LUNAR
Lightweight Underlay Network Ad-Hoc Routing

Christian Tschudin and Richard Gold*
Department of Computer Systems
Uppsala University, Box 325
S-75105 Uppsala, Sweden
tschudin|rmg@docs.uu.se

April 2002

Abstract

In this paper we present an new ad hoc routing system based upon simple principles regarding the routing strategy and the implementation approach. In the routing area we (re-)introduce the end-to-end principle, letting the communicating end nodes make the decisions concerning the behaviour of intermediate nodes. We adopt a routing strategy that is a mixture of on-demand and pro-active routing in order to minimize the possible down-times of communication paths. Implementation-wise we use explicit "resolution commands" sent to neighbour nodes to provide LUNAR functionality. A freely available implementation has been produced that includes auto-configuration of IP network addresses and default gateway routing, making LUNAR a fully self-configuring ad-hoc routing solution which supports both unicast and broadcast styles of communication. In a direct comparison LUNAR matched or outperformed the available Linux implementations of AODV and OLSR.

Keywords: Ad-Hoc Routing, Lightweight Implementation, Underlay Networks

1 Introduction

Ad-Hoc networks are typically described as a group of mobile nodes connected by wireless links. The nodes in an Ad-Hoc network normally operate in a Peer-to-Peer manner where every node is an endpoint and every node is a router. This flat routing environment causes many challenges which have resulted in a slew of routing protocols and research projects designed to solve or ameliorate the problems of Ad-Hoc networks [1, 2, 3, 4].

In this paper we describe the LUNAR ad hoc routing system. With the common case in mind of setting up an ad hoc network with a dozen of nodes all reachable within 3 hops, our main goal with LUNAR is to reduce the complexity both of the routing algorithm and the implementation. LUNAR grew out of a reaction to the many MANET protocol proposals for which ever more refined documents are produced but which still lack robust implementations for example for the Linux operating system.

*Work done whilst second author was at FhI FOKUS, Berlin and was supported by the European Union COST263 action on *Quality of Future Internet Services*.

The quest for simplicity and essentials of an ad hoc routing protocol has lead us to adopt a mixed routing style: LUNAR combines elements of both on-demand and pro-active Ad-hoc routing approaches. It is on-demand in the sense that it discovers paths only when required; It is pro-active in the sense that it rebuilds paths from scratch at fixed intervals; together, for example, this removes the need for additional path maintenance procedures and link repair actions. By positioning LUNAR below IP, we can drastically reduce undesired interactions with the IP layer and add address self-configuration and automatic gatewaying for the fixed Internet in an easy way. LUNAR also restricts itself from applying "smart optimizations" and piggybacking of information in order to keep the system simple and its behavior more predictable. We believe that simplicity of operation as well as the ease of implementation is the prime factor for pushing MANET style networking into the mainstream. First controlled experiments for small size networks showed that LUNAR is par with the best available ad hoc routing protocols!

The paper is organized as follows: After briefly introducing the functioning of major ad hoc routing protocol candidates we explain in Section 2.3 the rational behind LUNAR. Section 4 provides more details of the LUNAR protocol and its implementation for which in Section 5 we also report on a performance and code size comparison with OLSR.

2 Related Work and LUNAR's Stance

There already exist various Ad-Hoc routing protocols, in particular those created by the work done by the MANET group. In this section we describe DSR, AODV and OLSR (which we refer and compare to later on in this paper) before elaborating on the rationale that lead to LUNAR.

2.1 DSR, AODV and OLSR

DSR, as described in [3], is an on-demand routing protocol i.e., it constructs routes when it needs to do so. When a node running DSR needs to look up a route to another node it first looks in its route cache to see if it already has a valid route to that destination. In the case that the node does not have the path in its route cache it must perform route discovery. In order to do this, the node floods the network with `ROUTE_REQ` packets. Any node which has a route to the destination replies with a `ROUTE_REPLY` packet and affixes its own address to the header to facilitate source routing. The source node then uses the `ROUTE_REPLY` packets to construct the per-hop path to the destination which it then affixes to the packets that it sends out.

DSR uses aggressive caching and overheard routes in order to minimize the impact of the routing protocol on the network. The DSR nodes in the network operate in promiscuous mode. In this case a node can also overhear other nodes performing route request and reply procedures, providing an additional source of information about the network state without having to actually transmit any more routing packets.

AODV, as described in [2], is also an on-demand protocol. It is inspired by both the DSR protocol and by the DSDV [10] protocol. It uses the same on-demand flooding techniques for route discovery and route maintenance as DSR, but removes the source-routing overhead from DSR and replaces it with a Distance Vector technique for route building. This means that the nodes in the network exchange vectors with each other which reflect the distances measured between nodes.

OLSR is an optimized version of the classical Link State protocol. In an OLSR network there are certain nodes which are designated "Multi Point Relays" (MPRs),

these nodes exchange link information with each other. Instead of every node in the network being of equal importance, OLSR imposes a hierarchy where a subset of nodes in the network are designated MPRs. These MPR nodes are thus responsible for all the routing work in the network. The idea behind this is to cut down on routing traffic by only having a portion of nodes performing routing duties. As well as this, OLSR also only maintains information about a subset of its neighbours as opposed to retaining information about all neighbours like classical Link State protocols.

2.2 Status of available implementations

Since the approach throughout this paper is centered on running code, our group surveyed the available implementations of Ad-Hoc routing protocols.

TORA (Temporally-Ordered Routing Algorithm) [17] is among those ad hoc routing protocols available for Linux. However, the implementation would repeatedly crash the kernel which made testing and comparison infeasible. Also the available implementation is an in-kernel implementation for the Linux v2.2.x series which is now not the current stable version of Linux.

Although there exists a DSR implementation for Linux [11] (as well another one for FreeBSD [12]), we could not use it because it would not work for TCP connections, prohibiting testing with realistic applications like WEB browsing.

The picture looks better for AODV where two implementations are available: Mad-Hoc [15] and AODV-UU [14]. Mad-Hoc has some serious problems with the routing logic (reroutes can take half a minute) while AODV-UU works fine and served as a good comparison point for LUNAR.

Similarily, OLSR [13] is available in a stable implementation from INRIA and appears to work well, therefore it was also included in comparisons.

Overall, the availability of Ad-Hoc routing protocols is far from satisfying and ease of installation and configuration are other domains that usually are neglected too. Other Ad-hoc routing protocol implementations do exist aside from the ones reviewed above, but they are not available with an open-source license which means that for research purposes it makes life considerably more difficult. The aim with LUNAR was to address both the complexity of implementation as well as the configuration issues.

2.3 Goals and Assumptions in LUNAR

LUNAR aims at contributing to the area of ad hoc routing protocols in the following way:

(a) Have a working and robust implementation today. The code size should be small and the algorithms not have many subtleties.

(b) Simple operation and integration into existing IPv4 networking. Ideally, starting the LUNAR software should not require any parameters.

(c) Solving the problem for the common case. In a first place we want to address spontaneous networks formed by a small group of people.

(d) Enabling many common cases in parallel. Several ad hoc clouds should be able to coexist, enabling people to set up logically independent but physically overlapping networks.

Stipulating simple operations is a must for ad hoc networks, otherwise we do not believe that ad hoc networking will cross the critical point beyond which one can assume that everybody in a meeting is ad-hoc enabled. Although a proposed ad hoc routing protocol implementation may be well-written it can be hard to configure and may contain many subtle tuning parameters. A classic argument is that “protocol X could be adapted to also cover this or that scenario”. How this shall be done is then left

unspecified and unimplemented. Instead of aiming at the broadest possible coverage of use cases, LUNAR concentrates on a setting what we consider to be a typical scenario: We assume spontaneous groups (e.g., meeting room, airport lounge) with rather small number of nodes (5 to 15) forming in close vicinity such that 3 hops already are an exception.

2.4 Ad-hoc Horizon

A major motivation for a more modest ad hoc routing goal was the observation that 3 hops is already pushing the limits in many ways. We believe that there is an *ad-hoc horizon* beyond which it becomes uneconomic 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 at their limits, resulting in a highly fluctuating connectivity space: slight position changes or objects getting in the way drastically change the neighbour 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 as well as they introduce the need to buffer packets and create subsequent packet reordering problems. After exactly how many hops we hit the ad-hoc horizon is dependent on the technology at hand as well as the assumptions one makes on the stability of the network topology. Based on our experience we think that a value of 3 hops makes sense. Section 4.5 gives some additional rationale for these dimensioning of LUNAR parameters and shows the calculations performed to reach these parameters.

3 LUNAR Architecture

LUNAR comprises several technology pieces and policies. We first introduce LUNAR's underlay network abstraction called SelNet and its interaction with the IP layer via ARP. Configuration issues (IP addresses and gatewaying) are discussed next.

3.1 Underlay Routing and Address Resolution

Figure 5 shows the position of the LUNAR network with respect to the traditional IP stack. The design of LUNAR is based upon the architecture of the SelNet underlay network [5] which is a layer 2.5 routing system designed to support a wide range of data forwarding and routing styles. The main idea is to link ad-hoc path establishment to the usual address resolution activities going on at the IP–subnet layer border i.e., ARP. Historically, DSR also followed this approach by doing multihop ARP [6]. Typically ARP is confined to broadcasting only to the subnet that it 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.

3.1.1 From ARP to XRP: Types and Parameters

The SelNet underlay network functions by trapping all IP data and control traffic and translating it into an intermediate representation which can then be manipulated by the nodes in the SelNet network. With SelNet all ARP traffic is trapped and then rewritten to the XRP (eXtensible Resolution Protocol) command set which permits a much richer set of expressions than standard ARP. When the IP stack issues an ARP request for an IP number's Ethernet address, SelNet translates this to an XRP resolution request which is then broadcast to all nodes in the Ethernet subnet. Re-broadcasting

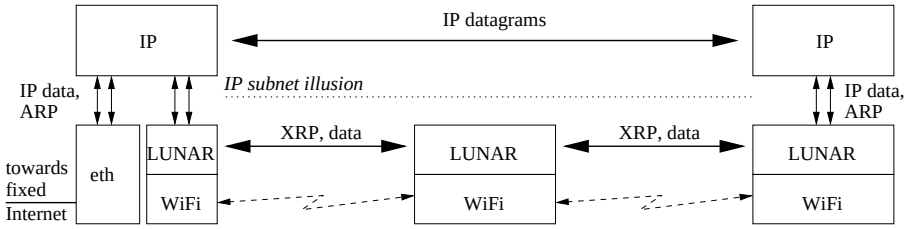

Figure 1: LUNAR as an underlay to the IP layer

logic is then responsible for getting the required resolution information across multiple hops, eventually creating state in the network for forwarding and possible rewriting of Ethernet addresses on the fly.

The XRP protocol messages are similar to the ARP "request-reply" messages: we ask other nodes to perform tasks for us according to our specification. Typically an ARP request can be thought of as a resolution function with one argument: IP Number, which returns a result i.e., an Ethernet address. With XRP we introduce the ability to define which types (in the Programming Languages sense) the argument and the result should have. We are not restricted to just resolving IP Number to Ethernet address other pair combinations are possible. As well as this we add more parameters to the (now generic) resolution function, for example: over how many hops the resolution query should be re-broadcast and where the result should be returned to.

3.1.2 SelNet – an Underlay for IP

Besides the rewriting of ARP control traffic, SelNet also rewrites the headers of the data traffic in the network: Each delivery path will have its own forwarding state, which at the packet header level translates into a set of selectors to identify this state. A data packet will have a different selector on each leg of its path because the SelNet network will rewrite the combination of Ethernet and selector on each hop, in the same way as IP forwarding rewrites the Ethernet address for each subnet crossed. For the selector header SelNet uses the Simple Active Packet Format (SAPF [7]) for the data traffic in the network. Since an additional layer has to be introduced between IP and Ethernet in order to allow SelNet to function, we wished to keep additional complexity to a minimum. SAPF packets can be forwarded at high speeds due to their extremely simple header format – up to 30% faster than packets with more complex header formats [19].

Underlaying IP has several advantages: we can create an virtual network at the subnet layer, so that we can have separate address spaces in parallel thus each ad-hoc LUNAR cloud is represented as an independent IP subnet. This allows physically overlapping subnets to be logically isolated. By fooling IP in this manner to represent multihop paths as single hop paths, we disassociate the complexity of the ad-hoc routing problem from the IP stack. Unlike traditional approaches to ad-hoc networking which seek to make ad-hoc routing "normal" for the IP stack, we attempt to let IP keep its assumptions about the underlying network, whilst still injecting new functionality and routing styles into the network.

3.2 LUNAR Self-Configuration with Profiles

Joining a wireless ad hoc network involves a multitude of parameters. We focused on IEEE 802.11 where the following items have to be defined: frequency/channel used,

speed, Ad-hoc mode, WiFi network name, IP address, IP gateway and DNS information. We propose to define "profiles" or parameter bundles which fix some of the set of parameters and provide methods of automatic resolution for *all* remaining other parameters. A default profile, for example: "profile red", bundles all parameters and procedures required to ensure the zero-configuration start of LUNAR.

In case people wanted to set up a parallel, independent Ad-hoc network, profiles come in handy again. Users would have to start LUNAR with the name of some predefined profile they agree on e.g., "profile blue", which would create a logically independent Ad-hoc network. This is also useful for scaling purposes when some frequency channels are overcrowded and users want to shift to different frequencies or link speeds.

In the remainder of this section we look at those parameters that need to be settled at run and for which LUNAR provides self-configuration logic.

3.2.1 IP Address Allocation

LUNAR allocates the private C class 192.168.42.0 to the default profile which also comprises settings for the wireless card. The node's host identifier will be dynamically determined in a Appletalk like fashion by picking a random number from the profile's address range and then probing for collisions: for this we use the XRP query procedure introduced above and if none of three resolution queries return successfully the node will use this address. Note that because XRP operates with SelNet selectors, there is no need for any transient IP address assignment during the configuration phase (see e.g. [18] for the need of transient IP addresses in an ad hoc network).

3.2.2 IP Gatewaying

Attaching a LUNAR cloud to the Internet can be achieved without explicit nomination of a default gateway. Any node that has a default gateway entry in its IP routing table before launching LUNAR (which we assume means that this node knows how to route to the Internet), will automatically switch on the LUNAR gatewaying mode. This means that the node will claim successful resolution for each resolution query of an IP number that is *not* in the 192.168.42.0 subnet – it becomes an *implicit* gateway. Any other node that tries to reach e.g., 234.5.6.7 will receive delivery path information leading to the implicit gateway node. The implicit gatewaying node then simply forwards subsequent IP packets its IP module, which in turn will forward them via NAT to the real default gateway in the fixed Internet.

A problem with this approach arises when several nodes perform gatewaying and packets can be sent to the Internet via different gateways: the NAT state will only be created on the first gateway picked, which might change when LUNAR resolves delivery paths differently in the future. The method we chose to pin down gateways involves DHCP, which we use only for disseminating gateway and DNS information and not for node address assignment. To this end, a node joining the LUNAR cloud performs the following steps:

- Assign itself an IP address
- Ask for gateway and DNS data (via DHCP_INFORM)
- If available: add the corresponding routing entry and stick to it
- Otherwise: fall back to "implicit gatewaying"

Complementary, a node acting as a gateway should start a DHCP server on its wireless interface, but is not required to do so.

4 LUNAR Protocol Details and Implementation

After a more detailed account of our ad hoc routing protocol we give additional details concerning the implementation in terms of the SelNet underlay layer, the XRP protocol and the sofwtare architecture.

4.1 LUNAR Operations

LUNAR creates *individual* unicast delivery paths for each source/destination pair. This reduces the complexity of shared delivery path information and keeps paths under the full responsibility of the originator.

Paths are discovered by a classical flooding mechanism, where the current version of LUNAR relies on the existence of bi-directional links in order to return forwarding information. First, a one-hop broadcast search is performed: if the destination is a direct neighbour, it will unicast the details on how it can be reached. These addressing details are unique for the given source/destination pair i.e., data delivery is based on LUNAR internal selectors rather than IP addresses. If the one-hop search fails, LUNAR immediately tries the maximum diameter search: neighbour nodes are requested to forward a search request and, if successful, they reply with a unicast with the same type of addressing details as in the one-hop case. An earlier version of LUNAR used an incremental ring search to discover nodes in the network, however we discovered that due to the three second ageing of paths this technique did not perform as well as immediately trying the maximum diameter search. The source node ages the information obtained (i.e., cleans the ARP cache entry) such that after 3 seconds it redoes the path discovery procedure. Note that the established delivery paths are uni-directional – the destination will launch its own path discovery if it has to send reply packets at the IP level.

Once a path is discovered, it is maintained in the following way:

1. After three seconds, the ARP cache times out and IP has to re-ask for the IP-to-Ethernet mapping
2. SelNet intercepts this and establishes a *second* path (the old one is still up, but IP will not use it anymore because it has an ARP resolution request pending)
3. The second path is established and IP starts using it
4. The first path dies away silently and unused and is garbage collected by SelNet

This form of "path maintenance" removes the need for discovery of link breaks and subsequent route repair, as a next path setup phase is scheduled already in advance. Data packets are not buffered, leaving the recovery of packet losses due to link breaks to the transport layer.

IP broadcast communications is implemented in a slightly different way. The source initiates the creation of a private delivery tree that will be used for all subsequent broadcast packets during the next 3 seconds. To this end, it broadcasts a setup message that will be rebroadcast by neighbours. Neighbours reply with a unicast message that they are interested in joining the tree. If only one neighbour replies, the tree will be extended by a unicast leg; if more than one neighbor is interested, we will use a broadcast for data delivery. The source starts using the delivery tree after some fixed delay, without waiting for any setup confirmation messages. If still more data has to be broadcast after 3 seconds, a new broadcast tree is built.

4.2 The Packet Demultiplexing Sublayer

The L2.5 underlay network of SelNet, on which LUNAR is based, delivers packets based on "selector" values that are independent of the IP addressing. All packets carry a selector that identifies the context (state) which will handle their further processing. We use the Simple Active Packet Format (SAPF) which uses 64 bit selectors for switching datagrams. These selectors typically are dynamically assigned at run time and are only valid for one hop. SelNet's L2.5 routing system translates into using a separate Ethernet type value for SAPF (currently we use the value 0x4242) and the insertion of a 64 bit selector field between the Ethernet header and the layer 3 protocol data.

When sending a payload together with the selector to the next SAPF node, the selector is used to lookup forwarding data linked to this selector and, after possibly rewriting the selector field for the next hop, the packet is sent out to the address specified in the forwarding data. The SAPF engine also permits to send packets into the local node's IP stack or call other packet handler routines. The packet handler routines provide the hook into the XRP module and are used to implement the LUNAR rebroadcasting logic as well as callback handlers for reply messages.

Forwarding state, as well as reply callbacks established on request from a neighbour, are subject to garbage collection by the SAPF engine. This automatically eliminates old routing state on top of the originator stopping the usage of a delivery path after 3 seconds.

4.3 XRP – the eXtensible Resolution Protocol

XRP is our generalized query and steering interface which LUNAR uses as a method of controlling the behaviour of transit nodes in the network. It is implemented as a set of request commands and reply messages. The basic model is "fire-and-forget": For those requests where we need some return information, the sender itself has to take care about establishing the reply channel, and to do retransmissions in case a confirmed request is needed. XRP commands travel via the SAPF underlay: a special well-known selector value is used to address XRP commands to a neighbour's LUNAR software.

The main XRP function is the "resolve" command: an application or a neighbour node can ask for name or address resolution and will get back an selector value that "stands for" the requested destination and which can be used to send data to it. XRP resolution commands have the following main parameters:

- **scheme** : the address family in which resolution should take place. Examples are: IPv4-over-ethernet, SAPF-over-ethernet, potentially also URL names (this was not implemented, though).
- **id** : the name to resolve. Examples are an IPv4 address, a SAPFselector/ EthernetAddress pair, or a URL.
- **resolution style** : in which form the resolution result should be delivered. Possible choices are a SAPF selector, or the actual address bits.
- **target** : at which selector value a given name should be made accessible after resolution, or to which selector value possible results should be sent back, depending on the chosen resolution style.
- **resolution depth** : controls whether requests should be forwarded in case a node can not resolve the given name and is decremented at each hop.
- **series** : a selector value for identifying the request.

Typically one would send two (or more) XRP commands in one message: the first one requests the installation of forwarding data pointing back to the originator, the second request is the actual query whose results will be sent back over the previously established reply path. For example, the one-hop discovery works by broadcasting the following two XRP commands in a single Ethernet frame:

```
resolve(scheme=sapf/eth,id=MYSEL/MYETH,style=selector,target=ABC);
resolve(scheme=ipv4,id=IPNUMBER,style=sapf/eth,target=ABC);
```

Because of the broadcast, all receiving node will install a back pointer to the originator at the address 'MYSEL/MYETH' (first command): the forwarding function will be made accessible under the given selector value 'ABC'. The second command requests the resolution of an IPv4 address. In case of a successful resolution (i.e., a node identifies itself with IPNUMBER) we request the result to be of the form 'selector/ethernet address' and this result being delivered via the 'ABC' selector. Once the originator receives a reply for the second XRP command, it can use this sapf/eth address pair to send packets to the IPNUMBER host.

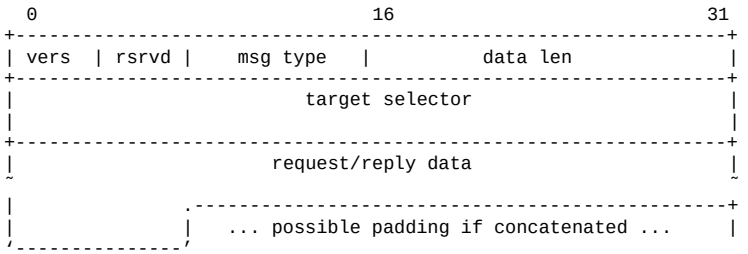

Figure 2: XRP Packet

Figure 2 shows the XRP packet format for requests and replies. Note that multiple XRP commands can be put back-to-back in the same underlying datagram: the commands will then be executed sequentially.

4.4 Using XRP for ARP Forwarding and Broadcast Dampening

The example of a single-hop resolution query above is a simplified one that would not work well in the multi-hop case. When resolution requests are re-broadcast it is important to suppress the rebroadcasting of a received message if another instance of the same query already visited a node. An important XRP command in this context is the "jump-on-existence" (JEX) request which has one selector argument. It allows to make SAPF delivery conditional to the existence (or absence) of a SAPF handler. If there is a valid handler for the given selector parameter, the complete packet will continue execution at this place. Otherwise the XRP module will continue to work on the following XRP command in the packet. We use this function to "divert" duplicate packets into a dead end such that they are not processed further.

The following pseudo code is a full example of how the multi-hop resolution query is written down as a sequence of XRP commands. Uppercase identifiers are used to show the variables that must be provided by the originator (as well as by rebroadcasting nodes).

```
jumponexistence(REQUESTID);
resolve(scheme=sapf,id=REQUESTID,style=selector,target=0);
resolve(scheme=sapf/eth,id=MYSEL/MYETH,style=selector,target=TMP);
resolve(scheme=ipv4,id=IPNUMBER,style=sapf/eth,
        target=TMP,series=REQUESTID,depth=2);
```

We start by examining the second XRP command which creates a packet redirection: The resolution command lets the system resolve the SAPF selector REQUESTID to the target 0, which by definition is the garbage bin and corresponds to the UNIX /dev/null device. Hence, future packets sent to REQUESTID will be discarded. The first command of our query requests this packet to be delivered to the REQUESTID handler, should it exist. This means that only the first instance of a query packet arriving at some node has a chance to be executed beyond the first command.

The third XRP command creates forwarding state pointing back to the querying node that becomes addressable via the selector TMP. This delivery path will be used for returning the result of the resolution request in line 4.

The XRP command in line 4 is the "real" ARP query that requests an IPNUMBER to be resolved and corresponding addressing information to be returned via the TMP selector. The querying node has installed a packet handler at selector MYSEL that waits for the result.

If the resolution request on line four fails, the XRP module will re-issue a full resolution query by itself. First, the "depth" parameter is consulted and if found to be bigger than 0 it is decremented and used for the outgoing query. The "series" field provides the same query identifier that the incoming query had and enables the broadcast dampening. Finally a new TMP' selector is randomly generated. Taking all these values the intermediate node is capable of forwarding the modified resolution request. Before sending the new query however, the node installs a reply handler: the first incoming resolution reply will trigger the setting up of a forwarding pointer to the found destination at some randomly generated selector DST, and this selector DST being sent back along the chain to the originator.

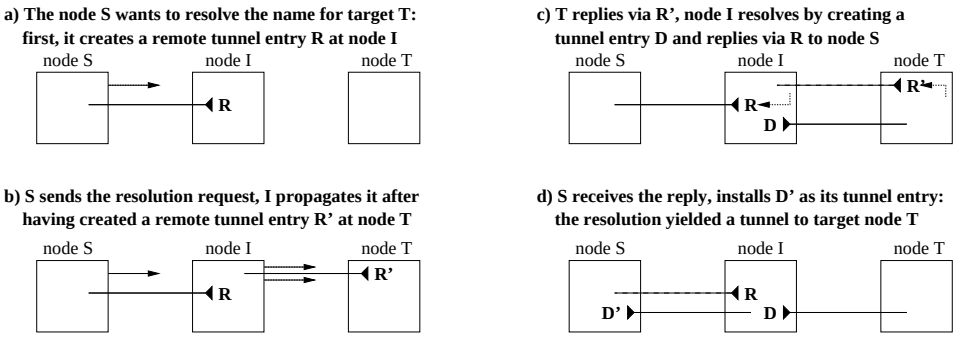

Figure 3: Example of a multihop-ARP in LUNAR

Figure 3 shows the sequence of a two-hop query and the installation of forwarding pointers as the query result travels back to the originator.

4.5 LUNAR Parameter Tuning

With LUNAR we tried to identify the sweet spot where Ad-hoc networking has a sufficiently good chance to work although we keep the algorithmic complexity low. The 3-hop limit, for example, mirrors the discussion on the ad hoc horizon in section 2.4 (note that a user can override this limit if desired, although it is not recommended). Two other LUNAR parameters are the 3-second timeout for delivery paths and the expected number of nodes that should be able to share a LUNAR cloud. In this section we briefly explain how these parameter values were adjusted.

Agressively reestablishing paths after 3 seconds makes sense because (a) a link break is fixed on average after 1.5 seconds (link breaks are spread over this 3 second interval), (b) 1.5 seconds is not an issue with WEB browsing applications, (c) other ad hoc routing protocols based on hello messages, which for overhead reasons are forced to keep the hello message frequency to roughly 1 second, will show similar times to react to link breaks.

Second, to get an estimate on the overhead of the LUNAR protocol we computed the number of control messages generated. Assuming a network with 12 nodes, half of them actively communicating with two others in the cloud, we need to maintain 12 delivery paths in forward and 12 delivery paths in reverse direction. As these 24 paths are refreshed every 3 seconds, we have 8 path discovery requests every second. If all nodes sit in the same collision domain we will have 1 broadcast, 11 rebroadcasts and 1 reply unicast per route discovery request, hence roughly 100 messages per second. Assuming an average size of 100 Bytes per control message at 2 Mbps, this gives a bandwidth requirement of 80 kbps. Even if this overhead doubles or triples, the bandwidth overhead obviously is not the limiting factor and the 6 users sharing the 2 Mbps already are down at a 300 kbps share (or approx. 1 Mbps for the 11 Mbps WiFi case) and might consider switching to an independend ad hoc cloud with another frequency.

4.6 Software Architecture

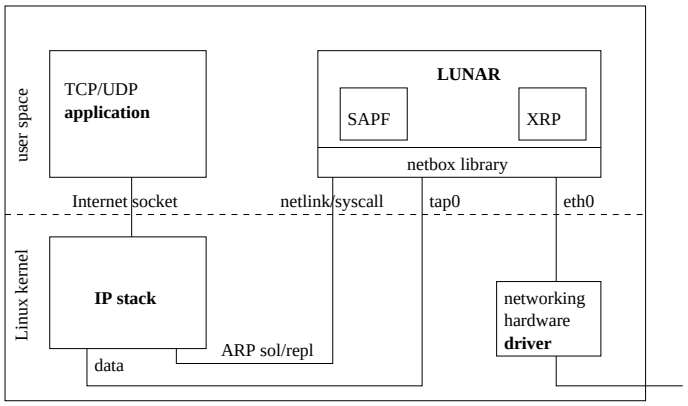

Figure 4: LUNAR node architecture

Figure 4 shows the software decomposition of LUNAR. Currently LUNAR is implemented 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. ARP entries are created via an ioctl system call. A code line count shows that the LUNAR system is rather small:

# of C lines	Module	Content
550	lunar.c	main() program and self-config logic
420	netbox.c	OS glue and timer support
330	sapf.c	SAPF engine
500	xrp.c	XRP and resolution forwarding logic
160	xrp_pkt.c	XRP encoding/parsing
1960	Total	

5 Performance Comparison with OLSR and AODV

LUNAR underwent many cycles of stress testing, ranging from simple pings and running multiple TCP and UDP based applications like http, ssh and nmap as well as parallel flood pinging and broadcast pings, in one-hop and in multiple-hop settings. LUNAR proved to behave very robust and seems to let nodes share the wireless Ethernet resources in a fair way, response times are also quite predictable. What was interesting though was to confront LUNAR with other ad hoc routing protocols. As mentioned in section 2.2, AODV and OLSR were the only protocols running under Linux which were usable for a comparison.

After first comparison runs done by the authors, Henrik Lundgren and Erik Nordström (who wrote the AODV implementation used in the tests below) used the APE testbed [9] and confronted the three protocols with three different test applications that ran in a single mobility scenario.

- The mobility scenario consists of three stationary nodes, one being a gateway serving documents from the internet, and one mobile node communicating with the gateway from various attachment points, resulting in routes ranging from 1 to 3 hops over a time interval of up to 5.5 minutes, including settlement time at the beginning, at intermediate positions and at the end.

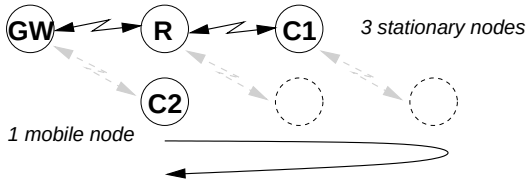

Figure 5: A simple “wireless ad hoc acccess network” mobility scenario.

- The test applications were:
  – WEB access (30 kBytes every 8 seconds)
  – ping (once a second, 64 Bytes)
  – MP3 streaming (continuous 128 kbps)

The 8 second wait time for the WEB access matches the attention span of typical WEB users []. Full results with more extensive tests will be published in a forthcoming report []. Here we restrict ourselves to present the overall outcome of the small set of tests as introduced above. The results were averaged over the outcome of several test runs.

Test app / Protocol	AODV	OSLR	LUNAR
Successful WEB fetch cycles	24	33	35
Successful pings	88%	93%	93%
MP3 streaming, data received	5.2 MB	4.9 MB	5.2MB

LUNAR was the best performing protocol for the WEB access application (35 access in the 5 minute interval) and matched the other top performing protocols in the ping and MP3 streaming application. A subjective assessment of the MP3 test furthermore matches the shown figures were AODV and LUNAR sometimes had almost inaudible switchovers (although it could take up to 2 or 3 seconds), were OLSR could take up to 10 seconds to get routing paths correct again.

Overall, we are very satisfied with the outcome as LUNAR matched or outperformed protocols which have been proposed since several years now. Another comparison, this time based on code size, reveals a similar picture. Although code size is not a measurment for the fitness of a protocol for the task at hand, it nevertheless shows that some algorithmic complexity might not be worth the specification and implementation price.

A code line count shows that LUNAR compares very favorably with relation to other MANET ad hoc routing protocol implementations (please see bibliography for links to implementation code):

Name	*.c	*.h	total
OLSR	3617	1025	4642
AODV-UU	2330	467	2797
DSR	1568	391	1959
LUNAR	1430	100	1530

These numbers were computed in the following way:

- *.c: all comment lines removed using ”gcc -E” after having commented out any header file includes (system and user), empty lines removed
- *.h: all comment lines removed, empty lines removed

Summing up, we think that LUNAR reaches its goals well by serving the common case of small diameter ad hoc networks with a simple and robust ad hoc routing protocol.

6 Conclusions

We have introduced the LUNAR ad-hoc routing environment which takes an extremely lightweight approach. It works by trapping ARP requests and re-writing them to an intermediate representation known as XRP (eXtensible Resolution Protocol). These XRP requests are then rebroadcast by transit nodes to reach nodes which are outside of the originating node’s subnet and result in the setting up of multihop data delivery paths. LUNAR is an underlay network i.e., it sits below IP and above the wireless link layer - this positioning of functionality allows us to control what the IP stack believes about the network. This allows a much more natural introduction of Ad-Hoc routing functionality into the network. LUNAR also exhibits self-configuring behaviour: it

can perform automatic address assignment in a decentralized environment i.e., without a central server, LUNAR also performs automatic Internet gatewaying.

The LUNAR environment is designed to target the common-case of network clouds with 10-15 nodes and a diameter of up to three hops. Through empirical tests we have shown that LUNAR performs comparably with established MANET ad-Hoc routing protocol, despite its simple approach and small code size that includes full self-configuring and IP broadcast support. For future releases of LUNAR we consider support for uni-directional links and plan to have a more streamlined representation layer for the XRP sub-protocol. A port of a stripped down LUNAR version to embedded devices is also under way.

Acknowledgements

The authors would wish to thank Erik Nordström and Henrik Lundgren for their invaluable work. This work was done at the University of Basel, Switzerland. We gratefully acknowledge the support of the University Computing Centre (URZ).

The LUNAR Web Page

`http://www.docs.uu.se/selnet/lunar/`

References

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

[2] 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`.

[3] 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`.

[4] 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`.

[5] Christian Tschudin and Richard Gold. SelNet: A Translating Underlay Network. Submitted for publication, `http://www.docs.uu.se/selnet/selnet.pdf`, 2001.

[6] 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.

[7] Christian Tschudin and Dan Decasper. Active Networks Request for Comment: Simple Active Packet Format (SAPF), 1998. `http://www.docs.uu.se/~tschudin/pub/cft-1998-sapf.txt`.

[8] Ralph Droms. Internet draft: Dynamic host configuration protocol, 1997. `http://www.ietf.org/rfc/rfc2131.txt`.

[9] Henrik Lundgren, David Lundberg, Johan Nielsen, Erik Nordström, and Christian Tschudin. A large-scale testbed for reproducible ad hoc protocol evaluations. In *WCNC2002*, 2002.

[10] Charles Perkins and Pravin Bhagwat Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications.

[11] Yih-Chun Hu. DSR implementation for Linux. URL has since been removed.

[12] Carnegie Mellon University. DSR implementation for FreeBSD. `http://www.monarch.cs.cmu.edu/dsr-impl.html`.

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

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

[15] Fredrik Lilieblad and Oskar Mattsson and Petra Nylund and Dan Ouchterlony and Anders Roxenhag. Mad-Hoc: AODV implementation. `http://mad-hoc.flyinglinux.net/`.

[16] Christian Tschudin and Richard Gold. Lightweight Underlay Network Ad-Hoc Routing implementation. `http://www.docs.uu.se/selnet/lunar`.

[17] University of Maryland, Ad-Hoc Networking Research Group. TORA implementation. `http://www.cshcn.umd.edu/tora.shtml`.

[18] Charles Perkins and Elisabeth Royer and Samir Das IP Address Autoconfiguration for Ad Hoc Networks. `http://www.6ants.net/doc/draft/draft-perkins-manet-autoconf-00.txt`

[19] Tilman Wolf and Dan Decasper and Christian Tschudin Tags for high-performance Active Networking. In *Openarch 2000*. `http://www.docs.uu.se/~tschudin/pub/cft-2000-tan.pdf`.