LUNAR – A Lightweight Underlay Network Ad-hoc Routing Protocol and Implementation

Christian Tschudin, Richard Gold, Olof Rensfelt and Oskar Wibling

Abstract— **In this paper we describe LUNAR, a lean and efficient routing protocol for wireless ad hoc networks. We report on its implementation, on performance comparisons and on a formal validation result. The protocol departs in several ways from standard ad hoc routing protocols: it does not feature route repair, route caching, route maintenance or packet salvation. Nevertheless it closely matches the performance of AODV in settings inside of the "ad hoc horizon". We discuss the philosophy of LUNAR, report on several ports including versions for Windows and a stripped down μLUNAR capable of running inside a microcontroller and operating over infrared or Bluetooth.**

Keywords— **Wireless networks, ad hoc routing, routing protocol implementation, LUNAR, formal validation.**

I. Introduction

AD-HOC networks are typically described as a group of mobile nodes connected by wireless links where every node is both a leaf node and a router. This flat routing environment causes many challenges which have resulted in a slew of routing protocols and research projects designed to solve the problems posed by ad hoc networks [7], [15], [10], [8].

Many ad hoc routing protocols started with a simple concept and then added "essential" features like route maintenance or packet salvation for staying ahead in the competition among protocols. While this happened at the level of protocol design and simulations, the availability of *implementations* of these refined ad hoc routing protocols is far from satisfying; Stability as well as ease of installation and configuration are other domains that usually are neglected too. This is in contrast to the continuous sophistication of ad hoc routing protocol specifications as is visible in their version numbers: All four top MANET protocol candidates went through 10–13 revision cycles, but for some protocols there is no public implementation available at all, others more or less have one reference implementation, and today none is cross-platform (Windows, Linux, Mac and PDAs).

A. The LUNAR Approach

The aim with LUNAR was to explore novel ad hoc routing strategies and to constrain ad hoc routing protocol design to pragmatic boundaries. Low protocol complexity helps to easily implement LUNAR in other environments, as we demonstrate with μLUNAR. Lean protocols also form a good starting point for adding the important self-configuration elements, as we also show in this paper.

LUNAR adopts a hybrid routing style as it combines elements of both reactive and pro-active ad hoc routing approaches. It is reactive in the sense that it discovers paths only when required; It is pro-active in the sense that it rebuilds active paths from scratch every 3 seconds, even if everything is fine with the current path. This removes the need for additional path maintenance procedures and link repair actions and reduces the complexity of the protocol. Another design choice was the separation of delivery paths. Each host pair potentially has its own path and softstate associated with it: It is the duty of the originating node to keep that state alive. This makes implementation easier as, for example, intermediate nodes can now be fully reactive and do not have to start protocol activities by themselves.

Christian Tschudin is a member of the Computer Networks Group, University of Basel, Switzerland. E-mail: christian.tschudin@unibas.ch . Richard Gold, Olof Rensfelt and Oskar Wibling are with the Communication Research Group of Uppsala University, Sweden. E-mail: rmg@it.uu.se, olof.rensfelt.4113@student.uu.se, oskarw@it.uu.se .

B. Ad hoc Horizon

LUNAR limits itself to 3 hops. This decision was based on our experience (when experimenting with the standard MANET protocols over IEEE 802.11) that 3 hops is already pushing the limits in many ways. We hypothesize that there is an *ad hoc horizon* beyond which it becomes ineffective to handle topology changes as they occur in mobile wireless networks. First, when multihop routing is in place, it means that the wireless cards operate close to their limits, resulting in a highly fluctuating connectivity space: Slight position changes or objects getting in the way drastically change the neighbor set. Second, the freshness of routing information decays rapidly with the number of hops – attempts to do local repair potentially mask or at least delay the recognition of trouble spots. Third, the discovery and maintenance of routes by the use of flooding beyond the ad hoc horizon disturbs the communications of remote nodes more than it serves the local needs.

C. Underlay Routing and Internet Access

Another major departure from the MANET protocols is that LUNAR positions itself below IP. Instead of making ad hoc routes visible at the IP level, we create a subnet illusion. To the IP stack is looks as if the mobile node was connected to a LAN. Inside LUNAR we use an underlay at "layer 2.5" which emulates the LAN behavior by establishing point-to-point multihop paths or point-to-multipoint multicast trees to mimic the broadcasts at the LAN level. The underlay approach leads to immediate implementation benefits as LUNAR does not have to interact with the IP routing tables. Moreover, the underlay discovery mechanisms inside LUNAR permits to add self-configuration elements (address assignment, gateway discovery) in a very straightforward way.

II. Related Work

Routing below the IP layer for ad hoc networks was independently adapted by [1] using label switching which

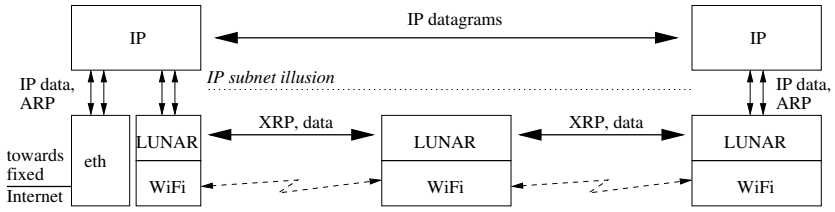


Fig. 1. LUNAR as an underlay to the IP layer

is equivalent to the "selectors" that we use in LUNAR. A similar project is [2] where the authors also aim at putting L2.5 routing logic inside the (wireless) network interface card.

There are four main routing protocols currently being considered for standardization by the IETF MANET group. Two are reactive (DSR & AODV) and two are proactive (OLSR & TBRPF). There is a DSR click router implementation available which seems stable, unfortunately it was not available at the time that this work was originally done. For AODV, which was recently raised to full RFC status, there are four full AODV implementations are available. For our comparisons we used AODV-UU [14] which is a stable and mature implementation. OLSR [16] is available in a stable implementation from INRIA and appears to work well, therefore it was also included in comparisons. Unfortunately, there is no publicly available implementation of the TBRPF protocol.

For AODV, formal validations have been carried out by the Verinet group [19] at the University of Pennsylvania. Using a theorem prover and a SPIN model of AODV in a 2 node setup (with an AODV router environment), it could be shown that – with additions – it is in fact a loop free routing protocol.

III. The LUNAR Protocol

LUNAR presents itself towards the IP stack as a network interface card that attaches to a local area network. All nodes participating in the LUNAR network will appear as being one IP-hop away. Internally, this connectivity is implemented by forwarding data packets over multiple hops.

Figure 1 shows the position of the LUNAR network with respect to the traditional IP stack. The design of LUNAR is based upon the SelNet underlay network [17] which is a frugal forwarding abstraction designed to support a wide range of data forwarding and routing styles. Typically, SelNet is implemented at L2.5, but can also be put at L3.5 as an overlay where it can be used for packet redirection.

The main idea behind LUNAR is to link ad hoc path establishment to ARP. Historically, DSR also followed this approach by doing multihop ARP [9]. Typically ARP is confined to broadcasting only to the subnet that the node is currently residing in. For LUNAR, we decided to allow nodes which receive such resolution requests to rebroadcast them to reach nodes which are outside of the original radio range of the requesting node.

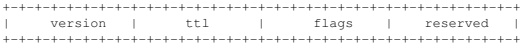
A. SelNet

The SelNet network basically offers a demultiplexing service. An incoming packet will be dispatched to an appropriate handler routine based on a single packet header field called "selector". As a packet is forwarded through a SelNet network, it will have a different selector on each leg of its path because the SelNet network will rewrite the combination of Ethernet address and selector on each hop, in the same way as IP forwarding rewrites the Ethernet address for each subnet crossed. A special selector value is defined for XRP packets (eXtensible Resolution Protocol) which enables to create rewriting state in remote nodes and, in the case of LUNAR, to address the LUNAR specific forwarding logic. Delivery paths dynamically obtain their own set of selector values such that XRP control traffic and the multiple ad hoc delivery paths are kept separated.

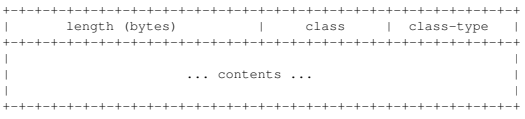
B. SelNet Control Messages (XRP)

XRP (eXtensible Resolution Protocol) is SelNet's external data format for requests and replies. Its name stems from the philosophy that "resolution" is a mandatory step before using SelNet's forwarding capabilities. An example would be the resolution of a neighbor's ethernet address such that one can send data to it, or a LUNAR resolution where we want to resolve an IP address to a multihop path. In both cases SelNet will set up the necessary state inside the local host as well as in the network, and return an opaque "handle" in form of a selector value. In a second step on can use this selector for sending packets to the now resolved destination; The corresponding API consists of a simple `selnet_demux(selector, data, length);` procedure.

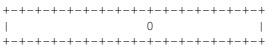
XRP messages are mere containers for parameters, its format was inspired by the CASP proposal [3] and RSVP. Here we describe the generic data structures and point to some LUNAR specific parameters. XRP messages begin with a header which is 4-Bytes long:



This header details the version, Time To Live (TTL) and the flags associated with that header. After the header, we have the various parameters that we wish to send. There can be multiple XRP parameters per packet.



A "null" object closes this list so XRP becomes a self-delimiting packet format.



A typical LUNAR "Route Request" would contain the following parameters:

- Request series (Len=12, class=request series, ctype=sel)
- Address to resolve (Len=8, class=target, ctype=IPv4)
- Requested resolution (Len=4, class=reqstyle, ctype=sel/eth)
- Reply address (Len=20, class=reply addr, ctype=sel/eth)

The request series (opaque 64 bit field in form of a selector, ctype=sel) is used to identify each discovery wave for preventing broadcast storms. We request the resolution of an IP address (address to resolve, ctype=IPv4) and wish to obtain the result in form of a selector/ethernet pair that encodes the first hop of the delivery path (requested resolution, ctype=sel/eth). The reply address is our local address where we wish to receive the result of this request.

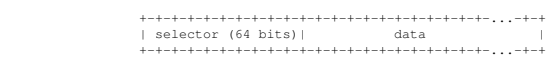
The LUNAR "Route Reply" message will have only one parameter contining a selector and ethernet address value pair:

- Forward address (Len=4, class=reqstyle, ctype=sel/eth)

This information tells the requesting node where to send data traffic to. All data packets sent to this port will be forwarded to the stack whose IP address was given in the original route request.

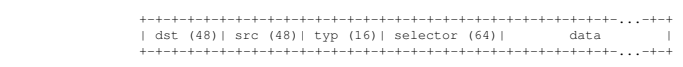
C. SelNet Packet Format

As can be seen in the XRP example above there is no field that specifies the LUNAR context in which the resolution should be carried out. In fact, this information is encoded as a "well known" selector port to which all LUNAR requests have to be sent to, whereas LUNAR replies are sent to a dynamically allocated selector port as indicated in the parameter list. This relates to SelNet's basic demultiplexing philosophy and its one-header-field packet format:



The same selector+data format is used for a) XRP requests, b) XRP replies as well as c) pure data traffic; Differentiation is made by the selector value only.

In order to underlay IP we define a new ethernet frame type such that IP and SelNet frames can be distinguished at arrival. A complete SelNet frame over ethernet thus has the shape:



D. LUNAR Protocol Logic

LUNAR interconnects with IP by trapping all control traffic and translating it into explicit signaling at the SelNet level. When the IP stack issues an ARP request for an IP number's Ethernet address, LUNAR translates this to a "route request" (RREQ) expressed as an XRP message. This RREQ is broadcast to all neighbor nodes. Rebroadcasting logic is then responsible for checking duplicate RREQs (each request has a unique ID which is stored on each node it arrives to), for propagating the resolution information to the next neighbors and for temporarily storing return information for the reply.

When the target node is reached, it sends back a "route reply" (RREP) – this time using a unicast message – to the node from which it received the broadcast. Using the return information there, this reply message travels back towards the originator while at the same time creating a data delivery path towards the target node. Data traffic travels inside the delivery paths that LUNAR has established.

LUNAR also has to take care of IP broadcasts that the communication stack might be sending (e.g. broadcast pings). To this end we use a similar flooding based discovery mechanism and express such resolution requests as XRP messages too. In order to emulate a L3 broadcast we send a L2 RREQ broadcast: all neighbors match the discovery request and will rebroadcast it before replying. We delay this reply a little in order to fuse it with the replies from downstream nodes. Depending on how many downstream nodes replied, the intermediate node will install either a "broadcast forwarding handler" or a "unicast forwarding handler". The net effect is the creation of a delivery tree where edges are using broadcast in case a node has more than 2 child nodes and using unicast otherwise.

All data forwarding state in intermediate nodes is phased out after approximately 6 seconds. Therefore, the originator of a delivery path would have to periodically refresh it. Instead of refreshing, we chose a more suitable strategy for ad hoc networks by building a *new* path every 3 seconds. This way LUNAR is able to react swiftly to topology changes, although it does not use any hello beacons or other link layer mechanisms to discover broken "links". Old state is garbage collected by each node without additional signaling overhead.

E. Intercepting DHCP for Address Self-Configuration

We have already described how LUNAR intercepts ARP messages and turns them into RREQs for establishing forwarding paths. Similarily, LUNAR intercepts DHCP messages and translates them into its own address allocation and resource discovery strategy. In fact, each LUNAR node implements a fake DHCP server. If the DHCP client asks for a specific IP address, LUNAR will try to resolve this address using the XRP messages. If no delivery path can be set up to the given IP address, we assume that the address is not in use and LUNAR grants the corresponding lease to the DHCP client. However, if a RREP message was received, a new random IP address is picked and tested several times before handing it to the client.

Gateways are also solicited using XRP resolution messages. In a variation of LUNAR's path setup described above, we allow several XRP replies to travel back instead of a single reply. Back at the originator we collect them and return the list of available gateways in the DHCP reply message.

IV. Implementation

Figure 2 shows the software decomposition of LUNAR. In a first approach we implemented LUNAR as a Linux user space program that interposes itself between the IP stack and the wireless card driver. The TUN/TAP device is used to interface with the IP stack while NETLINK provides all information on pending ARP requests in an early phase. Alternatively, one can also capture the ARP packets once the IP stack sends them out and use ARP reply

packets to signal that a delivery path was found. However, because there is no ARP packet type that could be used to *remove* an ARP entry, we still need direct access to the ARP cache. The tight interfacing with ARP is essential for throttling the IP stack as soon as a LUNAR delivery path times out. This avoids that any (TCP) packets are lost because the IP stack will know that it has to redo an ARP before continuing with the data transmission. In order to avoid a stop-and-go behavior at the ARP interface level we let LUNAR create a new delivery path after 3 seconds in parallel to the existing one and switch silently to the new. LUNAR removes an ARP entry only if no traffic is observed for a 3 second period and the path times out.

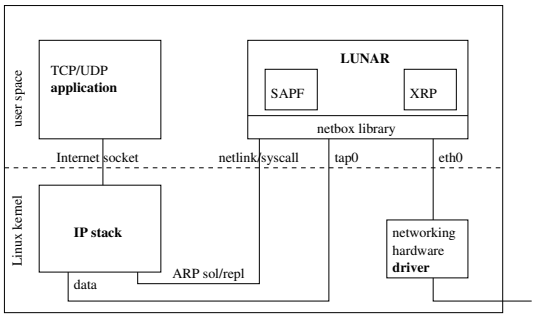


Fig. 2. LUNAR node architecture (user space implementation).

A new version of LUNAR has been development which is fully kernelized and which does not have the TUN/TAP and NETLINK dependency. This makes the deployment of the LUNAR module simpler as users do not have to recompile their kernel for TUN/TAP and NETLINK support. The kernelized LUNAR version also creates an ethernet device which is used by the IP stack like any other subnet technology. The source code is published under the GPL and can be accessed at [11].

A. Measurements

We measured the performance of LUNAR in a controlled setting of a linear network of three stationary nodes and one mobile node that roams along this linear network (see [12] for details). The following table gives the result for UDP traffic (ping and MP3 streaming):

Protocol	*Ping*	*MP3*
OLSR	89.0%	91.9%
AODV-UU	91.9%	97.9%
LUNAR	96.5%	96.8%
AODV-UU+SNR	99.1%	99.7%

LUNAR had better PING performance than plain AODV which is due to broadcast messages reaching farther than unicast messages: Because AODV trusts broadcast messages for identifying delivery paths, it can have problems in so called "communication gray zones" [12]. LUNAR is not subject to this problem because it does not really on HELLO beacons for link break dectection.The patched AODV-UU+SNR version, which eliminates the gray-zone problem, now outperforms LUNAR also for the PING case. However, we consider the LUNAR performance of 96% to be sufficiently good for justifying not adding new protocol features.

B. A Tiny Multihop Ad Hoc Access Point

In order to demonstrate the reduced complexity of working below the IP layer, we fitted LUNAR and a Linux system on a single 1.4 MB floppy disk. This "LUNAR-on-a-floppy" features an automatic gateway and contains a NAT module which will map the 192.168.42.x LUNAR subnet addresses to a topologically correct IP number in the fixed Internet. As a result we have implemented a self-configuring access point which enables mobile nodes to get Internet connectivity over multiple ad hoc hops.

C. LUNAR for Embedded Systems

LUNAR's simplicity also permitted us to implement the routing protocol in the very constrained environment of the LEGO Mindstorms RCX. The RCX is equipped with a h8300 Hitachi microcontroller with 32 KBytes of RAM and communicates over an infrared device running at 2400 Baud. The resulting system consists of a stripped down BrickOS (7KB), μLUNAR (3.5KB), Van Jacobsen header compression (4.5KB), μIP[5] (3.5KB), a Web server and the application (1.8KB).

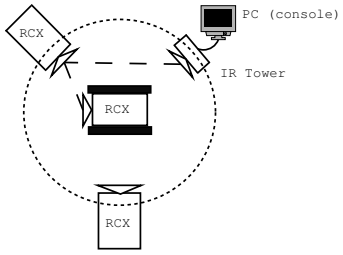


Fig. 3. Multihop IR coverage for mobile LEGO Mindstorm robot.

As an application we demoed the steering of an RCX-car over IP and the delivery of a dynamic HTML page displaying the unit's status. Because the IR device has a limited reception angle of 60^{o}, we covered a circle shaped area with two additional LUNAR-enabled stationary RCX nodes (see Figure 3). The mobile node in the center will always be able to communicate with the console via either a one-hop or a two-hop path.

To highlight the extremely constrained environment we note here that in μLUNAR we merged the link layer address and the selector values, effectively making layer 2 becoming the SelNet layer and shrinking the Ethernet+selector address fields from 48+64 bits down to 16 bits.

D. LUNAR for Bluetooth Scatternets

We have developed a LUNAR version which enables us to route UDP and TCP traffic over multiple Bluetooth piconets. A Bluetooth piconet always has one master node which connects up to seven slave nodes. All communication inside a piconet passes through the master node which regularly polls the slaves. New nodes join a piconet by a lengthy *inquiry* process and will become slave nodes. In order to interconnect two piconets i.e., forming a so called scatternet, it is necessary that one node switches back and forth between the master role in one piconet and the slave role in the other.

Porting LUNAR to Bluetooth was not a straightforward task. Firstly, there is no broadcast functionality within piconets, all data communication is connection oriented. Secondly, not all connected nodes are able to inquire for new nodes, neither are all connected nodes fully discoverable. Currently we solve this problem by letting idle slaves temporarily disconnect from the master and look for new piconets to bridge to and then feeding this information back as if it was gathered via a broadcast mechanism. This proactive activity is necessary because of performance: A source node, which according to LUNAR is responsible for establishing the delivery tunnel and sending the data, would otherwise have to frequently disconnect and subsequently reconnects only for finding nodes in potential neighbor piconets.

E. LUNAR for Windows

In an effort to provide LUNAR for multiple platforms, a Windows XP/2000 version has recently been produced. It is based on the new kernelized Linux version of LUNAR and is implemented as an NDIS intermediate network driver [13]. NDIS [1] is an interface for network drivers used in the latest Windows operating systems. In this manner, Ethernet packets going out to or coming in from the network card can be intercepted and new packets injected in both directions. This is the main functionality that is required by LUNAR since it operates at a layer 2.5.

The porting process was greatly simplified due to the strict modularization of kernelized LUNAR which separates the core logic from the networking and other platform specific details. Hence, the core logic is unmodified in the Windows version and the only parts that have been changed are the LUNAR `netbox` and `platform` modules. These modules respectively contain networking and various other platform specific functionality needed by LUNAR. To this end we wrote an NDIS wrapper which offers the same (Linux kernel) functionality that the LUNAR kernel module relies on.

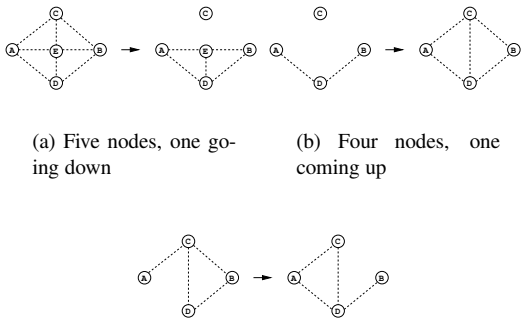
V. Discussion

A. Forced Path Re-establishment

At first sight, the forced re-establishment of delivery paths after 3 seconds seems unusual and inefficient. However, it keeps up well with the actual behavior of other ad hoc routing protocols. It takes approximately 2 seconds for AODV to conclude that two consecutive HELLO messages from a neighbor were lost and that the topology changed, before AODV will start its route repair or rediscovery procedure. For LUNAR, the change in topology can occur anytime inside the LUNAR interval thus on average at 1.5 seconds. We were able to observe this faster reaction time in a qualitative way when listening to streaming of MP3 over AODV and LUNAR enabled ad hoc networks.

B. Formal Validation

LUNAR relies on the efficient forwarding of data packets using a simple packet format which features only a single field called the "selector". Hence, there is no TTL field for data packets which can prevent packets from looping, should the routing protocol configure such a looping forwarding path. One safety net consists in that all forwarding entries in a node keep a "forwarding credit", which is decremented by every packet passing through as well as a soft state garbage collection mechanism. Nevertheless, it is important that the routing protocol does not lead to forwarding loops.



(a) Five nodes, one going down

(b) Four nodes, one coming up

(c) Simultaneous change of connectivity

Fig. 4. Example classes of topology changes.

In order to verify the correct operation of LUNAR we have carried out formal verifications using SPIN [6]. A LUNAR version has been implemented in the modeling language PROMELA in about 450 lines of code including topology generation. We have used this model to successfully verify the correctness of the established paths by varying over the topology.

Due to the state space explosion of this exhaustive search approach, we are so far limited to verify configurations with up to 5 nodes and belonging to selected topology classes. Figure 4 shows various topology changes that were considered (single nodes coming and going away, and connectivity being simultaneously gained and lost at different places in the network).

VI. Conclusions

We have introduced the LUNAR ad hoc routing protocol which targets the common-case of network clouds with 10-15 nodes and a diameter of up to three hops. We believe that such settings will be the most popular ones where ad hoc networks can and will be put into operation. More specifically, in larger settings and for IEEE 802.11 there are such severe degradations occuring under any ad hoc routing scheme that we do not consider this to be a relevant use case that a routing protocol should try to address.

Although the LUNAR protocol does not include any route repair or packet salvation optimizations, it offers comparable performance at less than half of the code size of other MANET protocols (despite LUNAR containing self-configuration and Internet connectivity logic). LUNAR successfully exploits the constraints assumed concerning the network size, which justifies the extremely lightweight approach.

[1] Network Driver Interface Specification

ACKNOWLEDGEMENTS

The authors would wish to thank Henrik Lundgren and Erik Nordström for their performance comparison. We also thank David Rosén for his work in the μLUNAR project.

REFERENCES

[1] Arup Acharya, Archan Misra and Sorav Bansal. "A Label-switching Packet Forwarding Architecture for Multi-hop Wireless LANs" Proc WoWMoM'02, Sep 2002, Atlanta, USA

[2] R. Bernasconi, I. Defilippis, S. Giordano and A. Puiatti. "An Enhanced MAC Architecture for Multi-hop Wireless Networks". In Proc IFIP 6th Personal Wireless Communications PWC2003, Venice, LNCS 2775, Sep 2003.

[3] CASP: Cross Application Signalling Protocol. Henning Schulzrinne http://www.cs.columbia.edu/IRT/casp/

[4] DSR Click Router Implementation homepage Homepage: http://pecolab.colorado.edu/DSR.html

[5] Adam Dunkels. "uIP - A Free Small TCP/IP Implementation for 8- and 16-bit Controllers" http://www.dunkels.com/adam/uip/

[6] G.J. Holzmann. The Spin Model Checker, Primer and Reference Manual. Addison-Wesley, 2003.

[7] IETF MANET Working Group. MANET Charter, 2000. http://www.ietf.org/html.charters/manet-charter.html.

[8] Philippe Jacquet, Anis Laouiti, Pascale Minet, Paul Muhlethaler, Amir Qayyum, and Laurent Viennot. Internet draft: Optimized link state routing protocol, 2001. http://www.ietf.org/internet-drafts/draft-ietf-manet-olsr-05.txt.

[9] David B. Johnson. "Routing in ad hoc networks of mobile hosts". In *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, U.S., 1994.

[10] David B. Johnson, David A. Maltz, Yih-Chun Hu, and Jorjeta G. Jetcheva. "The dynamic source routing protocol for mobile ad hoc networks", 2000. http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-04.txt.

[11] LUNAR home pages: http://cn.cs.unibas.ch/projects/lunar/ and http://www.docs.uu.se/selnet/lunar/.

[12] H. Lundgren, E. Nordström, C. Tschudin "Coping with Communication Gray Zones in IEEE 802.11b based Ad hoc Networks" In *WoWMoM 2002*. http://www.docs.uu.se/docs/research/projects/scanet/aodv/grayzones.pdf

[13] Microsoft Corporation. "Network Devices and Protocols: Windows DDK - NDIS Intermediate Drivers". http://msdn.microsoft.com/library/en-us/network/hh/network/301int_0x2f.asp.

[14] Erik Nordström and Henrik Lundgren. AODV-UU: AODV Implementation. http://www.docs.uu.se/scanet/aodv.

[15] Charles Perkins and Elizabeth M. Royer. Internet draft: ad hoc on-demand distance vector (AODV) routing, 2000. http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-07.txt.

[16] Adokoe Plakoo and Anis Laouiti. OLSR implementation. http://menetou.inria.fr/olsr/.

[17] Christian Tschudin and Richard Gold. "SelNet: A Translating Underlay Network". Uppsala University Technical Report 2003-020

[18] Christian Tschudin and Richard Gold. "Lightweight Underlay Network Ad-Hoc Routing" implementation. Uppsala University Technical Report 2003-021.

[19] Verinet group. Home page: http://www.cis.upenn.edu/verinet.

[20] Oskar Wibling. "Porting LUNAR to Windows". Communications Research Group, Technical Report. Uppsala University, 2003.