tendency to decrease presented.

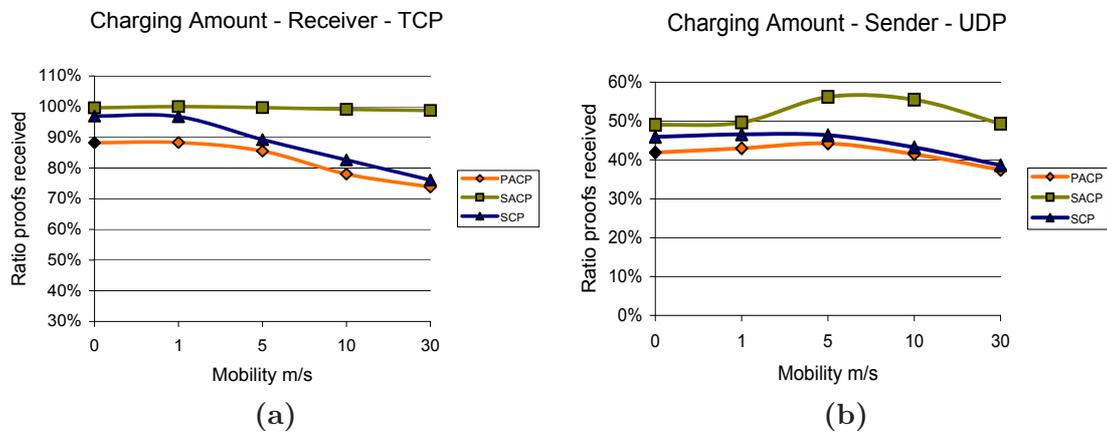


Figure 4.25: Efficiency of the charging process with increasing mobility. Sub-figure (a) relates to TCP flows charged to the receiver, while (b) relates to UDP flows charged to the sender.

Network load

The load in the network has the effect of increasing delays and collisions. As already explained in this section, collisions induce error into the charging process and should be analysed.

As presented in figure 4.26 the increase in the load is highly related to the accuracy of the charging process. TCP flows seems to be somewhat unaffected but there is a clear differentiation between proposals. PACP always achieves lower charging amount than other proposals. SCP, which was shown to be much affected by mobility, seems to do not suffer from the increase in load. This is probably due to the close proximity between the last forwarding node and the receiving node.

Results for UDP are, once again, different from the ones obtained by TCP. If the operator is charging the receiving node, congestion will increase packet loss resulting in

the user being charged for more packets than he actually received. The only exception to this case is PACP which maintain steady, although lower, percentage. For TCP, PACP showed the charging rate to decrease with network load; for UDP such tendency is not showing, because it is compensated by packet loss. If enough time would be given in the simulations to all proposals to report all proofs back, the results of PACP would be similar to the ones obtained for SACP and SCP.

When the sender is being charged of UDP flows, the results show a huge difference from the other cases. Many packets are being dropped and never reach the last forwarding. The error increases with mobility and, even for a network with 1 flow, it is always much lower than 100%. In a carefully deployed network, it is expected to exist routes between all nodes, but this was not always the simulation case. Nodes were deployed randomly, the movement pattern was randomly created and the flow end-points were also randomly chosen. There are no guaranties that the created flows can be established resulting in the high values of packet drops.

4.4.6 Rewarding Error

The Rewarding Error is defined as the ratio of incorrectly rewarded (or simply not rewarded) proofs in function of the total number of proofs received at the AR. This parameter is not the ratio between the actual proofs rewarded and the total packets forwarded. Such definition could be used, however its results are strongly related to the ones observed for Charging Error. Instead, the Rewarding Error analysed is able to predict the error inherent to some of the implementation choices of the proposals.

SCP charges and rewards all proofs at the same time and the operator always knows the list of forwarding nodes corresponding to those proofs. In SACP this is also true, however it is possible the list contains one or more incorrect entries. Such incoherences exist because when a route changes, packets are forwarded by nodes not identified in the last *RouteUpdate* message of that flow. They will issue a *RouteUpdateRequest* but packets will still be allowed to be forwarded for some time. These packets will be charged correctly, but rewarded to the wrong users. PACP suffers from the problem that the list of hops can be unknown. If the number of proofs of a flow is less than the number of hops, there is no possibility for the Charging Manager to recover the proofs. End-points are identified, and charged according to the business

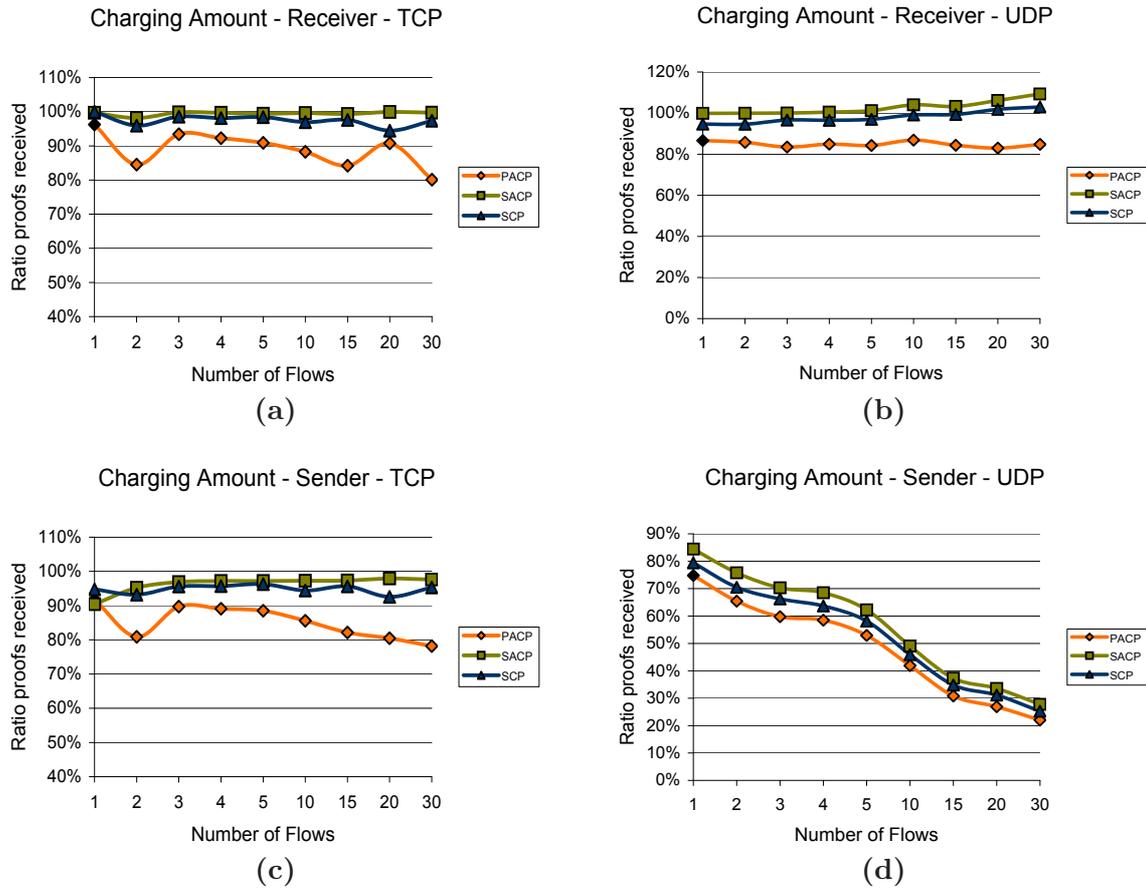


Figure 4.26: Variation of error in charging hosts both for (a) TCP and (b) UDP with increasing load. Sub-figures (a) and (b) relate to flows charged according to the receiver, while (c) and (d) relate to flows charged according to the sender.

model, however the forwarding nodes are not. The errors of both SACP and PACP are expected to be highly dependent on the mobility but, even when nodes are moving at $30ms^{-1}$, its value should be acceptable by users. As with Charging Rate, the routing protocol has no discernible influence on the accuracy of the rewarding mechanism, so, its analysis will not be presented.

Mobility

Mobility is one of the most important variables shaping the Reward Error. As the velocity of the nodes increase, the error is expected to also increase proportionally. In SACP this happens because the route changes more frequently which always induces

some error. In PACP rapid changes in the route can be problematic if the packet rate is low. Not enough packets are forwarded in each individual route and the Charging Manager is unable to identify the forwarding nodes. Figure 4.27 depicts the error obtained for TCP and UDP with increasing node velocity. Both proposals are directly affected by mobility. SACP error, even without mobility, is higher than PACP and this difference increases as the mobility increases. The maximum rewarding error obtained by PACP is below 1.5% and usually stays below 1%. SACP error is much higher, reaching 3% for a network with nodes moving at $30ms^{-1}$. The values obtained are small in relation to errors affecting the ratio of packets forwarded and rewarded such as packet loss. It is expected nodes will still cooperate in the forwarding knowing that only 1% or 3% of the traffic will not be adequately rewarded.

Higher values of error are possible, however, according to the results obtained, error should not reach values making rewarding unattractive to users. Moreover, networks with nodes moving at speeds higher than $30ms^{-1}$ ($108kmh^{-1}$) are only possible in automotive scenarios which present quite different mobility patterns.

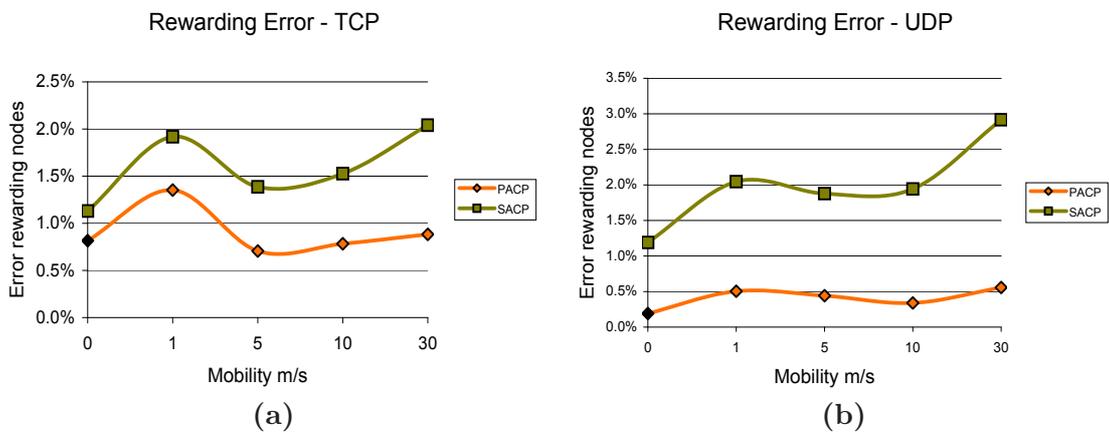


Figure 4.27: Variation of error in rewarding hosts both for (a) TCP and (b) UDP with increasing mobility.

Network load

Network load is expected to influence the Rewarding Error because of two factors: packet loss and delay. Of course, the most efficient proposal is SCP which has no error related to this aspect.

Packet loss affects SACP by requiring several *RouteUpdateRequest* messages to be sent until one is received successfully. Also, as the load in the network increases, also does the end-to-end delay. Even if the first message is successfully received, it will take more time to reach the sending node. During this period, packets are being forwarded by nodes different from the ones included in the last *RouteUpdate* message. Because the congestion increases, TCP will limit the rate of packets to congested flows. Such flows are first probably the ones with longer routes, and limiting them will decrease the average number of hops in flows. This issue has a positive effect on Rewarding Error. The less nodes there are forwarding a packet, the less error is possible to occur. PACP is affected by this issue, but not from the packet loss, because no additional signalling is required in case of route changes.

Figure 4.28 depicts the results obtained for TCP and UDP flows in a network with increasing load. TCP flows react to congestion and reduce the average route length which has the effect of reducing the error. As depicted, SACP error rate is higher than PACP, however it also decreases faster finding a common point when there are 20 active TCP flows. The maximum error corresponds to a network with only one flow, but still, the value obtained is considered to be perfectly acceptable. The behaviour of the proposals using UDP flows is different from TCP. The route length also become shorter, as the number of flows increase. However, as the delay increases, SACP will have difficulty in notifying the sending node about changes and will see its error increase with the load. Such topology changes are induced by false disconnections resulting from routing messages being dropped due to collision. Results obtained for PACP show the error to be very low, staying always below 0.2%. According to network load, both proposals are affected, however they are affected in different manners. From the two, PACP seems to be the proposal with better efficiency in identifying the forwarding nodes.

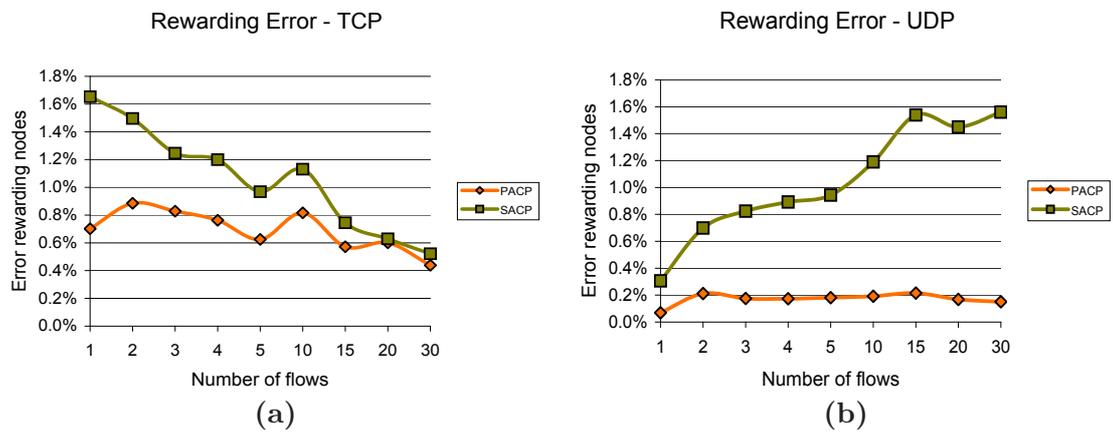


Figure 4.28: Variation of error in rewarding hosts both for (a) TCP and (b) UDP in relation to network load

Chapter 5

PACP implementation evaluation

This chapter describes the experimental work performed with PACP in a laboratory prototype. Besides the prototype NS-2 implementation, PACP was fully implemented. Every detail described in section 4.1 was implemented and the produced prototype was adopted by the IST-Daidalos project and presented in the Daidalos Ad-hoc demonstrator [sargento05]. Before the implementation started, the main modules were modelled using the Specification and Description Language (*SDL*) [ituz.100]. The complexity of the implementation was not sufficient to justify applying such methodology. However, the developed modules had to be integrated in the Daidalos architecture, which required clear specification of the interfaces and analysis of the interactions. As depicted in figure 5.1 the internal structure of both ad-hoc nodes and Access Routers is not trivial, with many dependencies between modules. PACP plays an important role in providing charging and rewarding mechanisms integrated with the central AAAC infrastructure.

5.1 Implementation details

The implementation was developed on a Linux environment using kernel 2.6.8.1. Extensive tests have been performed using kernels v2.6.3 through v2.6.16. All modules should also work with a 2.4 kernel, however no tests have been performed using such kernel. Code was developed using ANSI C/C++ having in consideration issues like portability between architectures, resource consumption and modularity.

All modules are implemented as user space daemons. This fastened the development and debugging of the implementation making it possible to focus resources in the optimisation and range of features supported. Ultimately, userspace programs maintaining a fairly high independence between kernel modifications increase the compatibility with further developments and among different systems. Threads were used where they proved to be necessary or where they could be useful in maximising performance and/or efficiency of the protocol. Locking mechanisms were carefully deployed so that concurrency is optimised and its related penalty reduced.

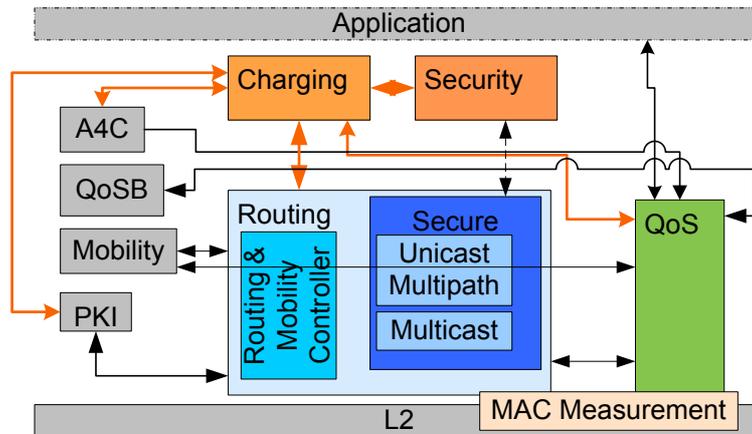


Figure 5.1: Internal structure of Daidalos ad-hoc nodes and ad-hoc Access Routers

5.1.1 Netfilter

Kernel changes were required because of limitations on *Netfilter* code in the Linux kernel. *Netfilter* provides an interface called *IP6Queue* allowing to intercept packets at different parts of the path inside the kernel. Using *Netfilters'* *IP6Queue*, an application can listen at inspection points (hooks) intercepting packets. Depending on the origin and destination, packets will be routed through different hooks making possible to selectively process packets. In combination with *iptables* rules, it is possible to further refine the criterion by which packets are sent to applications making this solution very powerful. When a packet is intercepted (i.e sent to an application), the application is allowed to fully inspect its payload, and return a verdict. The verdict indicates if the packet should proceed in the kernel, be dropped or just queued for further processing. Applications can even provide alternative payloads, replacing the original packets' content. Figure 5.2 depicts the structure of *Netfilter* with the different hooks where packets can be intercepted.

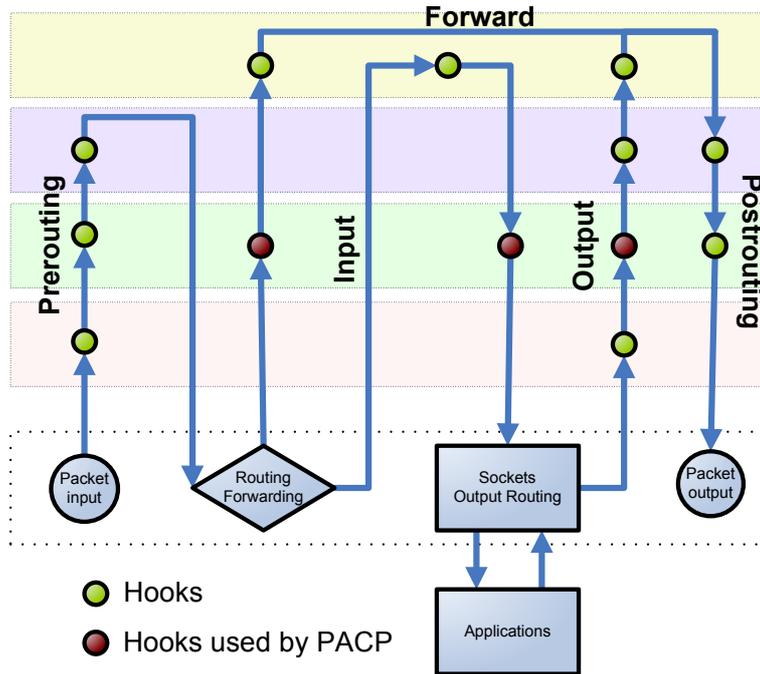


Figure 5.2: Netfilter hooks, tables and chains

PACP places rules intercepting packets at *INPUT*, *OUTPUT* and *FORWARD*. The *INPUT* hook is used to capture packets with destination address of the local host. Here, PACP performs admission control based on verification of the *MAC* and flow authorisation. Also it removes the charging header from packets. The *OUTPUT* hook is used to intercept packets generated locally and going out to the network. In this hook, PACP authorises and adds a charging header to packets. The *FORWARD* hook is used by PACP to intercept packets crossing the host but which are neither from or to it. Here, the charging header is updated, flows are authorised, the *MAC* is verified (if present) and proofs are collected.

As shown, *Netfilter* is very resourceful but there is one limitation to the mechanism: only one application can be connected to *IP6Queue* at the same time. Because several modules used this same interface, modifications to the *IP6Queue* interface were required. The modifications aimed at enabling *IP6Queue* to support multiple peers (applications) simultaneously connected to the interface, all issuing verdicts to packets.

The implemented algorithm is simple yet as efficient and effective as required. First the packet is sent to every application registered according to the order they connected. If the application issues a verdict *ACCEPT*, the packet is passed to the next application. If a *QUEUE* is issued, the packet remains on that position. If the

verdict is a *DROP*, packet is immediately dropped without further processing. Because packets could be changed by applications, the order by which they register is important. Although a more efficient method of registration could be used, allowing applications to specify the priority, that would also require changes to *libipq*, which was not desired.

5.1.2 Implementation modules

PACP was implemented using C++ and made full use of the object oriented model. Most of the classes are shared by the Mobile Node (MN), Access Router (AR) and Charging Manager (CM). Some others are only present in specific nodes. This modularity was vital for a rapid and efficient implementation.

Figure 5.3 represents the classes composing a Mobile Node or an Access Router and the interfaces between them. The main difference between a Mobile Node and an Access Router is the main protocol logic. Mobile Nodes' logic is implemented by the Charging Manager. Access Routers' logic is implemented by the Internetwork Adapter. All other classes are the same.

Figure 5.4 represents the classes composing the Charging Manager. This node is very different from both the AR and the MN. Most of the classes are different and only packet manipulation, timer management or cryptographic interfaces are shared.

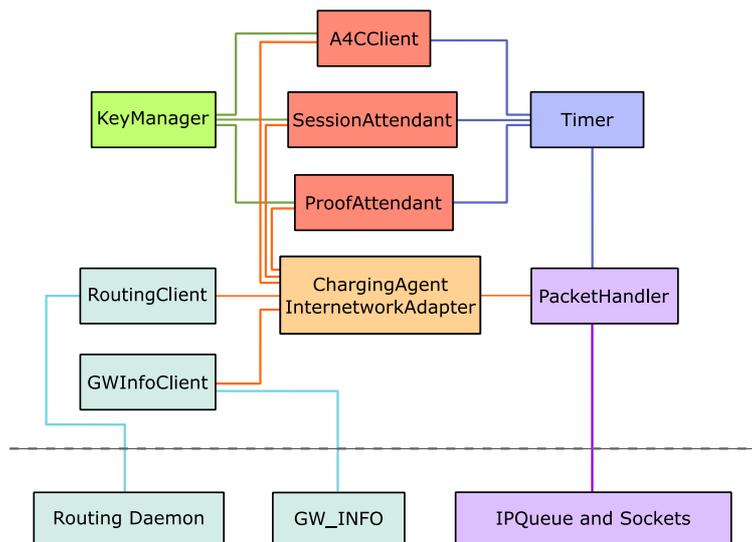


Figure 5.3: Class structure of an Ad-hoc Node (using ChargingAgent) or Access Router (using InternetworkAdapter)

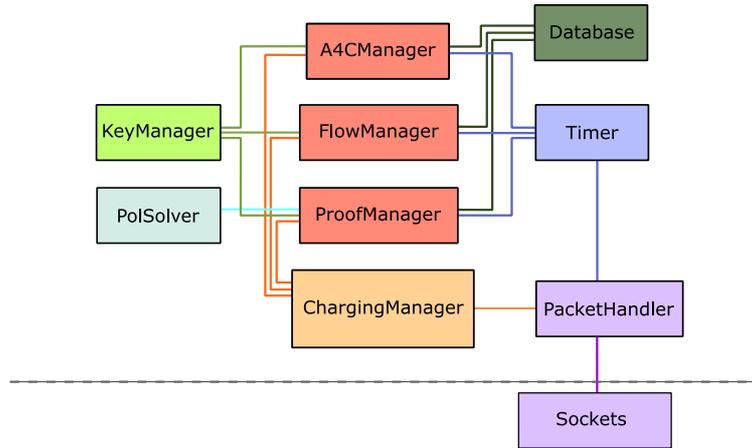


Figure 5.4: Class structure of the Charging Manager

KeyManager (All)

The KeyManager class exists in all entities and acts as an abstraction to cryptographic operations using keys. It offers various methods to (de)cipher, sign and verify messages which are used by all other classes. Also, a repository allows to cache keys and query them as needed. Other modules are totally unaware of the implemented cryptographic modules as long as the interface is maintained. This allows for further improvement of the implementation or test of other cryptographic methods.

PacketHandler (All)

The PacketHandler class is the interface between the network and other PACPs' classes. It provides methods to get packets from the network, abstracting the actual source of packet capture. Data packets are captured using *IP6Queue* while control packets are transmitted directly between sockets. In both cases, other classes will receive a Packet object correctly decoded depending of its type. It supports methods to hold and release packets which were queued until other event happened (such as authorisation). Sending packets to the network is also performed using PacketHandler. If there are data packets, it is issued a verdict to the *IP6Queue*. Otherwise, they are sent through the control socket.

Timer (All)

The Timer class is used by almost all classes to schedule events. The events to be scheduled can be either timeouts or packet retransmissions. The first type of events are used to call functions which should be called after a timeout. Examples of such methods are key renewal, flow authorisation renewal, or sending proofs. Other type of events is used to send control packets and insures retransmission of packets with configurable parameters. Retransmissions are performed until the maximum number of retransmission is reached or the event is cancelled. Packets to different destinations are maintained in different queues insuring some level of ordering at the source. If an event is rescheduled to a later time, all existing events added after it will also be rescheduled. Maximum resolution of both types of events is limited to 1ms. This results in a good compromise between timer resolution and performance.

A4CManager (CM)

The A4CManager class handles *SessionInitiationRequest* messages and tracks registered nodes. When a *SessionInitiationRequest* message is received, it decodes the message, verifies if the node is able to register the network, activates the charging process. When these processes are complete, it replies with a *SessionInitiationResponse* message to the requesting address. Verification is performed by consulting the permanent database and verifying the credentials provided.

ChargingManager (CM)

The Charging Manager class is the main class of the charging manager. It runs the main logic of the CM by invoking other classes and listening to control packets. When control packets arrive, they are dispatched to the correct class to further handling.

Database (CM)

The Database class implements an interface to a central repository of user profiles and accounting data. It is located at the Charging Manager and used by some of its classes. It provides methods capable of querying values of the database and return database independent results. This makes possible to change the database backend

without changing the application. All calls to the database are synchronous. The current implementation supports only MySQL [mysql] servers located either locally or on remote hosts. Other possibilities would be the usage of LDAP [sermersheim06] or Postgres [postgresql] systems. Also, this class could implement methods interacting with more complex AAAC systems over protocols such as DIAMETER or RADIUS.

FlowManager (CM)

The FlowManager class manages end-to-end sessions (or flows). It exists only at the Charging Manager and is responsible for processing *FlowAuthorisationRequests* and issuing *FlowAuthorisationResponses*. It uses the key repository to recover nodes' keys and the local database to verify users' profiles. The answer will depend on the profile and current status of both systems. If the source or requesting nodes have no key registered, it is assumed they are not registered and access will be denied. If any of the end-points or requesting nodes are not present on the permanent database, profiles are unable to be verified and access will also be denied.

PolSolver (CM)

The PolSolver class is fed with a set of proofs and calculates the inversion of the Vandermonde matrix based on the reported RID_i and X_i values. Because these calculations can be expensive, in order to maximise performance, this class communicates in an asynchronous manner. After the route is decoded, a callback is called in order to report the result.

ProofManager (CM)

The ProofManager class is instantiated at the Charging Manager and manages proofs reported by the last forwarding nodes. Proofs are verified and nodes are charged by this class. When sufficient proofs are gathered, they are sent to the PolSolver class for offline decomposition. Upon successfully decoding the proofs, the ProofManager will reward nodes by sending the correct order to the permanent database. After the first time a route is decoded, the process is only repeated if verification fails, indicating a possible collision of the *RHash*.

ChargingAgent (MN)

The ChargingAgent class is the main class of all mobile nodes. It is responsible for initiating other classes and processing packets accordingly.

InternetworkAdapter (AR)

The Access Router behaves almost as a normal node sharing almost all the classes. The exception is the main logic which is different. The InternetworkAdapter class replaces the ChargingAgent class found at ad-hoc nodes. It is able to interconnect the ad-hoc network with the infrastructure. Packets entering the ad-hoc cloud require a charging header to be added. Packets going out of the ad-hoc cloud require the proof to be collected and the charging header to be removed. Packet with source and destination inside the ad-hoc network are forwarded normally as if the Access Router as a normal ad-hoc node.

A4CClient (AR and MN)

The A4CClient exists at the mobile nodes and at the Access Router. It is responsible for registering the node upon join, and refresh that information. Only when the A4CClient in a registered stated, data packets are allowed to be forwarded. Otherwise the are queued, waiting for a registration, and later dropped if no registration is received or buffers are full.

FlowAttendant (AR and MN)

Each packet sent or forwarded will require some form of authorisation. This can be done automatically (i.e authorising all packets) of by explicit querying to the Charging Manager. The FlowAttendant class manages authorisation of flows, issues *FlowAuthorisationRequest* messages and handles *FlowAuthorisationResponse* messages. Session tokens are cached until they are about to finish. If the flow is active, a renewal is triggered on the background.

GWInfoClient (AR and MN)

PACP can be integrated with Jelgers' [jelger04] solution of gateway mobility. The class handling this interface is the `GWInfoClient`. Upon initiation, it connects to the mobility daemon receiving notifications about the current gateway used. These messages are used in order to identify the Charging Manager to use and the identification of the Access Router.

ProofAttendant (AR and MN)

The `ProofAttendant` class captures proofs from packets and issues Report messages periodically to the current Charging Manager. When packets are forwarded or sent and the next hop is the destination hop, proofs are sent to this class. Periodically, the `ProofAttendant` verifies the proofs pending submission and issues the proper report. Also, Report Response messages are sent to the `ProofAttendant` when received at the `ChargingAgent`. Upon reception, the proofs which were marked as being submitted are finally flushed from the cache. If no answer is received and the Report retransmission fails, proofs are marked as to be submitted and the process repeats.

RoutingClient (AR and MN)

The `RoutingClient` class provides methods making possible to query other services about the next hop a packet will take. Currently two services are supported: Plain Kernel and AODV. The first is used in static situations and gathers the required information using the Linux *RTNetlink* interface. The second should be used when AODV-UU (AODV implementation from Upsala University) [aodvuu] is used and queries the internal routing tables of the daemon directly. Although the first should be generic enough to be used in all situations, using AODV directly provides additional performance and eliminates some errors related to how AODV-UU manipulates kernel routes. In both cases, queries are cached for a configurable period. Using caches makes necessary to query the routing system less often, decreasing the delay of processing packets. After a request is issued and until a timeout is reached, the information in the cache is maintained synchronised with the routing system.

5.2 Ad-hoc testbed scenario

The implementation of PACP was tested on a IPv6, ad-hoc testbed, hosted at IT-Aveiro [itav] premises. The integrated ad-hoc testbed is comprised of several Linux computers running Mandrake 10.1. The kernel was not the one provided by Mandrake but a vanilla 2.6.8.1. Using a vanilla kernel provides higher control over the modules running and allowed better development of the required changes. All machines have between 400Mhz and 2.4Ghz CPU, 256Mb RAM, and enough storage space. They do not reflect typical, resource limited, ad-hoc nodes, but are suited to a test environment where heterogeneous machines coexist on the same network. All machines are equipped with 2 network interfaces: one wireless and one wired. The wired interface is used to provide remote access during the tests and to perform administrative tasks. Also, this enabled the usage of remote logging to a central point correlating information.

Wireless interfaces were D-Link DWL-650 PCMCIA cards either plugged directly into the PCMCIA slot of laptops or to PCI-PCMCIA adaptors in nodes lacking such slots. The chipset of these cards is the well known Prism2.5, developed by Intersil, in its versions 1.1.0 and 1.7.4 (primary and secondary). All cards were configured in ad-hoc mode using channel 12, rate was fixed to 2Mbits and RTS/CTS was enabled with threshold of 1 byte. The surrounding environment, as in most research facilities dealing with 802.11 technologies, was crowded with wireless networks. In order to reduce the external influences, most of the tests were performed without traffic in the building (weekends), and in some cases, with the nearby access points powered off.

One of the nodes is used to interconnect the ad-hoc cloud with the infrastructure network (acting as a Access Router), and here the wired interface will also be used to transfer data between the ad-hoc and infrastructure networks. The infrastructure network was a prototype of an operator manager network as envisioned by the Daidalos project. The entire Daidalos testbed is comprised of several tens of equipments providing functionalities such as AAAC, PKI, routing, MobileIP Home Agents, DNS, several application servers and over ten access networks each providing different access technology. The ad-hoc testbed represented two of the access networks.

Figure 5.5 depicts relevant nodes of the ad-hoc prototype. In the ad-hoc network, routing was provided by AODV while in the infrastructure RIP [hedrick88] was used. Because AODV only provide routing inside the ad-hoc cloud, an implementation of [jelger04], developed on the Daidalos project, provided auto-configuration and

connectivity to the outside of the cloud.

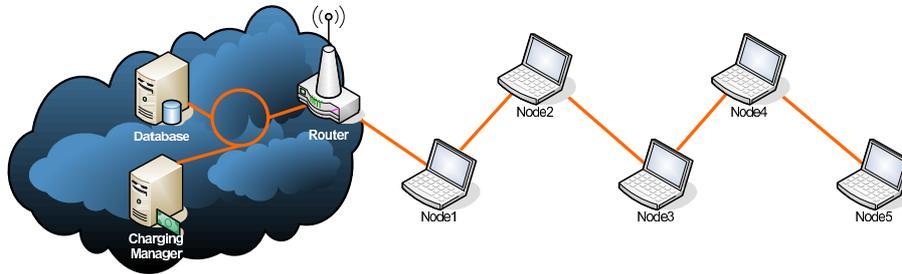


Figure 5.5: Representation of the integrated ad-hoc network testbed.

Because it is difficult to reproduce with precision high speed mobility of nodes (especially inside a research facility), mobility was not evaluated in the testbed. Such results were already extensively discussed the previous section. The nodes have been physically deployed inside a roughly square building with around 36x36m, and normal office/lab divisions. Figure 5.6 depicts the IT-Aveiro building and the location of the ad-hoc testbed.

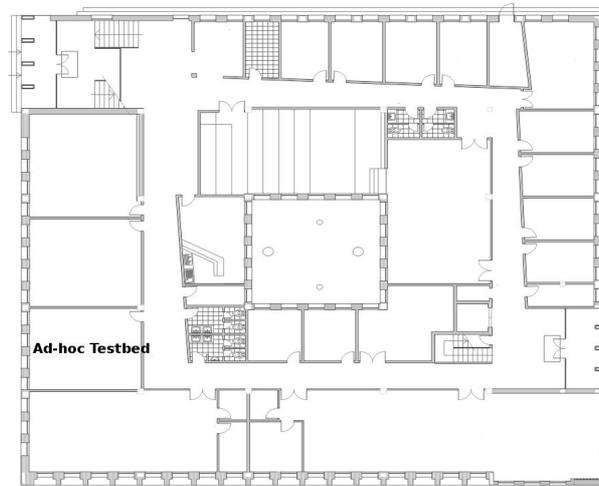


Figure 5.6: Map of IT-Aveiro and the location of the ad-hoc testbed. Nodes were distributed inside that room.

Since there is not enough physical space to create the desired topology without nodes interfering with each other, the MACKILL [mackill] tool was used to perform filtering (in kernel), based on the source MAC address, ensuring a logical string topology. MACKILL had to be adapted to the kernel version in use during the course of this thesis. Because MACKILL operates at the MAC layer, although no IP messages can be transmitted between nodes configured to appear as non neighbours, at the physical

level there are still interferences. The result is that throughput values measured will be lower than the maximum values achievable on a non-emulated topology. Because the focus of these experiments is to evaluate the impact of PACP, opposed to the throughput of the 802.11 medium, this situation was found to be acceptable. Moreover, further experiments revealed the difference in throughput to be minimal.

Experiments were performed either using MGEN [mgen] generating CBR UDP traffic, and IPerf [iperf] for TCP throughput analysis. The destination for the traffic was always the Access Router while the sender was iterated through the ad-hoc nodes, depending on the parameter being tested. Tcpdump [tcpdump] and Wireshark [wireshark] were required in order to intercept messages at various points, further processing the results at a central node. In all situations, TRPR [trpr] was used to analyse the captures originating both from tcpdump and MGEN, and provide graphs and summary results. LabPlot [labplot] provide far more data analysis capabilities than traditional tools and was used to generate some of the charts.

5.3 Delay

5.3.1 Registration

When a node joins a network, it must register with the local Charging Manager. In this process, node is authenticated, public key is stored at the Charging Manager and a shared secret is exchanged. Because the registration process is only performed upon join of a network, or to refresh keys, its duration is not critical. However, if more complex scenarios are envisioned, where Fast Handover [mccann05] mechanisms exist, this delay should be studied.

Figure 5.7 depicts the 6 nodes registering with the Charging Manager. The delay between individual *SessionInitiationRequests* is not meaningful as nodes were launched manually. As depicted, the registration delay varies between 5 and 36ms. Such variation is due to the delay in processing the *SessionInitiationRequest*, fetching and storing data in the MySQL database. Also, to these values should be added the processing delay at the receiving node. In reality, although the *SessionInitiationResponse* is received, only after verifying the *MAC* and decoding the packet, the registration is complete. Typical, 3 to 5ms should be added, corresponding to ECDSA signature verification. Analysing only this values shows PACP is able to provide very reduced registration delays. From

the results obtained, reasonable delay can be added, especially because the backend databases will get slower as the number of clients increase. Fast Handover situations could suffer from this additional delay, especially if other mechanisms are also to be supported in conjunction with PACP.

No. -	Time	Source	Destination
1	0.000000	accessrouter	chargingmanager
2	0.005064	chargingmanager	accessrouter
3	0.468200	node1	chargingmanager
4	0.036539	chargingmanager	node1
5	1.260159	node3	chargingmanager
6	0.009757	chargingmanager	node3
7	0.700968	node4	chargingmanager
8	0.025838	chargingmanager	node4
9	1.423087	node5	chargingmanager
10	0.033928	chargingmanager	node5

Figure 5.7: Packet capture showing the 6 nodes registering with the Charging Manager. Times are indicated as delay since last packet and not absolute.

5.3.2 Session establishment

Each time a session is established, if the node is unknown to the network, the session must be authorised by issuing a PACP *FlowAuthorisationRequest* message. This message is sent by the originator and by all forwarding nodes. The Charging Manager consults the database and verifies if the flow should be accepted or not. In this case the authentication data is stored using a MySQL database. The delay associated to the response from the Charging Manager, besides signature verification and generation, is highly influenced by the backend database.

No. -	Time	Source	Destination
1	0.000000	node5	chargingmanager
2	0.007507	chargingmanager	node5
3	0.098354	node4	chargingmanager
4	0.106437	chargingmanager	node4
5	0.144746	node3	chargingmanager
6	0.152488	chargingmanager	node3
7	0.215223	node2	chargingmanager
8	0.222962	chargingmanager	node2
9	0.236258	node1	chargingmanager
10	0.243926	chargingmanager	node1
11	0.260041	accessrouter	chargingmanager
12	0.268232	chargingmanager	accessrouter

Figure 5.8: Packet capture showing the 6 nodes asking for permission to forward a flow

Figure 5.8 shows a screenshot of Wireshark with the sequence of *FlowAuthorisationRequest* and *FlowAuthorisationResponse*. Node5 is starting to send a flow towards the Access Router and, as the first packet is routed, nodes start asking for authorisation. The entire process of session establishment takes 268ms after which the packets are free to flow until the authorisation token expires. In average, the Charging Manager takes 7ms to verify process the request, consult the database and issue a reply. Large delays between requests are visible, especially from Node5 and Node4. This happens because a number of routes are requested to AODV. As an example, the first request received is sent before there is a route to the destination or even AODV is aware of the packet. In a scenario using static routes or a proactive routing protocol this value is expected to be smaller. In the optimal case it will be the sum of round-trip delay between the requesting node and the Charging, plus the processing delays at the Charging Manager and requesting node.

After inspecting the token received in the *FlowAuthorisationResponse*, nodes much schedule a renewal of the token. The renewal should be scheduled to some time before real expiration. Otherwise flow would be blocked for the period of the request. As this operation is performed independently of the data packets, the impact is reduced. However because nodes only have 1 processor, while the *FlowAuthorisationResponse* is being processed, data packets are delayed. The delay introduced corresponds to packet decoding and more importantly, to verification of the *MAC*. Figure 5.9 shows the impact of a renewal, occurred at $t=100s$, to packet delay. As it is clear, other factors such as buffers or medium access contention, are introduce more variations than the renewal process.

5.3.3 End to end delay

End-to-end delay is of vital importance to be analysed. Applications dealing with bulk traffic such as HTTP browsers, email clients or ftp applications can operate with relatively high delay. Some streaming and real-time applications can also deal with delay. However, other applications generating interactive real-time traffic, such as VoIP or Video conferencing are not that tolerant. According to [itug.107] delay should be maintained below 150ms as users are usually not able to detect such low delay. Higher values can start affecting conversation, making real-time interactive services unusable.

In the current software implementation, delay can be added from four sources:

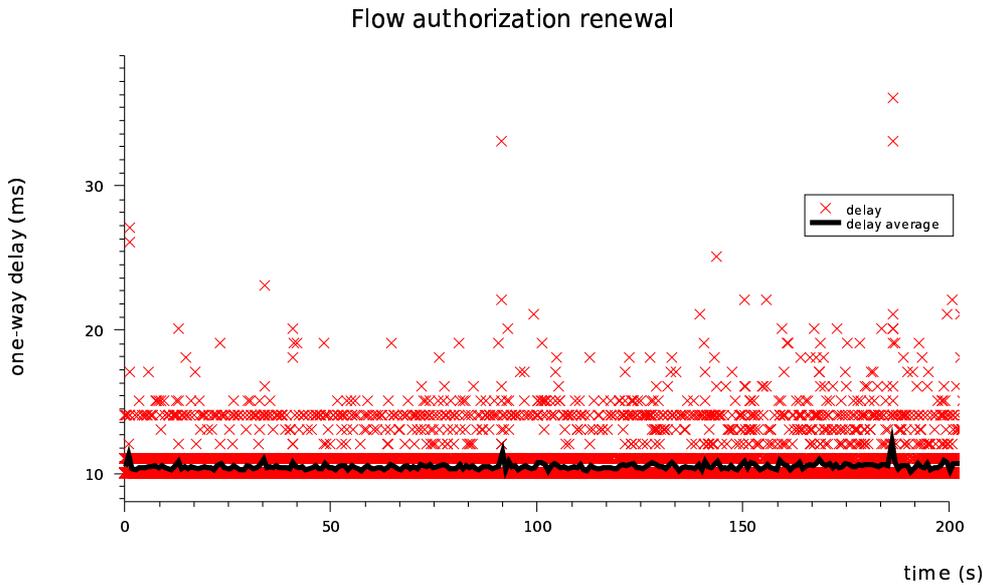


Figure 5.9: 256Kbit traffic from Node2 to the Access Router. Node1 is forwarding and issues a renewal at $t=100s$.

IP6Queue, PACP processing, Routing Protocol Query and cryptography. The total delay applied to packets will also be conditioned by available CPU resources or wireless medium access. In this case the CPU was idle and no other traffic was present.

Table 5.1 depicts the typical values obtained by the four delay sources. *IP6Queue* adds a fixed amount of delay to each packet due to transport between kernel space to userspace (and vice-versa). Internal processing by PACP adds almost the same amount of delay, both in the range of micro-seconds. Consulting the routing protocol is needed in order to verify if the next hop is the destination node. This query is performed periodically and asynchronously not impacting directly packet delay. Although this verification, as well as *IP6Queue*, are almost meaningful, they may both be penalised by context switching, adding a multiple of the context switch granularity to the delay. In the kernels used, this penalty was 1ms and will vary with kernel compilation flags. Still, if there is no previous information about a destination, the packet will have to wait for an answer before it is sent or forwarded.

In the case cryptography is enabled (generation of *MAC*), the sending node will create an ECDSA signature per packet. The forwarding nodes will perform a ECDSA verification operation. According to the values obtained with a 163 bit ECDSA key, using such methods will pose major restrictions to protocol operation. In the first place, it is expected each node to add +2.4ms to each packet upon forwarding. This will

increase the delay deterministically along the route. Because these values were obtained using a synthetic benchmark and CPU was idle, in real world values are expected to be even higher. Even considering 3ms per hop, still 40+ hops are possible without reaching the 150ms delay limit. Other aspect is that in busy networks, verifying packet signature will consume large amounts of CPU and battery. It is not clear whether nodes will simply accept all packets in order to be able to gain responsiveness and battery. One option is to use lower key sizes or dedicated hardware. Finally note that the maximum packet rate will be limited by the less capable node along the path. This aspect will be discussed on the next section.

Source	Added delay
IP6Queue	51 μ s
PACP Processing	43 μ s
Routing Query	1.5 ms
ECDSA verification	2.4 ms
ECDSA sign	0.5 ms

Table 5.1: Processing times obtained on a Athlon XP-M 1800+ (1533Mhz), ECDSA secp160r1, linux kernel 2.6.17, Openssl v0.9.8a

The results obtained for delay and jitter are presented in table 5.2. In order to synchronise machines, NTP [mills92] was used. During the course of the experiments, error was always less than 1ms between end-points. Values show that PACP in reality introduces little delay. Because machines do not have exactly the same resource capabilities the delay does not increase linearly. After 4 hops, there is a 8ms difference between a plain network and a network with PACP. Clearly context switching penalty is increasing the delay at each hop. The average delay introduced by this configuration is, in average, $\frac{8ms}{6} = 1.3(3)ms$. The denominator of the equation is 6 and not 4 because the sending and receiving nodes also process the packet and add delay. When adding message authentication and verification, the delay greatly increases. In the same scenario, after 4 hops, the additional delay is 30ms. There is 1 sign operation plus 5 signature verifications being performed, per packet. In the same Athlon XP-M 1800+ this would add $5 * 2.4 + 0.5 = 13ms$. Such difference is verified because no machine in the testbed has such processing capabilities which greatly increases the delay.

Even with these low cost machines and 4 intermediate hops, the results show VoIP calls are still possible to be performed without audible degradation to users. The average delay introduced by using PACP with MAC generation and verification, in this testbed, is $\frac{30ms}{6} = 5ms$. If every hop introduces such delay, 30 hops would be needed

for the delay to reach 150ms.

Delay (ms)	1hop	2hop	3hop	4hop	5hop
Plain	4.47	9.06	13.97	19.58	23.62
PACP no Sec	6.27	9.15	14.92	23.28	31.71
PACP w/Sec	10.04	18.20	26.74	35.77	53.62
Jitter	1hop	2hop	3hop	4hop	5hop
Plain	0.56	1.26	1.25	2.21	1.45
PACP no Sec	0.47	0.50	0.50	1.16	1.26
PACP w/Sec	0.50	1.00	1.02	1.43	1.34

Table 5.2: Delay and Jitter obtained for several numbers of forwarding nodes. Obtained with traffic UDP 64kbits CBR. Values are in milliseconds.

VoIP flows are also very sensitive to jitter. Jitter measurements are defined in [schulzrinne03] and can be summed as the variation in delay. Queuing, delays accessing the medium and variation in the processing of packets are some of the sources of jitter. Typically values under 80ms can be corrected easily using a jitter buffer. Higher values will be more difficult to correct without adding too much delay. Because the network was only lightly loaded, and processing times are constant, jitter suffers no apparent variation maintaining values between 1 and 2 ms.

5.4 Overhead

The control traffic by the control protocol (overhead) has a negative impact on the bandwidth resources. The less overhead is introduced, the most bandwidth is available to transport user data.

PACP has several mechanisms adding control traffic to the network: registration, flow authorisation, marking and reporting. Registration is performed at node login and consists of two messages not having a big impact on overhead. Flow authorisation is only relevant with many nodes or flows. Even in such situations, it is only performed once per flow. The marking process differs from the previous mechanisms by constantly adding control data into the network. All data packets sent will be added a charging header with, at least 48 bytes. To this value, it should be added the size of the *MAC* which, for an ECDSA 163bits, results in 88 bytes. As the packet rate increases, so will the control data added inband to packets. Variations in the size of the packets will have no impact on the size of the header. Applications producing small packets

(VoIP) will be responsible for a big increase in the ratio between control and user data. Typically, considering that the size of VoIP packets vary between 64 and 128 bytes, PACP will add close to 100% of overhead. Reporting overhead consists of one packet which is sent by the last forwarding nodes toward the Access Router which replies with an acknowledge. This process is repeated at each n data packets, in which n typically is equal to 36 ($MTU = 1500$ and $MAC = 40bytes$). Table 5.3 depicts the overhead of each process considering several flows from Node5 to the Access Router.

Packet Rate	16		31		62	
Registration	0.11%		0.06%		0.04%	
Authorisation	0.09%		0.04%		0.03%	
Marking	9.37%	17.18%	9.37%	17.18%	9.36%	17.18%
Reporting	8.07%		8.07%		8.08%	
Total	17.64%	25.45%	17.56%	25.35%	17.51%	25.33%
Packet Size	256		512		1024	
Registration	0.2%		0.06%		0.05%	
Authorisation	0.15%		0.04%		0.04%	
Marking	18.75%	34.37%	9.37%	17.18%	4.68%	8.59%
Reporting	15.84%		8.07%		4.01%	
Total	34.94%	50.56%	17.56%	25.35%	8.78%	12.69%
	no MAC	w/ MAC	no MAC	w/ MAC	no MAC	w/ MAC

Table 5.3: Overhead of each process measured for different packet rates. First sub-table depicts results obtained with constant packet size (512bytes) and variable packet rate. Second sub-table packet size varied (256, 512, 1024) and packet rate was maintained ($31pkts^{-1}$).

Clearly, marking, together with reporting mechanisms, are the principal contributors to control overhead. Without usage of ECDSA signatures, overhead of both marking and reporting is similar. Including a MAC in all packets almost doubles the marking overhead. As expected, these mechanisms produce the same amount of control bytes as the packet size varies.

Bulk transfers with large packets will surely produce less relative overhead than real-time applications such as VoIP. Registration and authorisation are almost irrelevant in all situations, and as the number of packets transmitted increases, their impact decreases. Except for the marking process, others' are not influenced by the usage of ECDSA MAC as all control messages are obliged to contain a MAC .

Table 5.4 depicts the values of control overhead using a TCP application. First column represent a standard scenario where TCP flows are charged as other types of traffic. In this case, the marking reporting processes, as in the UDP tests, are the mechanisms adding most of the overhead. Other important factor is the usage of ECDSA to authenticate packets which adds between 3 to 5% of overhead. Other

Mode	With TCP/ACK		Without TCP/ACK	
Registration	0.02%		0.02%	
Authorisation	0.01%		0.01%	
Marking	5.7%	11.0%	3.2%	6.6%
Reporting	5.3%		5.8%	
Total	11.03%	16.33%	9.03%	12.43%
	no MAC	w/ MAC	no MAC	w/ MAC

Table 5.4: Overhead of each process measured for TCP. Considered both the cases where ACK packets are charged and where they are free. Also evaluated the usage of ECDSA *MAC* on charging headers (ECDSA 163bits)

aspect described is the possibility to forward TCP ACK packets, without payload, for free. These packets are sent into the network as a user is consuming a TCP service. However they carry no actual data. They are very small, consisting only of an IPv6 header followed by a TCP header with the ACK flag to 1. Charging such packets normally would almost double its size when the actual data was already charged. There could be the possibility to forward TCP ACK packets without any manipulation, in order to reduce the control overhead. The results show (please refer to table 5.4) this optimisation is able to reduce control overhead by almost 5%. The same method could be applied to other types of traffic, namely control flows of VoIP applications. Should be noticed this optimisation is only reasonable if TCP flows are unidirectional, i.e. TCP ACK packets have no payload. If they carry data, they should be charged normally.

5.5 Maximum throughput

The maximum throughput achievable by PACP will be affected by several factors. First, the overhead introduced in the network in-band in each packet will reduce the bandwidth available to application data. Then, reporting proofs, authorising flows and registration packets, will introduce additional control data, also reducing available bandwidth. Last, the delay introduced will limit the number of packets per second transmitted, resulting in a limitation of the throughput. First two factors reduce the available bandwidth directly by introducing additional information in the network. TCP because it creates two flows of packets will produce more overhead and will be penalised in terms of throughput. One flow carries actual data while the other is used to send the TCP acknowledgements.

The results obtained for control overhead are described on section 5.4. One important value is the limitation imposed by delay. Because nodes only have one processor, they must process packets sequentially. If packets arrive faster than they are processed, they will be queued and even dropped. PACP implementation queue size is limited to the internal *IP6Queue* queue size which, by default, is of 1024 packets. After the queue is full, *IP6Queue* will automatically start dropping packets.

Internal processing by PACP, using AODV, is demonstrated (see table 5.2) to, in average, be situated around 1.5ms. The best possible value is the sum of *IP6Queue* delay and the processing delay, which equals, $100\mu s$. This value is however difficult to maintain due to context switching and locking mechanisms. Without authenticating packets, this delay will be the service time of each packet to be served at each node. Considering authenticating packets, delays will be situated between 2 and 5ms, depending on node capabilities, otherwise will be between $100\mu s$ and 1.5ms. The maximum packet rates are calculated by $u = \frac{1}{serviceTime}$. Considering the presented delays and various packet sizes observable in 802.11 networks, table 5.5 depicts the maximum theoretical throughput values observable. Without usage of data authentication, the maximum throughput exceeds current 802.11 throughput and even surpasses 100Mbits. As the delay increases, the throughput decreases rapidly. Service times of 1.5, 2 and 5ms, set the throughput to values which start to be less than 802.11 medium bandwidth. Applications generating many small packets, such as VoIP, are able to be supported. However the total number of such applications simultaneously using the same nodes to forward packets will be limited.

Delay	100μs	1.5ms	2ms	5ms
Packet Rate	10000	666	500	250
Throughput (48 Bytes/pkt)	3,8Mb	256Kb	192Kb	96Kb
Throughput (512 Bytes/pkt)	41Mb	2.7Mb	2Mb	1Mb
Throughput (1500 Bytes/pkt)	120Mb	8Mb	6Mb	3Mb

Table 5.5: Maximum throughput calculated for various packet sizes and rates.

On multi-hop wireless networks, the major impact on throughput will not be the charging protocol but contention in accessing the wireless medium. As the number of hops increase, the maximum throughput will decrease [sargento06]. TCP and UDP will suffer differently, first because of TCPs' congestion avoidance mechanisms and because it creates more transport overhead than UDP. TCP headers are larger than UDP and it uses acknowledgement messages. The overhead introduced to packets will sum its effects with the access penalty much limiting the maximum throughput of the network.

Table 5.6 depicts the results obtained by sending TCP and UDP flows to the access router from the different nodes. UDP throughput is determined as the average data rate received while no packet loss occurs.

Hops	1		2		3		4		5	
Transport Proto	TCP	UDP								
Plain	1400	1571	588	681	378	427	246	279	216	246
PACP no ECDSA	1310	1526	549	643	355	407	235	267	204	237
PACP w/ECDSA	1260	1487	528	623	344	385	226	254	192	215

Table 5.6: Maximum throughput measured for various number of hops. Values are in kb/s

The maximum throughput is observed in directly connected nodes, achieving 1.6Mbps of available bandwidth. These values are close to the 2Mbps configured at the wireless medium. It is clear the limitation in throughput as the number of hops increase. At 5 hops, the maximum throughput is, in average, reduced to 16% of the original value and 11% of medium bandwidth. This is due to contention in accessing the medium, already studied in several other publications [sargento06] [holland02]. In all situations TCP proved to provide about 10% less throughput than UDP, which was expected. Adding PACP to the network also introduces some penalty to throughput. The observed penalty is 3.9% in UDP and 6.2% in TCP being directly related to overhead. These values are directly related to overhead as the throughput penalty is of the same order as the overhead. When PACP is adding and verifying *MAC* fields in data packets, overhead and delay will much increase, decreasing throughput. The degradation of using PACP with ECDSA verification (163bits keys) is, in average, 9.9% to TCP and 7.2% to UDP.

5.6 Charging process

The charging process of PACP cannot be easily evaluated on a testbed with the described resources. Analysing the proofs reported, during the tests performed in the previous sections, reveals perfect behaviour. Because nodes do not move and the topology is somewhat fixed, there is no error introduced and the charging rate is always 100%. The only situation producing any anomaly is when network load increases causing the charging rate to decrease due to packet losses. This issue is however much better analysed on a simulation environment where mobility can exist, load can be shaped and topology changed dynamically.

In order to validate the implementation, the ratio of charging proofs reported versus the number of packets received was measured for two different load situations. In both cases an UDP flow was sent from the Access Router to the last node of the topology (node5). Packet rate was constant to 32 packets per second, the packet size was either of 256 or 1024 bytes and flows lasted for 300s. PACP was adding and verifying *MAC* both on data and control packets and enough time was given in order for PACP to report all proofs. Such values result in flows with either 64kbits or 256kbits of bandwidth. As presented in table 5.6, 256kbits is more than the throughput available at with five hops. Moreover, the overhead of the charging protocol will further reduce the available bandwidth. The result of such constrains will be a divergence between the data received and the data reported due to packet loss, before the receiving node.

Table 5.7 represents the values obtained in the described test. When the network is not congested, PACP is able to charge every packet received by the destination node and there is no charging error. As the bitrate increases, the link becomes more congested and some packets start to be dropped. According to the values obtained in 5.6, packets will be lost both before the last forwarding node and before the destination node. On both cases, the network has no available bandwidth to route the requested 256kbits. The Last Forwarding Node will thus report less packets than sent, but more important, it will report more packets than received. The total error corresponds to the percentage of packets lost in the last hop.

Packet Size (bytes)	Bytes Received	Packets Received	Bytes Reported	Packets Reported	Charging Rate
256	2457856	9601	2457856	9601	100%
1024	8780800	8574	9363456	9144	107%

Table 5.7: Comparison between the data received and reported in a 6 node string scenario using UDP flows.

One particularity about the encoding method used by PACP is the higher complexity to both encode and decode the route. This issue is dependent on the hardware and can be tested using the real implementation. Testing the performance in simulation environment is somewhat difficult because processing time is not real. Moreover, the load of the machine is being altered by the simulator. In order to test the performance of the charging process, both the encoding and the decoding processes were benchmarked. The benchmark program was divided in two blocks: encoding and decoding.

The encoding block generates 255 random host identities (IPv6 addresses) generated and stores them in a array; then a value r was iterated between 1 and the

maximum route length possible (255); for each route length, $r + 1$ proofs were generated and forwarded across r of the generated hosts. The resulting values are the same as if the packet was routed in the real environment with the operations used. In order to better measure the delay between operations, the forward process was repeated t times (typically $t = 10000$). The total duration of these operations was measured and used to calculate the delay of each encoding.

The decoding block receives the proofs previously encoded and performs the decoding as the Charging Manager would perform. As with the forwarding process, proofs are decoded a few thousand of times in order to better measure the delay. Measuring delays in a multitasking environment is not easy, particularly the time resolution is only of 1 or 2 milliseconds using standard libraries.

Tests were performed on an Athlon XP-M 1800+ laptop with power management software disabled and no other service running. Figure 5.10 depicts the results obtained for decoding 10000 proofs, recovering the encoded addresses. The encoding process is not depicted because its impact is negligible. In reality, encoding is implemented as only 16 multiplications and additions consuming only a few instructions per packet forwarded. The decoding process is more complex than the encoding, however, as shown, it does not require significant amount of processing. Even if the route length is of 250 hops, when decoding 250 proofs and identifying the forwarding nodes, only $170\mu s$ are required. These results make possible real-time decoding of the proofs and identification of the forwarding nodes, even on low resource equipments. Particularly, considering that the average route length in MANETs is considerable shorter, the decoding process will be comparable to calculating a simple digest such as MD5 or SHA-1.

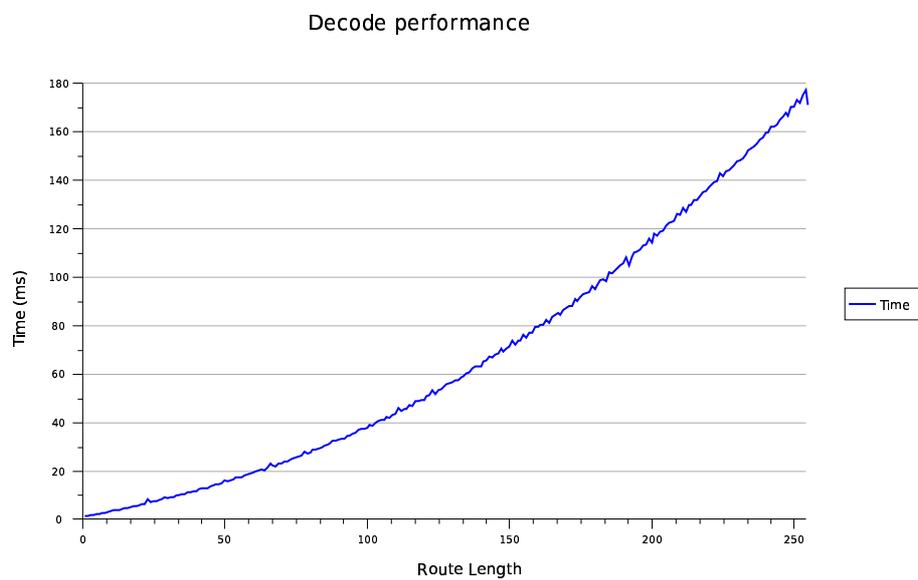


Figure 5.10: Time consumed when decoding proofs 10000 times for various route lengths.

Chapter 6

Conclusions

Self-organised ad-hoc networks are of much importance on future networks. The complexity of the systems requires higher coordination much beyond the possibilities of centralised networks. Ad-hoc networks are of vital importance to the new ubiquitous networks where network access is available anywhere. In such networks equipments interact with others in the neighbourhood and cooperate in order to provide access and share resources. Connectivity and routing are already thoroughly addressed with many proposals already at a stable state. Using the routing proposals existing, many other, higher layer solutions, are already developed covering other aspects like auto-configurations, service discovery, QoS and security.

Enforcing a good behaviour of nodes is fundamental to the correct function of the ad-hoc network. Without proper behaviour, application of self-organised networks is severely compromised. In heterogeneous scenarios, where the resources nodes and requisites of each user are very different, conflict of interest is provable and active enforcement is easily required. Many solutions are already able to provide proper monitoring of node behaviour, acting accordingly when detecting suspicious events. Other solutions enforce proper utilisation by means of credit exchange. Either performed directly between nodes or using a central bank, credit based solutions make use of common incentives (credit) to both control the usage of the network and promote forwarding. Solutions considering the usage of a central bank have the advantage of making possible to integrated ad-hoc network on, commercially driven infrastructure networks with wireless hotspots.

This thesis developed two protocols for secure charging in ad-hoc extended hotspot scenarios. They are able of providing efficient yet secure charging mechanisms in op-

erator driven scenarios. Interface with a central credit clearance system is envisioned. Also, integration with user profiles specifying either QoS or service based restrictions is supported. Forwarding nodes are rewarded for the packets they forward. The possibility of crediting or charging users based on the packet size, service port QoS class or time is provided. This makes it possible to create products, providing ad-hoc network access, with much granularity.

The solutions presented and analysed on this thesis proved to be capable of efficiently integrating ad-hoc networks in infrastructure based environments. The solutions developed also proved to improve the current state-of-the-art with better management capabilities, performance and reduced overhead. This was proved using an analytical analysis, extensive simulation and, in the case of PACP, a real world implementation. PACP was adopted as the charging and rewarding solution in the Daidalos project, further enhanced the requisites of the solution and its implementation. PACP was further tested in the Daidalos testbed.

Overall, results show that the different proposals do not present equal performance. In all results, UDP flows present more variation than TCP flows. The relative packet delivery ratio is influenced by mobility, network load and the routing protocol. All proposals introduce some penalty to throughput which, in the scenarios simulated, is never less than 11%. Using AODV, PACP presented the best (and more stable) results, while SCP the worst. When using DSR, because of the optimizations present in SCP and SACP, they provide slightly better throughput than PACP. One issue to consider is that increasing the route length or mobility has a negative effect on SACP and SCP performance while PACP is more resilient.

Overhead is much affected by the parameters analysed, in particular the routing protocol and node movement velocity. Network load seems to only marginally affect the resulting charging overhead. Mobility affects overhead in two different manners: more control messages and reduced session end-to-end length. SACP needs to send more *RouteUpdate* messages as mobility increases while reducing the session length will require less nodes to be reported. The second effect affects all proposals by decreasing the produced overhead and is dominant over the first. In terms of overhead and using AODV, PACP is the most efficient proposal followed by SACP. SCP is the proposal presenting higher overhead values when using AODV. DSR apparently reduces the charging overhead of both SACP and SCP. However, the information previously added by these proposals is still added to packets. As it is added by DSR and not by the

charging protocol, it is not accounted as charging overhead but routing overhead.

One of the primary functionalities of the protocols evaluated is to charge network traffic. Charging can be done to the sender or to the receiver and both possibilities are supported by these proposals. Because proofs are gathered at the last forwarding node, the efficiency is different when charging the sender or the receiver. The charging error is always higher when charging the sender and much aggravated when using UDP flows. Using such combination, as the network load increases, only a fraction of the packets sent will be charged. With the same load and mobility, the receiver is accurately charged. Other aspect affecting the charging error is mobility. As mobility increases, the charging rate of SCP and PACP decreases. In SCP this is due to the need of proof acknowledgement. In PACP this is the result of slightly lower number of proofs reported in each report message.

All packets charged are also able to be rewarded, and SCP is actually able to always preform both. In some situations PACP and SACP are unable to reward users accurately. PACP is sometimes unable to identify the forwarding nodes, while SACP can credit a small amount of proofs to the wrong users. Results show the rewarding error to be always less than 3% (SACP). The major contributors to the increase in the rewarding error are mobility and network load. Increasing one of the variables will result in an increase in the rewarding error. Overall, PACP presented lower error rates than SACP. Also, it is more imune to load and mobility.

Regarding the implementation, *IP6Queue* proved to be an usable method of intercepting packets, with the possibility of change and block. However, because of the exclusive nature of the interface, a closer integration with the Linux kernel would be more suited. The implementation is well designed providing the mechanisms described along this thesis. The performance provided makes possible real-time communications such as VoIP. Using ECDSA to secure the payload of data packets proved to add much delay, limiting the throughput and the maximum number of forwarding nodes.

Future work may focus on the development of even more efficient reporting methods. This may be achieved either by following the work presented or by novel interaction schemes. ECDSA is secure and lightweight in terms of bandwidth, however, without any hardware acceleration, it adds much delay to nodes. Novel authentication methods could be used resulting in increased performance.

Bibliography

Publications

- [80211] “*IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*”, Nov. 1997. P802.11.
- [ahn02] Gahng-Seop Ahn, Andrew T. Campbell, Andras Veres, and Li-Hsiang Sun, “*Supporting Service Differentiation for Real-Time and Best-Effort Traffic in Stateless Wireless Ad Hoc Networks (SWAN)*” In IEEE Transactions on Mobile Computing, vol. 1, no. 3, pp. 192-207, 2002
- [badis03] H. Badis and K. Al Agha, “*QoS for ad hoc networking based on multiple-metric: Bandwidth and delay*”, IFIP MWCN’03: Mobile and Wireless Communications Networks, 2003
- [bansal03] S. Bansal and M. Baker, “*Observation-Based Cooperation Enforcement in Ad hoc Networks*”, Research Report cs.NI/0307012, Stanford University, 2003.
- [beldingroyer02] Elizabeth M. Belding-Royer, “*Report on the AODV Interop*” UCSB Tech Report 2002-18, 2002
- [bellur99] B. Bellur and R. G. Ogier, “*A reliable, efficient topology broadcast protocol for dynamic networks*”. Proceedings of IEEE INFOCOM 1999.
- [bhatia03] Hsu J, Bhatia S, Takai M, Bagrodia R and Acriche MJ, “*Performance of mobile ad hoc networking routing protocols in realistic scenarios*”, Military Communications Conference, 2003. MILCOM 2003. Vol. 2, pp. 1268-1273.
- [blazevic01] Blazevic, L.; Buttyan, L.; Capkun, S.; Giordano, S.; Hubaux, J.-P.; Le Boudec, J.-Y., “*Self organization in mobile ad hoc networks: the approach of Terminodes*”, IEEE Communications Magazine, vol. 39, Issue 6, pp. 166-174, 2001
- [buchegger02] S. Buchegger and J.-Y. Le Boudec. “*Performance Analysis of the CONFIDANT protocol: Cooperation of nodes - Fairness In Distributed Ad-Hoc Networks*”. In Proceedings of IEEE/ACM Symposium on Mobile Ad-Hoc Networking and Computing (MobiHOC), Lausanne, CH, 2002.

- [buechegger03] Sonja Buchegger, Jean-Yves Le Boudec, “*Coping with False Accusations in Misbehavior Reputation Systems for Mobile Ad-hoc Networks*”, EPFL Technical Report IC/2003/31
- [calçada05] T. Calçada, and Manuel Ricardo, “*Extending the Coverage of a 4G Telecom Network using Hybrid Ad-hoc Networks: a Case Study*”, Proceedings of the Fourth Annual Mediterranean Ad Hoc Networking Workshop, vol. 197/2006, pp. 367-376, ISBN: 0-387-31171-8, 2005
- [chen02a] K. Chen, S. H. Shah, and K. Nahrstedt, “*Cross Layer Design for Data Accessibility in Mobile Ad Hoc Networks*”, Journal on Wireless Communications, vol. 21, pp. 49-75, 2002
- [chen04b] Chen, K., Nahrstedt, K., “*iPass: an incentive compatible auction scheme to enable packet forwarding service in MANET*”, Proceedings of the 24th International Conference on Distributed Computing Systems, 2004. pp. 534-542, ISBN: 0-7695-2086-3, 2004
- [chiu89] D-M. Chiu and R. Jain, “*Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks*” Computer Networks and ISDN Systems, vol. 17, pp. 11-14, 1989.
- [chown03] T. Chown, T. Ferarrii, S. Leinen, N. Simar, S. Venas, “*Less-than-Best-Effort: Application Scenarios and Experimental Results*”, Proceedings of the Second International Workshop, QoS-IP 2003, pp. 131, ISBN 3-540-00604-4, 2003
- [clausen01] T. Clausen, P. Jacquet, A. Laouiti, P. Muhlethaler, A. Qayyum and L. Viennot. “*Optimized Link State Routing Protocol*”, Proceedings of IEEE International Multitopic Conference, INMIC2001, 2001.
- [crisóstomo05] Sérgio Crisóstomo, Susana Sargento, Marek Natkaniec, Norbert Vicari, “*A QoS Architecture Integrating Mobile Ad-Hoc and Infrastructure Networks*”, Proceedings of 3rd ACS/IEEE International Conference on Computer Systems and Applications (AICCSA-05), 2005.
- [dean02] D. Dean et al, “*An algebraic approach to IP traceback*”, ACM Transactions on Inf. and System Security, v.5 n.2, 2002.
- [domingo04] M. C. Domingo and D. Remondo, “*A Cooperation Model between Ad Hoc Networks and Fixed Networks for Service Differentiation*”, Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN’04), pp. 692-693, 2004
- [dressler06] F. Dressler, “*Self-Organization in Ad Hoc Networks: Overview and Classification*”, Ad-Hoc Networks 2006
- [ecuyer99] L’Ecuyer, P., “*Good parameters and implementations for combined multiple recursive random number generators*”, Operations Research 47 (1): pp. 159-164, 1999

- [fang04] Zuyuan Fang, Brahim Bensaou, “*Fair bandwidth sharing algorithms based on game theory frameworks for wireless ad-hoc networks*”, INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 2, pp. 1284-1295, ISBN: 0-7803-8355-9, 2004
- [frei98] A. Fei, G. Pei, R. Liu, and L. Zhang, “*Measurements on delay and hop-count of the Internet*”, In Proceedings of Globecom’98, 1998
- [girão04] João Girão, João Barraca, Bernd Lamparter, Dirk Westhoff, and Rui Aguiar. “*Qos-differentiated secure charging in ad-hoc environments*”. In 11th International Conference on Telecommunications ICT2004, 2004.
- [haas97] Zygmunt J. Haas, “*A New Routing Protocol For The Reconfigurable Wireless Networks*”, Proceedings of 6th IEEE International Conference on Universal Personal Communications, IEEE ICUPC’97, vol. 2, pp. 562-566, 1997
- [hafslund04] Hafslund A., TÄyñnesen A., Rotvik J. B., Andersson J., Kure Äÿ., “*Secure Extension to the OLSR protocol*”, OLSR Interop Workshop, 2004
- [henrik02] Henrik Lundgren, Erik Nordstr”om and Christian Tschudin, “*The Gray Zone Problem in IEEE 802.11b based Ad hoc Networks*”, ACM SIGMOBILE Mobile Computing and Communications Review, vol. 6, no. 3, pp. 104-105, 2002
- [holland02] Gavin Holland, Nitin Vaidya, “*Analysis of TCP Performance over Mobile Ad Hoc Networks*”, Wireless Networks, vol. 8, no. 2 - 3, pp. 275-288, ISSN: 1022-0038 , 2002
- [huang04] Elgan Huang, Jon Crowcroft and Ian Wassell, “*Rethinking incentives for mobile ad hoc networks*”, PINS ’04: Proceedings of the ACM SIGCOMM workshop on Practice and theory of incentives in networked systems, pp. 191-196, ACM Press, ISBN: 1-58113-942-9, 2004
- [johnson94] D. B. Johnson, “*Routing in Ad Hoc Networks of Mobile Hosts*”, in Proceeding of ACM Mobicom 1994
- [koblitz87] N. Koblitz, “*Elliptic curve cryptosystems*”, in Mathematics of Computation 48, pp. 203-209, 1987
- [lamparter03] B. Lamparter et al., “*Charging Support for Ad Hoc Stub Networks*”. Elsevier Journal of Computer Communication, Special Issue on Internet Pricing and Charging: Algorithms, Technology and Applications, 2003.
- [lee98] Seoung-Bum Lee and Andrew T. Campbell, “*INSIGNIA: In-band signaling support for QoS in mobile ad hoc networks*”, Proceedings of 5th International Workshop on Mobile Multimedia Communications (MoMuC’98), 1998
- [li02] J. Li and Z. Haas and M. Sheng, “*Capacity Evaluation of Multi-Channel Multi-Hop Ad Hoc Networks*”, IEEE International Conference on Personal Wireless Communications, pp. 211-214, ISBN: 0-7803-7569-6, 2002

- [marina02] M. Marina and S. Das, “*Ad hoc on-demand multipath distance vector routing*”, ACM SIGMOBILE Mobile Computing and Communications Review, vol. 6, no. 3, pp. 92-93, ISSN:1559-1662, 2002
- [michiardi00] Pietro Michiardi, Refik Molva, “*Core: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks*”, Proceedings of the IFIP TC6/TC11 Sixth Joint Working Conference on Communications and Multimedia Security: Advanced Communications and Multimedia Security, pp 107-121, ISBN:1-4020-7206-6, 2002
- [paul02] Krishna Paul and Dirk Westhoff, “*Context aware inferencing to rate a selfish node in dsr based ad-hoc networks*”, Proceedings of the IEEE Globecom Conference, 2002
- [perkins94] C. Perkins and P. Bhagwat, “*Highly Dynamic Destination Sequenced Distance Vector Routing (DSDV) for Mobile Computers*”, in Proceedings of ACM SIGCOMM 1994.
- [press92] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. “*Numerical Recipes in FORTRAN: The Art of Scientific Computing*”, Cambridge University Press, 1992.
- [salem03] N. Salem et al, “*A charging and rewarding scheme for packet forwarding in multi-hop cellular networks*”, 4th ACM Int. Symp. Mobile ad hoc networking & computing, 2003
- [salem06] Naouel Ben Salem, Levente Buttyán, Jean-Pierre Hubaux and Markus Jakobsson, “*Node Cooperation in Hybrid Ad Hoc Networks*”, IEEE Transactions on Mobile Computing, vol. 5, no. 4, pp. 365-376, 2006
- [sargento05] Susana Sargento, Tania Calçada, João Paulo Barraca, Sergio Crisostomo, João Girão, Marek Natkaniec, Norbert Vicari, Francisco Cuesta, and Manuel Ricardo, “*Mobile ad-hoc networks integration in the daidalos architecture*”, IST Mobile and Wireless Communications Summit, 2005
- [sargento06] Susana Sargento, João Paulo Barraca, Miguel Almeida, Rafael Sarrô and Rui Aguiar, “*Experimental Evaluation of an Integrated Ad-hoc Network*”, 15 th IST Mobile and Wireless Communications Summit, 2006
- [weyland04] Attila Weyland and Torsten Braun, “*CASHnet - Cooperation and Accounting Strategy for Hybrid Networks*”, in The 13th IEEE Workshop on Local and Metropolitan Area Networks, 2004. LANMAN 2004, pp. 193-198, ISBN: 0-7803-8551-9, 2004
- [xiao00] H. Xiao, W. K. G. Seah, A. Lo and K. C. Chua, “*A Flexible Quality of Service Model for Mobile Ad-hoc Networks*”, Proceedings of VTC-2000 Spring, 2000
- [zhong03] S. Zhong et al., “*Sprite: A Simple, Cheat-Proof, Credit-Based System for Mobile Ad Hoc Networks*”, IEEE INFOCOM’03, San Francisco, 2003.

IETF Drafts

- [chakeres06] I. Chakeres, C. Perkins, “*Dynamic MANET On-demand (DYMO) Routing*”, draft-ietf-manet-dymo-05.txt, Internet Draft
- [clausen06] T. Clausen and C. Dearlove, “*The Optimized Link-State Routing Protocol version 2*”, draft-ietf-manet-olsrv2-01.txt, Internet Draft
- [haas02] Zygmunt J. Haas, Marc R. Pearlman, Prince Samar, “*The Zone Routing Protocol (ZRP) for Ad Hoc Networks*”, draft-ietf-manet-zone-zrp-04.txt, Internet Draft
- [jelger04] C. Jelger, T. Noel and A. Frey, “*Gateway and address autoconfiguration for IPv6 adhoc networks*”, draft-jelger-manet-gateway-autoconf-v6-03.txt, Internet Draft
- [jeong06] Jaehoon Paul Jeong, Jungsoo Park, Hyoungjun Kim, Hongjong Jeong and Dongkyun Kim, “*Ad Hoc IP Address Autoconfiguration*”, draft-jeong-adhoc-ip-addr-autoconf-06.txt, Internet Draft.
- [johnson04] David B. Johnson, David A. Maltz, Yih-Chun, “*The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)*”, draft-ietf-manet-dsr-10.txt, Internet Draft
- [moskowitz04] R. Moskowitz and P. Nikander, “*Host Identity Protocol Architecture*”, draft-moskowitz-hip-arch-06.txt, Internet Draft
- [perkins97] C. Perkins, “*Mobile Ad Hoc Networking Terminology*”, draft-ietf-manet-term-00.txt, Internet Draft
- [perkins03a] Perkins, C. and E. Belding-Royer, “*Quality of Service for Ad hoc On-Demand Distance Vector Routing*”, draft-perkins-manet-aodvqos-02.txt, Internet Draft
- [royer00] Elizabeth M. Royer, Charles E. Perkins, “*Multicast Ad hoc On-Demand Distance Vector (MAODV) Routing*”, draft-ietf-manet-maodv-00.txt, Internet Draft, 2000
- [sethom06] Kaouthar Sethom, Hossam Afifi, Frank Y. Li, Andreas Hafslund, “*Gateway Selection in Multi-homed Ad Hoc Networks*”, draft-sethom-adhoc-gateway-selection-01.txt, Internet Draft
- [wakikawa06] Ryuji Wakikawa, Jari T. Malinen, Charles E. Perkins, Anders Nilsson and Antti J. Tuominen, “*Global connectivity for IPv6 Mobile Ad Hoc Networks*”, draft-wakikawa-manet-globalv6-05.txt, Internet Draft
- [zapata05] Manel Guerrero Zapata, “*Secure Ad hoc On-Demand Distance Vector (SAODV) Routing*”, draft-guerrero-manet-saodv-05.txt, Internet Draft, 2005

RFC and Standards

- [ansix9.62] ANSI. X9.62 - “*Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*”, 1999
- [aura05] T. Aura, ‘ *RFC 3972 - Cryptographically Generated Addresses (CGA)*’, IETF RFC, March 2005
- [braden97] R. Braden, Ed., S. Berson, S. Herzog, S. Jamin, “*RFC 2205 - Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification*”, IETF RFC, September 1997
- [calhoun03] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, J. Arkko, “*RFC 3588 - Diameter Base Protocol*”, IETF RFC, September 2003
- [clausen03] T. Clausen, Ed., P. Jacquet, Ed., “*RFC 3626 - Optimized Link State Routing Protocol (OLSR)*”, IETF RFC, October 2003
- [deering98] S. Deering, R. Hinden, “*RFC 2460 - Internet Protocol Version 6 (IPv6) Specification*”, IETF RFC, December 1998
- [droms03] R. Droms, Ed., J. Bound, B. Volz, T. Lemon, C. Perkins, M. Carney, “*RFC 3315 - Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*”, IETF RFC, July 2003
- [eastlake01] D. Eastlake, 3rd, P. Jones, emph“*RFC 3174 - US Secure Hash Algorithm 1 (SHA1)*”, IETF RFC, September 2001
- [hedrick88] C. Hedrick, “*RFC 1058 - Routing Information Protocol*”, IETF RFC, June 1988
- [itug.107] ITU-T Recommendation G.107, “*The Emodel, a computational model for use in transmission planning*”, 1998
- [ituz.100] ITU-T Z.100, “*Specification and description language*”,
- [mccann05] P. McCann, “*RFC 4260 - Mobile IPv6 Fast Handovers for 802.11 Networks*”, IETF RFC, November 2005
- [mills92] David Mills, “*RFC 1305 - Network Time Protocol (Version 3). Specification, Implementation and Analysis*”, IETF RFC, March 1992
- [ogier04] R. Ogier, F. Templin, M. Lewis, “*RFC 3684 - Topology Dissemination Based on Reverse-Path Forwarding (TBRPF)*”, IETF RFC, February 2004
- [perkins03b] C. Perkins, E. Belding-Royer, S. Das, “*RFC 3561 - Ad hoc On-Demand Distance Vector (AODV) Routing*”, IETF RFC, July 2003
- [rigney00] C. Rigney, S. Willens, A. Rubens, W. Simpson, “*RFC 2865 - Remote Authentication Dial In User Service (RADIUS)*”, IETF RFC, June 2000

- [rivest92] R. Rivest, “*RFC 1321 - The MD5 Message-Digest Algorithm*”, IETF RFC, April 1992
- [schulzrinne03] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, “*RFC 3550 - RTP: A Transport Protocol for Real-Time Applications*”, IETF RFC, July 2003
- [sermersheim06] J. Sermersheim, Ed., “*RFC 4511 - Lightweight Directory Access Protocol (LDAP): The Protocol*”, IETF RFC, June 2006
- [thomson98] S. Thomson, T. Narten, “*RFC 2462 - IPv6 Stateless Address Autoconfiguration*”, IETF RFC, December 1998

Thesis

- [ingo03] Ingo Riedel, “*Security in Ad-hoc Networks: Protocols and Elliptic Curve Cryptography on an embedded Platform*”. Diploma Thesis, Ruhr-Universität Bochum, 2003.

Web sites and Applications

- [aodvweb] “*Ad hoc On-demand Distance Vector Routing*”, <http://moment.cs.ucsb.edu/AODV/aodv.html>
- [aodvuu] “*AODV-UU CoReSoftware* ”, <http://core.it.uu.se/core/index.php/AODV-UU>
- [autoconf] “*Ad-Hoc Network Autoconfiguration (autoconf) charter*”, <http://www.ietf.org/html.charters/autoconf-charter.html>
- [daidalos] “*FP6 IST Integrated Project DAIDALOS*”, <http://www.ist-daidalos.org>
- [gcc] GCC, the GNU Compiler Collection, <http://gcc.gnu.org/>, as in July 2006.
- [iperf] “*IPERF Homepage*“, <http://dast.nlanr.net/Projects/Iperf/>
- [itav] “*IT: Instituto de Telecomunições - Aveiro premises*”, <http://www.av.it.pt>
- [labplot] “*LabPlot Homepage*”, <http://labplot.sourceforge.net/>
- [manet] “*Mobile Ad-hoc Networks (manet) Charter*”, <http://www.ietf.org/html.charters/manet-charter.html>
- [mackill] “*MacKill*”, <http://core.it.uu.se/adhoc/ImplementationPortal>
- [mgen] “*MGEN - The Multi-Generator Toolset*”, <http://pf.itd.nrl.navy.mil/mgen/>
- [mysql] “*MySQL AB: The world’s most popular open source database*“, <http://www.mysql.org>

- [ns2] “*The Network Simulator NS2*”, <http://www.isi.edu/nsnam/ns>, as in July 2006.
- [postgresql] “*PostgreSQL: The world’s most popular open source database*”, <http://www.postgresql.org>
- [tcpdump] “*TCPDUMP*”, <http://www.tcpdump.org>
- [trpr] “*TRace Plot Real-time*”, <http://pf.itd.nrl.navy.mil/protocols/trpr.html>
- [wireshark] “*Wireshark: The World’s Most Popular Network Protocol Analyzer*“, <http://www.wireshark.org>