

THE MANIAC CHALLENGE

Live and Let Live 09

Ivan Klimek

*Computer Networks Laboratory,
Department of Computers and Informatics, University of
Košice, Letná 9, 041 20 Košice,
Slovak Republic
E-mail: Ivan.Klimek@cnl.tuke.sk*

Tomas Korenko

*Faculty of Mathematics and Physics,
Charles University in Prague,
Ke Karlovu 3, 121 16 Praha 2,
Czech Republic
E-mail: Korenko.Tomas@gmail.com*

Introduction

Mobile Ad hoc Networks are greatly limited by current technological possibilities, concerns like battery power or packet loss etc. limit their form and possible usage from all sides. If we would try to generalize MANETs to better understand them, we could say that they mostly form few-hops networks around uplinks. While designing the presented solution we used the mentioned, as a definition of "usual" MANET topology.

Every approach needs to have its priorities set, our could be defined as: maximal power saving for nodes running on battery without sacrificing end-to-end connectivity and the ability of network to grow. The question standing is how to manage that other nodes data, which in fact are uninteresting data for us, don't flow through us if there is any other possibility for them to go? Of course when designing strategy that affects end-user performance in a way that tries to be fair, there will be always the risk of someone trying to get more for himself. The best protection is to don't need any protection at all, make something so simple that it cannot be even hacked. What does that mean in this context? Active hiding, we have only one way to the network, so we are a network "dead-end". If every node running on battery would behave like this, the network would not be Ad hoc at all, as all the nodes would be able to see and connect only to the nodes which are not running at battery – uplinks - Access points. That's why we need to implement a cooperation mechanism. If we (local node) detect a node that has no other means of connection to the network than through us, we provide the uplink. This way the network is able to grow but the maximum traffic is directed through the nodes without battery concerns, the battery driven nodes forward only the completely crucial minimum traffic for the network to keep its basic function, end-to-end connectivity. The network will reach the state known as Strong Pareto Optimum (SPO), where no node can take any action/change without "harming" any other

node. This idea is not new, we came with it on the MANIAC Challenge 07, but it was not tuned enough nor completed and its implementation was not satisfactory too. We could say that it was just the first experimental run of Live and Let Live (LLL) in real life environment. The current LLL shares with the original version just the base idea. For the better understanding we divided it into 3 main components: Topology Minimization aka Active Hiding, Optimal Next-Hop discovery/selection and Garbage collection aka disconnecting nodes with other uplink possibilities.

1. Topology Minimization - Active Hiding

Immediately after the node goes UP, it sends OLSR HELLO messages, this messages contain its originator IP address. Every other node which is in range hears this hello and if LLL is running on it, it will using the routing table check if a node with that IP address is already in the network, if yes it continues to hide to it, if no, it adds that node to its list of neighbors and provides him the uplink. The hiding works using unicasting all traffic on Layer 2, that means the local node's HELLO messages are send only to preselected nodes so only nodes we select know of us and send data through us.

2. Optimal Next-Hop Selection/Discovery

Because the local node does not limit or control the incoming traffic in any way, the node has full routing table information and according to that it selects its next-hop. However in a network in which every node would be running LLL, all nodes would be hiding itself as much as they can, the information in the routing table would not need to be absolutely relevant and actions based on them would not need to be optimal. Let's imagine a situation in which our selected next-hop is for whatever reason not able/willing to forward our data in a performance level we desire. The level of ability/willingness to forward our data

is checked by passively listening for our data being forwarded by our next-hop. Even if the next-hop is forwarding for example only 1 percent of our traffic, will we risk there is no one else better and disconnect from him? Is not that 1 percent still better than nothing? That's why it is important to know if there is another possibility at all. LLL hiding happens at Layer 3, so if we would be able to detect nodes at lower layers it would be a way to go. Some drivers (Atheros mostly) build their statistical information not per-SSID as most drivers do, but per-neighbor, that enables us to check if there is someone else around with sufficient signal quality/strength level. This gives us exactly the information we need to know to disconnect from the uncooperative next-hop and become a node without any mean of connection to the network. All LLL nodes around (that nodes we detected that are there, but are hiding) will see our HELLO packets and check against their routing tables, in a few seconds after the information that we are down propagates through the network, they won't be able to detect our IP in the routing table and will start to send us their HELLOs. That will build up our routing table and according to the destination we want to communicate with we will select the new next-hop. It is not clever to make such next-hop switching blindly, because the propagation of network information isn't instant and we don't even know if there is anything better out there. But if we suppose that our new next-hop will have better performance (based on detection on L1/L2) then few seconds without connectivity will be definitely worth it.

3. Garbage Collector

Until now we dealt only with adding nodes to which we will provide uplink resp. selecting/changing next-hop which we use to forward our data. Now we will look at removing the nodes that have some other means of connectivity to the network than us. LLL is written in a way that every node keeps only one the currently best next-hop, in a dream world without hackers this would be sufficient. In the real world it would be only a question of time when someone would exploit this to get more of the network by using more than one next-hop as uplink which would not be fair for other network nodes. OLSR implements the concept of MPRs, to every two and more hop neighbor the data needs to go through the MPR. MPR propagate so called Topology Control (TC) messages, which contain the links it knows. So if we are able to detect the IP address of a node, that we are providing uplink to, in the TC messages, we can remove it from our neighbor list as it already has some other mean of connection to the network. This check mechanism could be implemented much easier if we would have an interface to the Topology table inside of the routing

protocol. Just simply check for a duplicate record of the IP addresses we are interested in. For example OLSR-NG has a plug-in interface to accommodate such needs, for real life LLL deployment this would be a solution. Other possible solution is parsing the debug output of NRL olsrd, which on debug level 2 prints out the topology table information.

4. Implementation

Besides the MANIAC API we are also using LIBPCAP because it is providing greater flexibility through the capture filters than the support of promiscuous mode included in the API. Of course we are capable of doing so without LIBPCAP only with MANIAC API, but it wouldn't be so effective without these filters. All outgoing traffic goes through MANIAC API, where we are pushing it through the ChangeNextHop function to unicast all of it. Right now we are fighting the packet duplication, because we don't support multicasts we have to resend/unicast the HELLOs to all nodes we want to be "visible" to, we have some problems with the CopyPacket API function so we are doing it via reducing the HELLO timeout appropriately to the number of the nodes and then forwarding it using a semaphore logic at the MANIAC API level. Incoming traffic is not being modified in any way, so the local node receives all the echo messages from other nodes, that together with the ARP at Layer 2 represents a possible source of problems so that is another reason why we are pushing all data using ChangeNextHop.

Summary

We believe that Live and Let Live is the optimal solution of interoperability and cooperation while minimizing the amount of uninteresting data forwarded by nodes running on batteries. The presented solution should be completely bullet-proof and more than just an experiment, we personally consider it as a release candidate. While working with MANETs we found several other problems that are not directly linked with interoperability and cooperation but greatly affect the efficiency and power consumption, we are presenting some possible solution in the appendixes of this paper. We hope we will be able to test the LLL 09 at the this year's MANIAC Challenge as it is an unique experiment giving us a real life MANET testbed full of nodes with different cooperation levels that we don't know how will behave, something like that cannot be simulated.

Appendix 1: TXPOWER optimization

During our experiments we noticed the inefficiency and wasting of default and automatic selection of TXPOWER - output power. As laws of physics clearly say that the energy is proportional to the square of the radius, we thought it may be possible to save some energy on "intelligent" TXPOWER selection resp. adding the radius to the routing metric. Let's look for example at a commodity WiFi module as WNC CM9 AR5213 MiniPCI, its maximal ERP is 18dBm, that's 63mW with total power consumption of 430mW. So the maximal output power is only about 1/7 of the whole power consumption needed for the transmission. We experimentally tried if setting TXPOWER on its minimal level resp. maximal level will produce a measurable difference in the power consumption. We removed the battery of the test notebook, and attached it to a Wattmeter. The difference between TXPOWER set on 1dBm and 18dBm was so small that we were not able to detect it on the used apparatus. The difference between transmission and no-transmission was about 1 to 1.5W¹. From this fact we deduce that it is not possible to save energy directly on minimizing TXPOWER, with for example selectively choosing of only the nearest next-hops and setting the output power to the lowest possible levels. However indirect effects of doing that are interesting. One of the biggest problems of wireless communication is interference. By minimizing the TXPOWER to lowest possible levels to keep the network connection usable the radius to which the signal can cause interference is massively reduced. By reducing the interference we reduce the packet collision rates and that saves energy as fewer packets need to be retransmitted. Currently we are experimenting with an automated system of manual TXPOWER setting based on the signal strength/quality levels with experimentally determined boundaries.

Appendix 2: Cognitive Radio Emulation

The next issue that would acutely need optimizing is the inefficient spectrum usage. Most devices work their whole lifespan on the default settings, and therefore on the default channel. In the infrastructure mode the channel is selected on the AP, some APs have automatic channel selection based on scanning all channels and choosing the least polluted one. Clients then associate to the AP according to the SSID, they switch to the channel used by the AP. In Ad hoc mode the situation is a bit worse,

¹ The difference does not consist only of the WiFi adapter power consumption, the rest is created by the data transmission overhead.

because the default behavior, choosing the channel on which we hear the SSID places most if not all nodes on the same channel. This massively reduces the network throughput and increases the number of collisions. We experimentally determined that this automatic behavior happens on all the adapters we have tested². We propose an experimental solution based on the usage of two adapters connected using a virtual interface into a single Layer 3 entity. The interface A stays at the "majority" channel, the interface B is by default set on the "opposite" channel e.g. if interface A is at channel 11 the interface B will be at channel 1. Interface B actively scans all channels and if it detects a node on other than the majority channel and it doesn't need to be the currently set channel, it will connect to it. Using this method we are able to avoid congested channels and spread the spectrum usage. We could say we are transforming the topology from 2D to 3D.

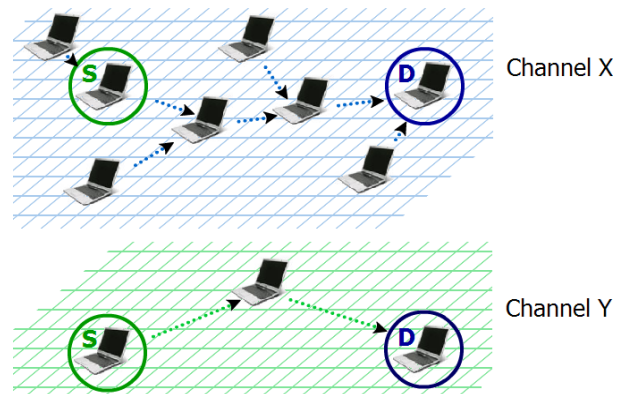


Figure: 3D Topology example, channel Y is more free so it is used for communication.

From the implementation point of view, the biggest problem was with the default driver behavior to switch to whatever channel it hears the set SSID on, even if the channel was manually set before to. This could be solved by modifying the driver itself, but to save time we chose a different approach with the same effect. We send IOCTL commands to the driver to stay at the selected channel in regular intervals shorter than the automatic channel switch intervals that have been determined experimentally.

² Tests were done on five different adapters, both on Windows and Linux.