Linking Ad Hoc Charging Schemes to AAAC Architectures

Joao Girao[1], Bernd Lamparter[1], Dirk Westhoff[1],
Rui L. Aguiar[2,3], and Joao P. Barraca[3]

[1] NEC Europe Ltd., Heidelberg, Germany
{joao.girao, bernd.lamparter, dirk.westhoff}@ccrle.nec.de
[2] University of Aveiro, Portugal
ruilaa@det.ua.pt
[3] Institute of Telecommunications, Aveiro, Portugal
jpbarraca@av.it.pt

Abstract. The current state of today's networks allows us to take one step further in merging the research community's work with every day's life. Wireless ad hoc networks are already well developed for specific scenarios. This work shows how to build the link between the wired network and a wireless ad hoc infrastructure, in particular routing and AAAC aspects. Such integration might lead, for example, to a better spacial and resource distributed hotspot solution.

We provide the basis for inter-operation of AAAC[1] protocols known for the fixed network, with the accounting protocol that performs the accounting and charging functions in the ad hoc network.

This paper further describes the implementation of the Secured Charging Protocol as an instantiation of a charging protocol for ad hoc networks and the features which were added to improve the interface to an external accounting system. It covers the interaction with the MANET routing protocol and how to deal with routes to or from outside the ad hoc cloud.

1 Introduction

The Internet is today's best example of the fixed networks achievement over the years. These networks have proved to be robust and to have a very high success rate. Nevertheless, technology evolves and the use of wireless devices has become as common as that of their wired counterparts. Hybrid systems are now common in homes, offices and leisure environments. Even widespread wireless networks are no longer a tendency, but a reality.

The research community has long studied these matters and as a result, ad-hoc networks have been developed, and are been looked as a communication paradigm for the future. However, we should not neglect the years of experience

[1] Authentication, Authorization, Accounting and Charging.

C. Castelluccia et al. (Eds.): ESAS 2004, LNCS 3313, pp. 66–81, 2005.

on routing, security, QoS and network management done, even if founded on wired networks. We must use such value and adapt it into this new arena.

Hotspots have appeared as points of access to the Internet via a wireless medium in places where conventional wiring would be inconvenient or even impossible. This concept of network access seems to be able to benefit from the usage of ad hoc like infrastructures in order to maximize network connectivity.

Unfortunately, ubiquitous wireless multi-hop ad hoc networks are inherently troublesome due to privacy and security issues. Security and routing is also linked to the fact that some nodes may not wish to cooperatively forward for other node's traffic. This is why certain mechanisms have been engineered to avoid the selfish nodes problem (the case when the ad hoc node does not forward packets which are unrelated to it).

There are two main approaches to the problem: The first is to identify misbehaving nodes and exclude them from the community [1], [2]; the second, with which this paper deals, is to lead nodes to cooperation through the use of some incentive, such as proposed in Nuglets [3], Sprite [4] and SCP [5].

SCP is based in the notion that all nodes must gain when a packet is received correctly at its final destination. Nodes that send and receive data benefit from the other's cooperation, and have to pay for this service. Nodes that forward data receive money as an incentive for helping others. Since this is a highly flexible communication paradigm where a sending node can forward or vice-versa; nodes can use the money they earn by forwarding foreign data, to send and receive their own.

This is the business model adjacent to SCP, which can be used in the hotspot scenario by extending node connectivity to an ad hoc network, and provide Internet connection to a wider range of clients. Within an ad hoc cloud, nodes are persuaded to forward each other's traffic with the incentive earning money for it. On the other hand, nodes that do want to use the network resources will not mind paying accordingly for the amount of resources consumed. They know that SCP helps improving co-operation and increases the end-to-end reliability of the network.

This paper provides a realistic solution to the interaction between ad hoc and fixed networks in what concerns authentication, authorization, accounting and charging. Its main contributions include a possible way in which to deploy an ad hoc network that maintains services and applications from the fixed networks.

We present a practical, yet powerful approach to fasten the process of integration between the studied ad hoc mechanisms and current network architectures.

In the next section we introduce the Secured Charging Protocol. We continue with the implementation description of the protocol in Section 3 and 4. Section 5 introduces the integration with the routing protocol and the mechanisms necessary to extend SCP to the Internet. In the following two sections we insert SCP in the infrastructure. We further describe our testbed and results on the implementation in Section 8. We finish the paper with a discussion on future work in this area and our conclusions.

2 The Secured Charging Protocol

As we have previously argued, cooperative networks are a strong candidate to achieve realistic connectivity in a potentially selfish environment. Obviously, selfishness is particularly felt in civilian ad hoc networks with power restricted and battery constrained devices and networks open to everyone. We propose to overcome this problem and increase cooperation by using SCP. We describe the applied primitives of SCP, the assumed architecture and give an overview on the protocol.

2.1 Protocol Architecture

SCP assumes an architecture consisting of several different components. They are located in the ad hoc network and in the fixed network:

- The AAAC architecture in the fixed network is composed by the AAA Foreign Server (AAAF) and the AAA Home Server (AAAH). The AAAF belongs to the same domain as the Access Router (AR) of the ad hoc cloud. The AAAH is able to identify the node as belonging to his administrative domain. If a mobile node has not left its administrative domain, both aforemntioned AAAC components are physically the same.
- The AR is the first point of contact a node has with the network. Initially, it solely allows nodes to access the AAAC system so they can register and provide accounting information. The AR performs the role of a translator between the SCP and other AAAC protocols specific to the fixed network.
- A node of the ad hoc cloud may act in different roles depending on its place in the routing path:
  – Should the node be the first in the path, it is the sender (we will also use the notation of Mobile Node (MN) for this node). Its objective is to transmit packets in a multihop fashion across the (wireless) network even if this means he has to pay for this service.
  – If the node is receiving the packets as the final destination, it is the receiver (We will refer to this node as the Correspondent Node (CN)). For the service of receiving packets from the MN, this node is also willing to take part in the payment of the bill.
  – Any other node in the path that forwards traffic is a Forwarding Node (FN). These nodes are willing to forward packets for other nodes since they subsequently receive incentives in the form of money.
  – A special case of a FN is the Last FN (LFN), the immediate predecessor of the CN. This node occupies the last position in the routing chain before the CN. More than the functions of a normal FN, a node in this role is also responsible for transmitting the accounting information to the AR (when a connection to the AR is available) and before a charging period of the Internet Service Provider (ISP) has expired.

Note that each node may act in different roles. It may play the role of MN, CN, FN or LFN. In fact, it might even be in several roles at the same time when multiple communication sessions are active.

2.2 Protocol Overview

SCP operation can be subdivided into three different phases:

1. Registration Phase
2. Forwarding Phase
3. Charging Phase

In the one-time Registration Phase, a node will seek initial authentication within the domain the ad hoc network belongs to [6]. To do so, it may or may not have to contact its home AAAC server, e.g. via a challenge response protocol. The Security Association (SA) between different AAAC servers (AAAF and AAAH), between the AR and the AAAF and between the node and the AAAH are considered to be static. Thus, SCP does not create new SAs but simply establishes the previously known connection between the MN and the AAAH.

Registration occurs only when the node arrives to a new network where it is not authenticated or when its authentication needs to be refreshed due to an expired certificate.

Should the AR be able to prove the node's identity, it will respond with a valid certificate the node can use in his domain and a shared secret. It will also provide charging information, namely the factors by which the node will pay or be rewarded in an Access Response message.

If the AR has no confirmation on the node's identity, it will respond with an Access Response containing an error message denoting the incorrect parameter or the cause for the error.

Once the Registration is complete, a node is allowed to send, receive and forward data.

The Forwarding phase is best described as the act in which a node decides to pass a data packet to the next node in the path. This builds up to the sending and receiving of data from source to final destination. During this phase the sending node will sign the packet. Intermediate nodes will then verify the signature of the sender and include their marking on the packet by means of a hash chain. They also add themselves, should this be required[2], to the route contained in that same packet.

This hash chain is extended by hashing the previous value and a shared secret between the node and the AR. A random start value, also transmitted with the message, assures the freshness of this data.

If a node cannot verify the signature contained in the packet, the packet must be dropped since the node now is assured it will not receive any incentive for that particular transaction.

We implement the forwarding information needed by SCP as an extension header to the IPv6 protocol over which it travels. All information needed for accounting is in the packet itself.

A periodic exchange of two confirmation messages that we now describe allows for the synchronization between the LFN and the CN in what concerns the

[2] Certain routing protocols, such as DSR [7], already provide this information.

amount of data actually received. This exchange occurs only seldom and can be triggered after a certain number of packets, bytes or an amount of time. The messages used are:

Confirmation Request. This packet is sent by the LFN to the CN to ascertain the amount of data the node has received correctly. This packet is a contention mechanism not to overcharge the end-to-end pair. Since digital signatures are applied here, the information contained in this packet is non-repudiative.

Confirmation Response. As a reaction to the former packet, this message contains information on the amount of data the node received for that session.

Finally, the Charging phase of the protocol corresponds to the sporadic exchange of two packets, which takes place depending on the availability of the Access Router (this exchange of packets always occurs after the Forwarding Phase and before the end of the ISP's charging phase):

Verification Request. The LFN now sends a Verification Request message to the AR to perform the accounting function. The relevant information this message contains is the session, the route, a list of hash chains corresponding to each of the packets, the number of bytes accounted for and the number of bytes acknowledged by the CN[3].

Verification Response. The AR confirms the received accounting information.

All the packets in SCP, with the exception of the Access Request, are signed by the sender of the message, independently of whether it is data or the signaling flow. This, together with a Certifying Authority (CA) that makes sure the keys being used by the participants are trustworthy, provides a per bundle or per packet authentication.

2.3 Cryptographic Primitives

SCP makes use of two basic cryptographic primitives:

- Hash functions
- Digital Signatures

The choice of both primitives relates to the envisioned security level of SCP. We opted for a 16 byte MD5 as it provides sufficient security for the expected short lifetime of the values. A random number provides freshness to avoid replay attacks. MD5 is preimage and second preimage resistant, as well as collision resistant. These features combined with the short byte length distinguished it as a reasonable candidate. Other choices such as SHA-1 were also considered but MD5 was found preferable due to the size of the hashed value.

In the communication scenario proposed for SCP, a symmetric key scheme would not scale. Due to its distributed nature and the fact that communication does not necessarily share any common point, every communication pair

[3] The path from the LFN to the AR is again considered to be a multihop route. Nodes which will forward these messages will receive incentives.

would need its own shared secret. Although symmetric algorithms are generally faster, the low resources of the end devices make it impossible to hold one different symmetric key per peer node. Therefore, we chose to adopt an asymmetric cryptographic scheme which allows for a single public and private key pair per host. The public part of the key is then shared within the network and can easily be broadcasted in form of a certificate over the wireless medium. Since we assume temporary connectivity to the fixed network where a Certifying Authority (CA) resides, we have no need for distributed approaches.

For the digital signature choice we considered RSA [8] and Elliptic Curve Cryptosystem (ECC) [9]. Although the first provides faster execution times for the verification operation, which is especially important since SCP requires verification at each intermediate node, the size of the key and other necessary resources to run RSA have pushed it to the second position. ECC produces less overhead in the network and in storage requirements at the end points. In particular, less overhead is imposed by SCP on the network which allows the protocol to be suitable even for real time traffic. We have used our own speed optimized ECC implementation [10] using the Fixed-base comb method and the Montgomery method. Table 1 shows the execution times for different security levels. These measurements were done using a PDA device (the Sharp Zaurus) which we assume to be a realistic destination platform for a charging scenario in ad hoc networks. For SCP, the key size was chosen in accordance to a low lifetime system. This translated into a 163 bit key which is the equivalent security to a 1024 bit RSA. For detailed information, we again refer to [11].

Table 1. Execution Times for Signature Operations based on ECDSA and RSA on a Sharp Zaurus SL-5500G

Security level bit		Time for signature generation [ms]			Time for signature verification [ms]		
ECC	RSA	ECC	RSA	Ratio	ECC	RSA	Ratio
113	512	2.8	13.7	4.9	7.5	1.3	5.7
131	704	3.8	32.4	8.5	11.5	2.5	4.6
163	1024	5.7	78.0	13.6	17.9	4.3	4.1
193	1536	7.6	251.9	33.0	26.0	9.7	2.6
233	2240	10.1	731.8	72.0	37.3	20.4	1.8

3 Implementation Overview

SCP is implemented using C++ for the Linux Operating System (OS). Modules and external libraries are written mostly in the C language. The chosen OS offers many of the utilities and libraries required to work with IPv6, packet capturing and multi-tasking/multi-threading.

We have also chosen two distinct target hardware platforms: the i386 compatible PC (AR) and, the Zaurus PDA (MN, CN, FN, LFN). The latter was chosen to easily test mobility and demonstrate the protocol functionalities. Although most PCs share common architectures, this is not the case for PDAs. Our choice was driven both by the fact that the Zaurus is a PDA in the market that natively runs Linux, and the convenience of the available open source compiler for the StrongArm processor, core of this PDA.

We position SCP between the network and the transport layer. It has strong interaction with the ad hoc routing protocol as well as high level functions of the OSI model which are accessible to the application.

Certain functionalities, such as the underlying ad hoc routing protocol and the authentication methods, have been modularly built into SCP. This simplifies the process of adapting SCP to a certain network. As a side effect, they can also be dynamically loaded at runtime. For this purpose we used the Dynamic Loading Library (libdl) which Linux provides. Through the use of this library, we are able to load external functions, not compiled with our main program, at runtime according to a configuration file. After the program has read the configuration file, it loads the described modules for both routing and AAAC [12]. Moreover, the configuration can be changed and reloaded without stalling SCP. This allows a network provider to update and reconfigure all nodes and the access routers without interrupting the service.

At the Access Router, not all SCP related information should be kept in main memory as some of it is of persistent nature. For this data, we use the relational database model present in mySQL (a free SQL database, available also for Linux). Per node and per charging period SCP data is then inserted and retrieved using a set of SQL queries and a C interface to mysql. In a similar fashion, we combined mySQL databases with the Pluggable Authentication Modules (PAM). PAM is provided by Linux to form a simple modular authentication backend.

Linux itself provides the facilities to queue packets and process them in userspace, where the prototype realization of SCP resides. This greatly simplifies network level operations. Packets are captured into IP Queue (IPQ) using iptables on the hooks provided by the Linux netfilter architecture. The packet is then sent to userspace and processed. Once they are ready to be sent, they are simply re-injected back into the network with possible changes to their payload and format. Should the packet verification fail or any other error occur, the packet is dropped.

By using the capabilities of C++ to their full extent, we have been able to develop a system in which packets are read and built from the bottom and up and then delivered to the higher protocol layers. Each protocol layer was fitted into its own Class which, through inheritance, forms the actual layers as seen in the network.

As to the implementation of some of the security functions and abstraction layers, we used the Secure Socket Layer (SSL) [13] model provided by the Linux OpenSSL project, since it already provides a great amount of cryptographic

primitives. We have even extended OpenSSL to include our speed optimized ECC library.

Both the timer and the event systems required for asynchronous execution have been built into SCP. The functionalities have been divided into different threads that allow concurrent execution, therefore improving scalability. On top of this, the system provides event queuing for function scheduling.

3.1 Implementation Building Blocks

The SCP implementation architecture can be divided into four main layers, as depicted in Fig 1. The first is the Network Layer which consists of the transport for both the signaling and the data flow. In addition we define the kernel as the interface between the Network Layer and our application. For the specific SCP related implementation, we provide two more levels of abstraction: while the Packet Handler deals with the complexity of an event driven multi-threaded system, the SCP implements the actual state machine and contains the interaction between users, sessions and the databases.

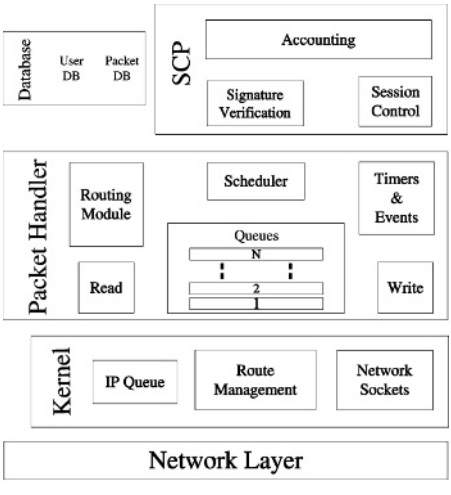

Fig. 1. Software Architecture

3.2 Building Blocks

The building blocks of our implementation are as follows:

Network Layer. We refer to the actual physical layer, the link layer and the IP layer as the Network Layer. It provides the data transmission protocols and mechanisms to retrieve information on the network.

Kernel. We focus on the Linux kernel although the model can be ported to other OSs.

IP Queue - The IPQ system provides packet queuing and processing in userspace by using iptables and special libraries.

Route Management - This part of the kernel deals with the creation and deletion of IPv6 routes. It is mainly used by the routing module to define the path a data packet takes.

Network Sockets - This is the kernel implementation of the Berkeley socket API. It provides the main functions to make use of the transmission protocol implementations from the Network Layer.

Packet Handler. The Packet Handler is an abstraction to provide a simple interface to the most common and basic operations within the program. It is the basis used by the state machine that implements the actual protocol.

Read Thread - This thread is responsible for the polling of new data from IPQ. It reacts to a new message by putting it into the SCP internal queues for processing.

Write Thread - This thread polls the queues for packets that should be sent to the Network Layer.

Routing Module - The routing module is an independent piece of software that discovers the paths between two nodes that wish to communicate. The current implementation supports a module based on static routing (for testing purposes) that depends on a configuration file, and another one using a modified version of the AODV6 HUT implementation.

Scheduler - The scheduler is responsible for service differentiation within SCP. It distributes the most resource consuming operation, the signature verification, unevenly between the different flows[4].

Queues - The Queues represent the form in which SCP differentiates classes of traffic. Packets are grouped together using their IPv6 header flow label field and a different weight is attributed to each queue. This weight is then translated in the processing time each of the queues receives to run the verification function.

Timers and Events - Because SCP is event driven, we define a special class for timers and events. Both of these classes provide support for asynchronous function calls.

Database. This building block provides an interface to the mySQL database.

User Database - This table contains the users identity and attributes to each user a username and a password. Authentication is then performed by using this table from within PAM.

Packet Database - Once the accounting packet has been verified, the information contained in it is dumped in this set of tables for persistent storage.

[4] We define a flow as being a packet with a different value in the IPv6 flow label field.

Secured Charging Protocol. This is the implementation of the protocol state machine.

Signature Verification - In this class we will find the primitives for signature verification. Because this is by far the most computationally expensive operation, it has been implemented separately for performance and scheduling issues.

Session Control - This class tracks the sessions between users in which it participates and the amount of information exchanged between end points. It is also the internal repository for the certificates relevant to other nodes.

Accounting - Finally in this module we store the accounting information for that session, should the node be the last forwarding node of a communication. This information includes the number of bytes transmitted, the number of packets and the hash chain and signature route proof for each packet. This information is cataloged by session on a per packet basis.

4 Graphical User Interface

The SCP's Graphical User Interface (GUI) at the client side provides an internal view of the program running in the nodes.

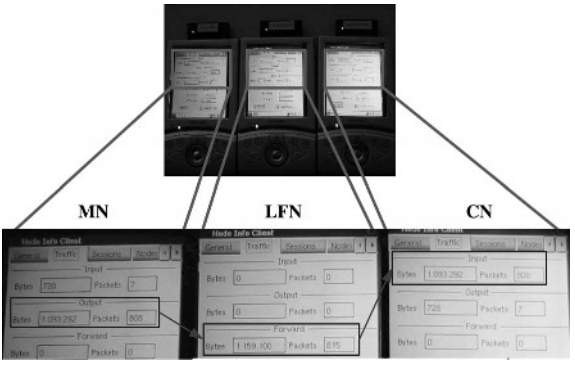

Fig. 2. In depth view of the statistics information in the GUI

The demonstrator setup depicted in Fig. 2 depicts the GUI of three nodes during the forwarding phase. The arrows indicate the flow of the packets between the three nodes while using the SCP implementation. The packets sent from the sending node on the left (MN) are forwarded by the node on the middle (LFN) and finally received by the rightmost node (CN). The LFN is now responsible for transferring this information to the AR in the charging phase. Other information such as certificates and general statistics are also available in the GUI. They are primarily used for debugging purposes.

5 Integration with the AODV6 Routing Protocol

In principle, SCP is independent of the ad hoc routing protocol (should this be proactive, reactive or any other category of ad hoc routing protocol). Nevertheless, certain functionalities have to be implemented into it for SCP to work. They are in particular necessary to calculate the next hop and retrieve the IP addresses. For our demonstration set-up we have chosen to use AODV [14].

Some routing protocol implementations, such as the HUT AODV6, make use of IPQ. Since access to this query process is exclusive and SCP also requires it, we placed special hooks into IPQ to forward packets to the routing module if required. The AODV daemon has been converted into an SCP routing module and both the information functions and the hooks are linked together with the module to empower SCP with the AODV routing protocol.

For proper operation, AODV requires a UDP port. Since some vital routes may not be established during the route discovery process, packets headed for this port cannot be processed by SCP. A rule was added with iptables so that no information passing through this port is filtered by SCP but is instead delivered directly to the AODV routing module.

To allow connections with the fixed network, we follow the same ideas presented by the drafts [15] and [16] on ad hoc Internet connectivity in a mutual direction. We also extended our charging scheme and prototype to nodes that wish to communicate with nodes outside the ad hoc network.

This work is based on a modification of the AODV6 HUT implementation which allows routes to be resolved to outside the ad hoc cloud. The SCP prototype was integrated with this modification in the same way as was described previously.

We make no assumptions on the network topology and allow multi-hop routes, even toward the Access Router.

6 Secure Charging Data Aggregation

In accordance with the last phase of the SCP protocol, accounting information is sent from the LFN and stored at the AR. This data contains the number of bytes transmitted and acknowledged since the last accounting message, the identity of all the nodes that participated in the communication, the disposition of these nodes in the route and a per packet proof of all the values mentioned.

Finally, the data stored at the Access Router contains the individual evidence of every packet transmitted on the network. It is impractical, if not unfeasible, to store all this data in the core network, where the AAA home server lies. Nevertheless, any aggregation scheme must still prevent repudiation. To cope with this problem, an extension to the base protocol has been made by adding new fields to the confirmation reply packet which is sent by the corresponding node, and to the verification request, sent to the Access Router. The former contains a signature that covers the Sequence Number of the confirmation request, the time (with a low resolution), and the number of acknowledged bytes . This sig-

nature is replay attack safe and provides proof that, at that point in time, the correspondent node acknowledged a certain number of bytes. As to the latter, the verification request, it has been completed with the date and sequence number of the confirmation request, to be used with the AAA architecture at any time and to allow the verification of the afore mentioned signature.

In the end, the information necessary at the AAA can be reduced to the total amount of money to be received by and charged to each node. This bulk information must follow the non-repudiation rules set above.

This process provides non-repudiation at the AAA architecture level of the aggregated data.

Data aggregation assumes different lifetimes for the stored data. The information kept at the Access Router is seen as having a lifespan ranging from days up to two or three months while the data in the AAA home server should not expire at all. It is envisioned that there could be points of caching of the accounting information throughout the core network. This would facilitate accounting and charging for foreign nodes at these intermediate levels. A AAA server that wishes to have a more detailed view on the accounting information gathered at the Access Router, can do so explicitly, up until the expiration time.

7 Connecting to AAA

In the real world users are distributed between different Service Providers, which may or may not have contracts between them, and may or may not let them access their services. When integrating an ad hoc network into a real case scenario, we must take this into account. Moreover, the access to one domain may be provided by several AR which share and distribute the functionality of providing authentication and accounting. In the case of SCP, should an Access Router require to verify a hash chain, to confirm the route it must have access to the shared keys. The same way, if it must validate the identity of a node, it must have the node's certificate. It is also feasible that a node registered with one AR and then provides the accounting information to another. This case requires that the second AR can verify the data and the identity as stated above.

A MN is binded to its AAAH. The information on the node's identity is only present at this point. For security and privacy issues, it must never leave the AAAH. Therefore, identity verifications must also be routed to the home domain. We distinguish between the shared keys, which must be shared by the AR of the local domain, and the node's identity and certificate, which must be issued at the home domain, yet verifiable at the local domain. Each AR serves as a translator between the MN and the AAA architecture. The messages exchanged between the MN and the AR in the Registration phase and the Accounting phase are decoded, verified and the appropriate fields are filled accordingly and inserted in the AAA protocol. The message is then sent to the home domain where decisions and final accounting are processed. A top level AAA protocol such as Diameter [17] will be responsible for routing the messages from the AR,

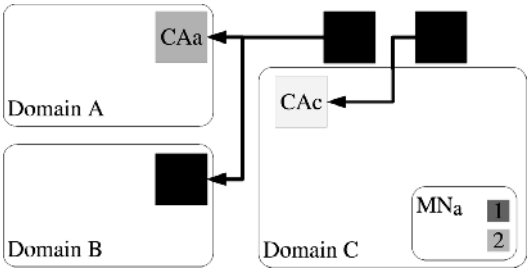

Fig. 3. Hierarchical CA architecture

through the AAAL to the AAAH. This procedure is well defined within such protocols and therefore will not be further discussed in this document.

7.1 Distributed Hierarical Certifying Authorites

To solve the problem of verifying a node's certificate, we propose a two level hierarchical distribution of CA in such a way that any node can verify the authenticity of any other node's certificate with only a subset of the CA keys.

We associate each leaf CA (or several) to a local domain. A node wishing to verify a certificate issued by the local CA must first verify that CA's certificate with it's subset of high level CA keys. Should this verification prove successful, we extend the trust chain to the local CA and can now verify the node's certificate as valid and trustworthy.

The certificates should be obtained from an interaction between the Public Key Infrastructure (PKI) and the AAA architecture. This paper does not further explore this synergy.

8 Testbed, Tests and Measurements

As part of several demonstrators, SCP was tested for scalability, performance and network impact. Although the size of the current demonstrator cannot account for the first, we have tested this SCP implementation to a maximum of 5 intermediate hops and for different traffic classes (audio, video, http).

The testbed is composed of four PDAs and two PCs. One PC is the AR and all the others act as ad hoc nodes. The nodes in this testbed do not have pre-determined roles. We conducted our tests in parallel and independent of the node's position. The position of the nodes does not impact the node to which it communicates because, in order to facilitate our experiments, we have pre-set the routes (the use of AODV is also available). All PDAs have a direct connection to the AR, even though this is not a requirement, and we established the order in such a way that the PC is never an intermediate node. This provides a worst case scenario in what concerns to resources consumed by the low powered nodes.

Note that SCP is only acceptable if the end-to-end delay and the jitter in the forwarding phase still allows for real time traffic. Our tests show asynchronous traffic is not affected by the presence of SCP.

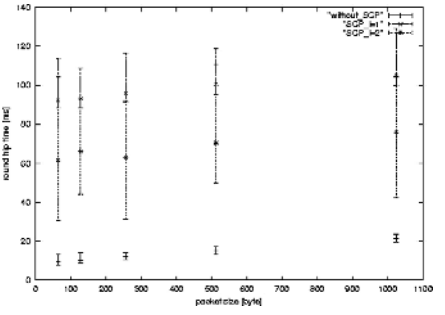

Fig. 4. Round trip times and jitter on the demonstrator in the absence and in the presence of SCP with verification rate (1/l) l=1 or l=2 and varying packet sizes

Our tests have taken the following parameters into account:

End-to-End Delay. The usage of cryptographic primitives, even if optimized for our demonstrator destination platform, severely affects the time each node needs to forward a packet. We have estimated a delay of 17ms per node at a full verification rate[5]. If we decrease the verification rate to 50% (verification of every forwarded packet at each intermediate node), which we have accepted as a still reasonable security threshold, in an average 4 hop network, we observe an end to end delay of around 40ms. This value is well within real time applications such as audio streaming that require a maximum end to end delay of around 50-70ms.

Network Overhead. For the network overhead, in addition to SCP's signaling messages which are out-of-band, we consider: A 16 byte MD5 hash value of the hash chain, the 4 byte seed for the hash chain, the route and the signature of the sender over the packet that is usually around 22 bytes. Typically, on the 4 hop network example above, we should count on 90 byte packet increase by the time the packet reaches the CN. These values depend on the level of security enforced and the number of intermediate nodes.

CPU consumption. Even with the great computational effort of the verifications we notice no humanly perceivable misbehavior on real time applications (such as audio decoding). By observation, we have determined that the implementation, even in low power devices, has no major impact on the device's performance.

The following measurements were done in the testbed described at the beginning of this section using *mgen* [18] and *netperf* [19] for connectionless and connection oriented traffic respectably. In the plot of Fig. 4 we solely focus on UDP (connectionless) traffic.

[5] This is only a prototype implementation. Once the concept is proven to work, one would implement ECC in hardware, resulting in a 10 factor faster execution times of digital signature operations.

9 Further Work and Conclusions

We believe that this work is a good starting point to combine ad hoc with fixed networks in what concerns charging. Nevertheless, it needs further exploration. Once a node reaches an administrative domain under the control of several ISPs, it must negotiate prices and capabilities with the nodes currently being served by that Access Router. A node may not wish to conform to the pricing scheme first presented in that network and therefore this is a challenging issue and certainly worth analyzing.

It is also our belief that other work in this area can be applied to deliver a gaming theory based mechanism that provides a probabilistic model for the price negotiation.

We also hope to realize a hardware implementation of the cryptography functions used in order to speed up our test results.

This paper proposes SCP as the reunion point of wired and ad hoc networks' charging and accounting, and outlines the future work in the interaction between the both.

We also show, both by prototype implementation and evaluation, that SCP can indeed be used in real life scenarios to provide a secure, incentive based approach to solve the selfish node problem.

References

1. Buchegger, S., Boudec, J.Y.L.: (Performance analysis of the CONFIDANT protocol (cooperation of nodes: Fairness in dynamic ad-hoc NeTworks)) 226–236
2. Marti, S., Giuli, T., Lai, K., Baker, M.: Mitigating routing misbehaviour in mobile ad hoc networks. In: 6th International Conference on Mobile Computing and Networking, ACM MobiCom 2000 Conference (2000) 255–265
3. Buttyan, L., Hubaux, J.P.: Nuglets: a virtual currency to stimulate cooperation in self-organized mobile ad hoc networks. Technical Report DSC/2001/001, Swiss Federal Institute of Technology, Lausanne (2001)
4. Zhong, S., Yang, Y., Chen, J.: Sprite: A simple, cheat-proof, credit-based system for mobile ad hoc networks (2002)
5. Lamparter, B., Paul, K., Westhoff, D.: Charging support for ad hoc stub networks. Elsevier Journal of Computer Communications **Elsevier Science** (2003) Special Issue on Internet Pricing and Charging: Algorithms, Technology and Applications.
6. de Laat, C., Gross, G., Gommans, L., Vollbrecht, J., Spence, D. (Technical report)
7. Johnson, D.B., Maltz, D.A.: Dynamic source routing in ad hoc wireless networks. In Imielinski, Korth, eds.: Mobile Computing. Volume 353 of The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers (1996) **http://athos.rutgers.edu/ imielins/book.html.**
8. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems (reprint). Communications of the ACM **26** (1983) 96–99
9. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of Computation **48** (1987) 203-209
10. Riedel, I.: Security in ad hoc networks: Protocols and elliptice curve cryptography on an embedded platform. Diploma thesis at NEC, University of Bochum (2003)

11. Lamparter, B., Paar, C., Weimerskirch, A., Westhoff, D.: On digital signatures in ad hoc networks. IEEE Journal on Selected Areas in Communications **'Wireless Ad Hoc Networks'** (2004) submited.
12. de Laat, C., Gross, G., Gommans, L., Vollbrecht, J., Spence, D.: Generic AAA architecture. Internet Request for Comment RFC 2903, Internet Engineering Task Force (2000)
13. Hickman, K.E.B.: The SSL protocol. RFC draft, Netscape Communications Corp. (1994) Version 1.0.
14. Perkins, C., Belding-Royer, E., Das, S.: Ad hoc on-demand distance vector (AODV) routing for ip version 6. Internet-Draft draft-perkins-aodv6-01.txt (work in progress) (2001)
15. Jelger, C., Noel, T., Frey, A.: draft-jelger-manet-gateway-autoconf-v6-01.txt. Internet-Draft draft-jelger-manet-gateway-autoconf-v6-01.txt (work in progress) (2003)
16. Cha, H.W., Park, J.S., Kim, H.J.: Support of internet connectivity for aodv. Internet-Draft draft-cha-manet-AODV-internet-00.txt (work in progress) (2004)
17. Calhoun, P., Loughney, J., Guttman, E., Zorn, G., Arkko, J.: Diameter base protocol. Internet Request for Comment RFC 2903, Internet Engineering Task Force (2003)
18. : Mgen - the multi-generator toolset. http://manimac.itd.nrl.navy.mil/MGEN/ (2004)
19. : The public netperf homepage. http://www.netperf.org/netperf/NetperfPage.html (2004)