Sensor Network as Internet Gateway

Anupam Ashish
MSc Software Systems Engineering
RWTH Aachen
anupam.ashish@rwth-aachen.de

Oana Comanici
Oana Comanici
MSc Media Informatics
RWTH Aachen
oana.comanici@rwth-aachen.de

Ignacio Martin Avellino Martinez
Ignacio Martin Avellino Martinez
MSc Media Informatics
RWTH Aachen
ignacio.avellino@rwth-aachen.de

ABSTRACT
The aim of this paper is to research and investigate the network characteristics of a sensor network deployed to provide Internet access. In the past, work was done to provide Internet access using a client and gateway. However, our contribution was to add the routing functionality to enable multi-hop topology and also make it robust to a changing topology. Both of them also helped to enable roaming. Interesting characteristics of such a network were discovered during this process. The maximum throughput achieved by this network came to be around 170-180 kbps which is inline to maximum bitrates of TelosB motes. Also, in accordance of the fact packet loss and jitter increased with the increase in multi-hops. We also conducted a load time test by accessing a web page which led us to further investigate at very low levels like changing packet size. This paper also summarizes these facts about network characteristics and tries to find out the correlation between them.

Keywords
Wireless sensor networks, TelosB, Routing, IPv6.

1. INTRODUCTION
Today's IPv6 networks are not intended for extremely mobile devices. These devices are often only equipped with limited power supply and transmission range as discussed in [3] and [4] for multi-hop networks. Transmission range and power supply are the two major limitations for dynamic networks where nodes change their topology or availability continuously. The problem of highly dynamic mobile networks introduces new questions on how routing can dynamically adapt to achieve roaming and be simple and lightweight. In this paper a protocol for routing through low-powered mobile devices is presented. It is based on storing a list of all available routes on each node in the network, which provides an efficient way of switching between routes to use the best one when topology changes occur. The routing protocol has been developed for a Wireless Sensor Network (WSN) so that it can provide Internet to any host. This network consists of a special node called gateway that provides Internet access, a client node that makes Internet requests, and several intermediate nodes that forward packets between the first two when these are not in range of one another.

The rest of the paper is organized as follows: firstly related work is introduced, in the areas of metrics on routing algorithms, existing routing algorithms and how Internet access can be provided on a wireless sensor network; secondly, the designed algorithm is presented with its problems and an example for clarifying understanding; following, performed evaluations and comparisons are shown; lastly, conclusions and future work are presented.

2. RELATED WORK
The work presented in this paper is related to routing algorithms or improvements on existing routing algorithms that apply on WSN devices compliant with the IEEE 802.15.4 standard [6].

2.1 Routing protocols
ZigBee is an alliance for developing communication protocols using small, low-powered devices. The ZigBee routing protocol is specified in [5]. Based on distance vector, it relies on special nodes called ZigBee routers that maintain routing tables in order to be able to relay frames from a source to a destination. This protocol can work in star, tree or mesh topologies. Focus will be made in mesh topologies.

Basically routes are established on-demand. When a node wants to communicate with another node it broadcasts a route request command and the destination sends back a route reply. Intermediate nodes forward the request and when the destination is reached, it replies with the route. Intermediate nodes fill their routing tables using this mechanism.

This approach has the problem of flooding the network in order to discover a route. Once the route is established it is used without the need of extra routing packets to be used. This approach seems efficient for networks whose topology does not change so frequently. In a highly dynamic network the availability of a route on demand may take some time. Also, it might not be desirable for an IP based network with only acceptable limits for delay and retransmissions.

Solutions can also be made with clusters. In [7] a proactive vector routing protocol that relies on a cluster network architecture is presented. The basic interaction of this protocol relies on advertising routes through route description lists. A node stores several routes to a certain destination but only advertises the best route to that node. This approach, even if for advertising clusters and not individual nodes, seems adequate for rapid and efficient route selection. Keeping records of different routes to a particular destination allows for efficient routing when links change dynamically.

2.2 Routing metrics
One of the main tasks when building routes between two nodes is how to evaluate good routes. The most common metric is the amount of hops a packet must pass through in order to reach its final destination, also known as the hop count.

A novel approach for measuring route quality in wireless sensor networks is presented in [1]. The scheme called Cumulative Link Routing Algorithm (CLQR) is an innovative approach since it does not use minimum hop count or accepted transmission count; instead the cumulative link quality is used.

Minimum hop count is criticized due to possible high Packet Received Rate (PRR) since wireless sensor networks tend to be asymmetrical and unstable. On the other hand, the Excepted Transmission Count (ETX) metrics are not adequate for network load balancing and, furthermore, they flood the network with probe packets. The presented approach lays its efficiency on the probability that a packet can be successfully received by the receiver. The metric used here can be evaluated from CLQ_{old} using

$$CLQ_{new} = CLQ_{old} \times PRR$$

where PRR is the packet received rate, calculated as the number of received packets at destination node over the number of transmitted packets at the source node.

2.3 Routing loops

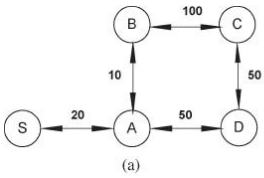
Routing loops are a common problem on networks where packet routing is used. When a route is advertised through the network, it may happen that loops are generated. Simply put, a loop is generated when a node A uses node B in order to reach a destination C, and B uses node A to reach C. This leads to a routing loop since A will send a packet with destination C to B, and B will send it back to A. Several solutions exist to this problem, as stated in this section.

A common solution to routing loops is split horizon. As mentioned in [8], the loop problem is caused because two nodes are mutually deceiving each other that they can reach a third node via the other. Split horizon proposes avoiding including routing updates to a node from which that update came from originally.

Also in [8] route poisoning is proposed as a solution to routing loops. In this approach a maximum value is set for the metric between two nodes. When a loop is detected in a routing table, the metric is set to a maximum value, which allows other nodes to understand that such a route has to be ignored.

Split horizon and route poisoning can be combined into a method called "split horizon with poisoned reverse". Here the basic mechanism of split horizon is implemented, but instead of not advertising routes through the same path they were originally received, they are advertised but with the maximum possible metric. This way unreachable routes are advertised. The advantage of this approach is that in pure split horizon, loops are stored for a short time and are eliminated by timeouts eventually. Advertising unreachable routes eliminates the need to wait for a route with loops to be eliminated since it is never used or advertised again due to its unreachable state. The only drawback is the increased size of routing updates, due to the advertising of routes with maximum possible metric.

In [7] several loop prevention mechanisms are used. Firstly, the entries in the routing table are purged periodically, in order to process the new incoming information about a particular destination. Secondly and more interestingly, instead of using a next-hop field in the traditional sense in the routing tables, the protocol uses an Out Neighbor ID (ONID). This field represents the node that is chosen to send all outgoing communications in order to reach a sink node (a data receiving node).



(a)

Time	Node ID	SID	ONID	HC	LPL
1	S	S	S	0	255
2	A	S	S	1	20
3	B	S	A	2	10
3	D	S	A	2	20
4	C	S	D	3	20

(b)

Figure 1: An example of Descriptor Reject on the routing scheme presented in [7].

Here every node advertises a descriptor in which they indicate their ONID. In the example of Figure 1 when node C sends out its descriptor, node D and B will receive but D will reject it. Node D can see in nodes C descriptor that it is its ONID, thus if it adds C to its table it will create a loop (C would send all outbound communication to D and vice versa).

This approach proves effective for preventing loops but limits the interaction within the network. It introduces the existence of a sink node towards all communication flows, limiting the ability to send packets to any node in the network.

2.4 Internet access

The standard presented in [9] allows IPv6 packets to be sent from and received over IEEE 802.15.4 based networks. The frame format for transmission of IPv6 packets is defined in the 6lowpan standard. Addressing is allowed for both IEEE 64-bit extended addresses and 16-bit short addresses. The most notable aspect of the standard is its adaptation layer, which provides fragmentation and reassembly of packets. The specification in [9] does not include specific routing protocols since the objective is to allow IPv6 packet transmission only, leaving any other network functionalities out of its scope.

3. DESIGN

In this section the design of the algorithm is explained. Firstly the main challenges are presented which motivate the purpose of the algorithm. Next a high level description is provided which provides the reader with insight on how the protocol works. Finally implementation details are provided.

3.1 Challenges

The two main problems tackled in this work are routing and roaming on WSNs. The purpose of having routing is that a client mote of the sensor network does not need to connect directly to the gateway, but it can reach it by hopping through other motes in the sensor network. Routing needs to be provided efficiently, so that, if a route between the client and the gateway exists, it is used, and the packet reaches its destination in acceptable time. This implies that the best possible route should always be used. Roaming will allow a client to change the used mote dynamically. This requires that, if one of the intermediate motes is out of reach, another mote that forms a route to the destination can be used.

3.2 Solution design

This work is based mainly on the code provided by Andrea Crotti, Marius Grysla, and Oscar Dustmann, which enables Internet usage between one client and one gateway with a direct connection between them[1]. In the rest of this paper, this work will be referred as the project's starting point. With this starting point, the code was forked in order to modify it and add routing capabilities. A new fork was created for collaboration between

[1] See: https://github.com/AndreaCrotti/Network-mote

[2] See: https://github.com/ignacioavellino/Network-mote

different developers[2]. The starting point allows for one node to have Internet access through another node, which must be directly in its range, not supporting routing or multi-hop environment.

The implementation is built for the TelosB platform [2]. It consists of three main components: a client mote, several intermediate motes and finally a gateway mote, which provides Internet access. This can be seen in Figure 2. A client computer makes an IPv6 request, which, through the starting point project, is translated into several packets handled by the client TelosB mote. These packets need to reach the gateway in order to be sent through the Internet, and thus need to be routed through a network of motes when the gateway is not directly in reach.

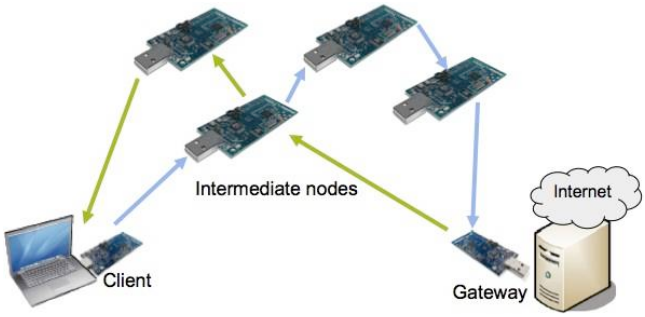


Figure 2: Implementation architecture. The client, gateway and several intermediate nodes are depicted.

The programming language used is nesC, intended to run in the TinyOS 2.1.1 platform. BLIP[3], the Berkeley Low-power IP stack, provided an IPv6 layer on top of `6lowpan` and IEEE 802.15.4. The development was done on Ubuntu 11.0.4 as the blip is only supported on Linux as of 2.1.1. Git was used for version control, code collaboration and distribution.

In order to implement a simple but efficient routing algorithm, routing tables will be used. Each node maintains a list of routes for each possible destination, which includes:

- Destination node ID: the ID of the node it is intended to reach.
- Metric: a metric that indicates how good the route is.
- The ID of the next hop: the next node the packet should pass through in order to reach this destination
- Timeout

This list of routes is sorted according to the metric parameter, so only the path with the best one is always used. The timeout parameter is used for link expiration; when a node does not receive a route advertisement for a certain period of time defined in the timeout, the route is removed from the routing table.

The mechanism for performing routing updates and sending regular packets is explained in the following two sections. Later, encountered problems with the solution are presented.

3.2.1 Updates

In order to maintain the routing table, each node sends periodically a routing update containing a list of all the destinations it can reach. For each destination it includes:

- Destination node ID
- Metric to that destination

When a mote receives an update packet, it adds the received routes to its own routing table using the sender of the update as the next hop.

3.2.2 Sending packets

When a node wants to send a packet to a destination it uses the route with the best metric. If sending to the next hop of the best route fails, the route is deleted, the next best route is chosen as the current best one and sending is retried. Also all the routes that have the dead route as a next hop are deleted since they are broken. If there are no possible routes to the desired destination, the packet is discarded.

3.2.3 Problems

As mentioned before, routing has to be provided efficiently, thus the algorithm is kept as simple as possible so that routing does not add too much overhead to the purpose of the communication itself. However another important limitation is the memory available on these devices. Since alternative routes are stored, the memory needed for storing the routing table can be significant. This solution is suited for dispersed wireless sensor networks with a relatively small number of nodes. This means that the number of routes a node can have knowledge of, and the number of possible paths to a certain destination should be relatively small.

Another major problem is the existence of routing loops, which can prevent the algorithm from routing packets correctly and efficiently. Routing loops cause count to infinity as nodes advertise a dead link between each other before they realize it is down. This implies that other nodes store information on dead links, and broadcast it. This process is repeated and each time the dead link is stored with hop count increased by one. If this is not prevented, the hop count will grow continuously (to infinity). However, routing loops can be detected by putting a maximum limit to hop count. But this method is too slow to converge and the delay introduced in this approach is often not acceptable, since many timeouts might occur and TCP might try resending packets that are effectively, but later, acknowledged.

3.3 Implementation

In the current implementation, the used metric is composed of the hop count and the link quality to that particular destination node. A metric with smaller hop count is considered better, and when two routes have the same hop count, the link quality is used to decide which one is better. The timeout for each entry in the routing table is of 15 seconds. Routing updates are sent every 5 seconds. Lastly, if a received update contains an average link quality worse than -80dB, it is automatically discarded and not considered as a valid route.

The algorithm provides efficient routing in the sense that an alternative route is immediately chosen when the current best route is not available. When a route becomes unavailable, the current node does not have to wait for routing updates from neighbors to determine the new best path. This is done by storing every possible route that is advertised to a destination, ordered by best metric. Given the dynamic nature of wireless sensor networks it is expected that routes will often become unavailable, and new routes to a destination have to be determined.

3.3.1 Example

In order for the protocol to be clearer, an example is presented. Firstly routing will be explained by depicting how motes build their routing tables using Motes 2, 3 and 4 as examples; then

[2] See: https://github.com/ignacioavellino/Network-mote

[3] See: http://docs.tinyos.net/tinywiki/index.php/BLIP_Tutorial

roaming will be explained by showing how Mote 1 changes its routing table to reflect the topology changes caused by its mobility. Picture the depicted sensor network in Table 2.

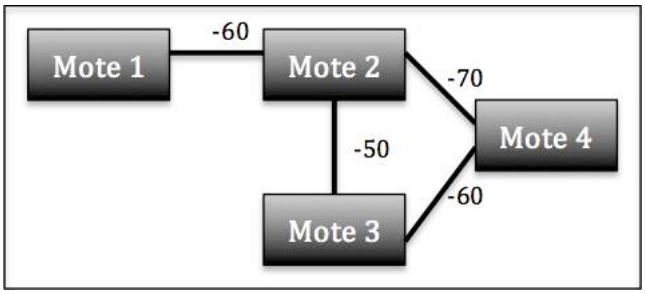


Figure 3: A simple sensor network topology.

Suppose Motes 3 and 4 have not sent a routing update yet, so the routing table of Mote 2 would contain only a direct route to Mote 1 as seen in Table 1.

Table 1: Routing table of Mote 2 before Motes 3 and 4 send a routing update

Destination	Next hop	Hop count	Link quality
1	1	1	-60

Now Motes 3 and 4 send a routing update. The update of Mote 3 contains a route to Mote 4, whereas the update from Mote 4 contains a route to Mote 3. Both also contain a route to Mote 2. These routing updates can be seen in Table 2 and Table 3 respectively.

Table 2: Routing update of Mote 3

Destination	Hop count	Link quality
4	1	-60
2	1	-50

Table 3: Routing update of Mote 4

Destination	Hop count	Link quality
3	1	-60
2	1	-70

When Mote 2 receives Mote's 3 routing updates, it ignores the route to itself and adds the direct route to Mote 3, and the route to Mote 4 through 3. Mote's 1 new routing table is depicted in Table 4.

Table 4: Routing table of Mote 2 after receiving the routing update from Mote 3

Destination	Next hop	Hop count	Link quality
3	3	1	-50
4	3	2	-55

Then, Mote 2 receives the update from Mote 4 and adds a route to both Motes 4 and 3 as seen in Table 5.

Table 5: Routing table of Mote 2 after receiving the routing update from Mote 4

Destination	Next hop	Hop count	Link quality
3	3	1	-50
3	4	2	-65
4	4	1	-70
4	3	2	-55

At this point Mote 2 can reach Mote 3 through two different routes. It will choose the best one first, and if this one is not available, it will delete it and choose the next one. The same is valid for reaching Mote 4.

In order to demonstrate roaming we will use Mote 1 from the previous example. Suppose now Mote 2 broadcasts its routing update, as shown in Table 6.

Table 6: Routing update of Mote 2

Destination	Hop count	Link quality
3	1	-50
4	1	-70

When Mote 1 receives the routing update from Mote 2, it will add paths to Motes 2, 3 and 4 as shown in Table 7.

Table 7: Routing table of Mote 1 after receiving the routing update from Mote 2

Destination	Next hop	Hop count	Link quality
2	2	1	-60
3	2	2	-55
4	2	2	-65

Now Mote 1 is mobile and changes its position, leaving it out of reach of Mote 2 and in reach of Mote 3, as shown in Figure 4.

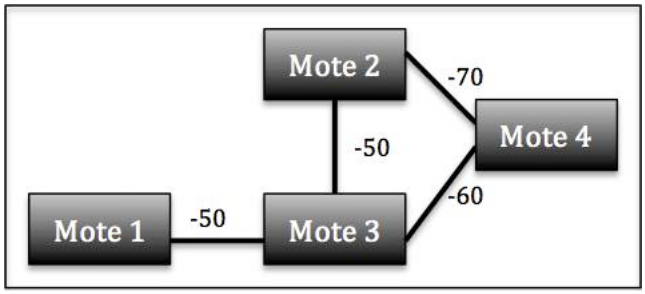


Figure 4: Network topology after Mote 1 has moved

If Mote 1 was communicating with Mote 2, it will detect that Mote 2 is no longer acknowledging its messages so it will delete the entry to Mote 2, whereas if it was not communicating with

Mote 2, the entry will expire after its timeout since updates from 2 are no longer received. When the route to Mote 2 is deleted, all of the routes that include Mote 2 as the next hop are also deleted. When an update from Mote 3 is heard, it will be added to its routing table. Consider the routing update from Mote 3 as shown in Table 8.

Table 8: Routing update from Mote 3 after Mote 1 has moved

Destination	Hop count	Link quality
4	1	-60
2	1	-50

After Mote 1 receives this routing update, it will rebuild its table to reflect the topology changes. The resulting table depicting the topology changes is shown in Table 9.

Table 9: Routing table of Mote 1 after its location change

Destination	Next hop	Hop count	Link quality
3	3	1	-50
4	3	2	-55
2	3	2	-50

3.3.2 Advantages

Firstly, in this algorithm count to infinity is avoided in the general case. Since a route that is no longer available is immediately deleted, it is not broadcasted and therefore other motes will not know of it. Count to infinity still might occur in the case when a route that is not used is no longer available. Since it is not used it will not be immediately deleted from the routing table and thus will be advertised even though it's not available. But as mentioned before, this algorithm is more suitable for disperse wireless sensor networks.

Secondly, this algorithm provides efficient routing, as the motes are aware of more than one possible route to one destination. This allows choosing an available route when the preferred one is down, without having to wait for a routing update from the neighbors.

Finally, roaming is supported since when an intermediate node used in a route from the gateway to the client or vice versa moves and is out of reach, another intermediate node is immediately chosen and the route to the moving node is deleted. This node will advertise its existence and thus nodes in reach will now store a route to it, allowing its use as part of a route to a particular destination. The end nodes, the client and the gateway, do not notice that an intermediate node has moved, unless it is a neighboring node. In the latter case they will simply choose the next available node.

4. EVALUATION

Evaluation of the algorithm was performed using five TelosB motes, one as gateway, one as client and three intermediate motes. In order to facilitate the evaluation, the routing algorithm was enhanced with debug sentences that allow the interaction to be seen, through the use of LEDs, in the motes. A red LED is turned on every time an IP packet is sent, a green LED every time an IP packet is received and a blue LED every time a routing update is received. This allows visualizing which intermediate motes are chosen and which are not. This proved useful for making sure that an alternative route was chosen while roaming, and actually seeing on the motes which is this new route.

Three different evaluations were performed:

- Bandwidth variation: by testing different bandwidth values it is intended to find saturation point of the network.
- Number of hops variation: by increasing the number of hops it is intended to see how the performance of the network changes.
- HTTP request: the loading time of a web page using the presented algorithm is measured.
- Packet size variation: by changing the packet size it is intended to find how it affects the network performance.

4.1 Setup

The evaluation was done in a room with no other interfering motes, and Wi-Fi devices turned off. It is important to note that sometimes results were completely out of the ordinary (for example, a packet loss of 90%). It is assumed that these erratic events were caused by local interference, and in that case the test was suspended and continued at a later time. In all measurements, one computer was set as the client with a mote with IP address 10.0.0.1, another as the gateway with IP address 10.0.0.254 and several intermediate nodes with IP addresses ranging from 10.0.0.2 to 10.0.0.4.

The RF_Power of the TelosB motes was set to 3 in all experiments since they were all found in the same room and more power was not needed. It was tested that with an RF_Power value lower than 3, the network was highly unstable at even small distances of 0.5 m. The motes were aligned in one line with the intermediate distance of approximately 3m. Moreover, in order to achieve a specific topology, static routes were hard coded into the motes. In this way, we could force the data packet transmission to follow a certain path.

4.1.1 Tools

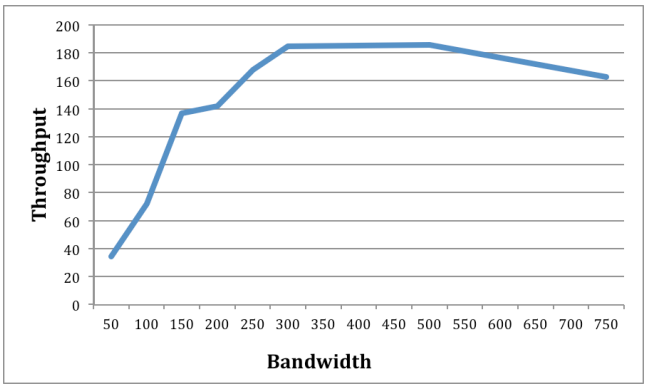
We used "JPerf" v2.0.2 to perform various tests. JPerf is a GUI frontend to "IPerf" based on Java and it was chosen because it provides a simple to use interface and allows saving test configurations in files which can be loaded later to perform the same experiment. IPerf allows us to vary parameters such as bandwidth and packet size yielding network characteristics like bandwidth, throughput and jitter. All the experiments were conducted with tools running on Ubuntu 11.0.4.

4.2 First experiment: bandwidth variation

The first experiment was intended to test the throughput of the network. It consisted of only one intermediate node (one hop) and repeating the IPerf test with different bandwidth values. It was done to find the saturation point of the network, where the throughput remains constant even on increasing the bandwidth.

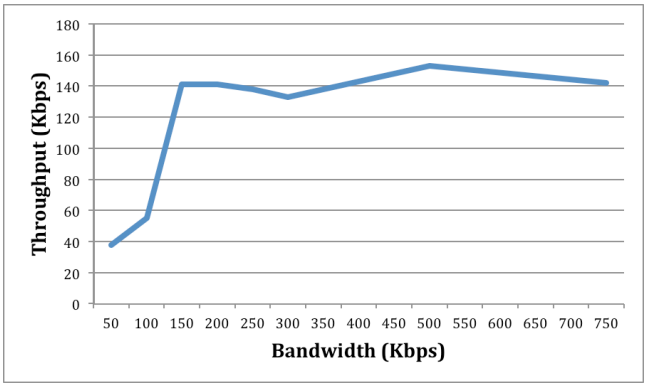
The experiment was executed with bandwidth raging between 50 and 750 Kbps. The results are shown in Graph 1. Here two different trends are present. At first the throughput increases linearly with the bandwidth. Secondly, after the maximum throughput is achieved, it either remains constant or decreases slowly. In this case, this point is found at bandwidth near 300 Kbps where a throughput of 180Kbps is reached. Moreover, it can be appreciated that after the saturation point is reached the

throughput remains stable. The throughput drops slowly towards the end of the graph. This could be possibly explained by an increase in packet loss (13%) in the last test run, but it does not affect the trend of throughput stability after the saturation point is reached.



Graph 1: Bandwidth saturation of network with one hop. Stream from server to client

The presented results are only for the download stream, from the server to the client. The upload stream from the client to the server presents similar results, which are depicted in Graph 2.



Graph 2: Bandwidth saturation of network with one hop. Stream from client to server

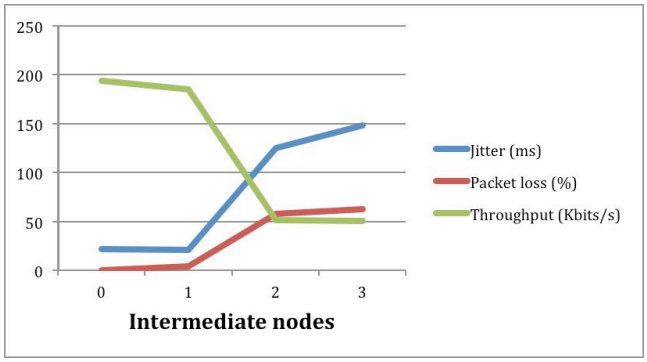
In the stream from the client to the server, results show that there is also an increase of bandwidth until around 300Kbps, where the throughput stabilizes at around 150 Kbps. The only unusual behavior detected is at bandwidth 150 Kbps where there is a peak of throughput of 140 Kbps. While repeating the experiment, similar results were obtained. However, we could not find any clear explanation for this behavior.

More importantly, the results obtained here can be seen in accordance with the maximum bit rate of TelosB motes of 250 Kbps. That also explains why the throughput becomes saturated at somewhere around 170-180 Kbps for the bandwidth values of 200 Kbps and more.

4.3 Second experiment: number of hops variation

In the second experiment it was our intent to test the variations in jitter, loss and throughput while increasing the number of intermediate hops. The previous test using IPerf was performed with intermediate nodes ranging from 1 to 3. The bandwidth was set to the maximum value of 300 Kbps obtained by previous experiments. As the number of intermediate nodes was increased, the network performance decreased accordingly.

The results of the experiment for the download stream, from the server to the client, are shown in Graph 3.



Graph 3: Network performance as intermediate nodes are increased

For comparing the results, the starting point is when no intermediate hops are present. Here the packet loss is minimum, jitter barely 25ms and throughput near 200Kbps. When the number of intermediate nodes was increased, as expected, the packet loss and jitter increased correspondingly, while the throughput decreased. The last measurement performed with 3 hops show that packet loss reaches over 50%, jitter increases significantly to 150 ms and throughput reaches a very low bit rate of 50Kbps. This shows that the network performance was not good for a large number of hops. Interestingly, when only one hop is present, the performance of the network does not vary significantly.

4.4 Third experiment: HTTP request

The third experiment was motivated by our desire to test our solution in a real world scenario. So this test was done to test the load time of a webpage. The test was done on Firefox 5.0 using the extension "lori" (Life-of-Request-info) and JavaScript disabled. "Lori" provided us with values like TTBF (time to see the first byte), TTC (time to display the page), page size and the number of requests. Before each run of the experiment the cache and the cookies of the browser were cleared. However, to our hearts' discontent, we could not succeed in getting even one page (www.google.de) displayed on our browser.

This led us to further investigate. One of the few possible reasons that could convince us was the IP packet size. TelosB motes [10,11] have the MAC payload size of 29 bytes. This prevents the message buffers from taking too much memory. Theoretically, any IPv4 link must be able to forward a packet of size of at least 68 bytes [12]. Similarly, for an IPv6 link it is at 1280 bytes [13]. This motivated us to do some tests based on maximum transmission units using "ping".

4.5 Fourth experiment: packet size variation

In the fourth experiment, we wanted to investigate how the variations in packet sizes (transmission units) affect the network performance. The experiment was performed with only one intermediate hop. We used the "ping" time, a very popular tool available for all the operating systems.

```
ping [-i 1][-M do] [-s <packet size>] [host]
```

The MTU Discovery strategy "do" was used to prohibit fragmentation of packets even locally. The packet sizes were varied from 50 bytes to 200 bytes. The ping interval was set to 1 second.

Table 10: Network performance while varying packet size

Pckt. size (bytes)	Pckt. loss (%)
25	3%
50	16%
75	60%
100	70%
200	86%

This experiment reinstated our doubt into the fact the TelosB MAC payload size was indeed a limit for the usage of our system for real life Internet.

4.6 Other observations

Another expected phenomenon was detected while making tests with using ping. Ping was tested with intermediate nodes ranging from 0 to 3, and with interval ranging from 0.1 to 2 seconds. Independently of the number of intermediate nodes, it was always observed that ping stopped working when the interval was below 0.3 seconds. This can be due to the fact that the round trip time for the ping ranged from 200 ms to 300 ms. When the interval is lower than the round trip time, ping ceases to work due to the fact that the queues in the sensor motes would quickly get full and it would eventually crash the mote. Another possible reason can be that the motes cannot process sending, receiving and processing of so many packets so fast. Also, network performance decayed but this was already observed in previous tests using IPerf.

Lastly, it was observed that, while roaming, packet loss remained null or very low. When performing different tests, nodes were made unavailable on purpose to see the results and compare them to results where the sensor network topology did not change. When removing a node from a path, a secondary route was used immediately and only in one case out of ten the packet loss increased significantly (more than 5%).

5. CONCLUSIONS

A simple but efficient routing algorithm has been implemented in a sensor network, through the use of route advertisement; each mote broadcasts periodically known routes to other motes, which build a routing table with several paths to different destinations. This enables rapid changing of routes that reflect changes in the topology of the network, allowing for a wireless sensor network to provide Internet access to any node in the network given that there is a route from that mote to the gateway mote. The most efficient route is based on hop count and link quality, which proved to be a good metric, but more complex metrics could be implemented such as Cumulative Link Quality [1].

Regarding the evaluation, several conclusions can be drawn. Firstly, the saturation point of the network was found around 300 Kbps. This shows how much can be demanded from the network if it were to be used in a real setup. Secondly, network performance proved to diminish while the amount of hops increased. The algorithm proved to be efficient on finding routes and keeping them since a path from the client to the gateway was always found when the motes were on range of each other. The problem arises when increasing the hops, and showed that for a real setup it might not be a viable solution for demanding networks. The cause of this can be blamed on high saturation of the medium, since all nodes were near each other and might interfere on their communication. Thirdly, it was observed how packet loss greatly increased when IP packets had to be split into smaller packets in order to be transmitted in the sensor network. This was expected, but with this test it was observed that a side effect of this is near to unusable Internet access.

Regarding the implementation itself, the algorithm proved efficient in its main goals. Routes were created, used, and maintained efficiently. The propagation of route information was done correctly since the expected route was always chosen by the motes, while arranging them in different physical positions. Roaming also proved to be efficient since, when changing the sensor network topology, an alternative route that was more efficient was chosen with no significant packet loss.

Lastly and more generally, it was observed that the network did not always behave in the same way. While performing different experiments, sometimes measurements were completely off the expected values. For instance, observing a packet loss higher than 90% when a previous experiment with barely different parameters showed no packet loss. This is blamed on environmental causes since when tests were repeated with the same parameters but at different times, more coherent results were obtained.

6. FUTURE WORK

At this point of the implementation count to infinity was prevented in the case where motes are dispersed and most of the stored routes are used. This allows for fast detection of dead routes and no transmission of dead routes. Methods such as route poisoning or split horizon are the most common solutions.

The infrastructure would benefit of multi client support, allowing more than one client to access Internet. At this stage of the implementation only one client can have Internet access. This would involve modifying other parts of the starting point project, related to how the packet is divided and prepared before it leaves the client or gateway mote.

Further investigations have to be conducted in order to improve the network performance. Optimizations in the algorithm, packet structure, messaging interfaces are needed to support a real world Internet usage.

7. ACKNOWLEDGMENTS

Firstly, to the lecturers Nicolai Viol and Tobias Vaegs for their great disposition and continuous help. Especially to Nicolai Viol for his encouragement all along the course and the project. Also to Andrea Crotti, Marius Grysla, and Oscar Dustmann for the code provided and their helpful documentation.

8. REFERENCES

[1] Liang, J.J. and Yuan, Z.W. and Lei, J.J. and Kwon, G.I.. 2010. Reliable Routing Algorithm on Wireless Sensor Network. In *Advanced Communication Technology (ICACT), 2010 The 12th International Conference on* (IEEE, 2010), 47-51. DOI= http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?tp=&arnumber=5440510.

[2] Crossbow Technology, Telosb mote platform datasheet. DOI=www.willow.co.uk/TelosB_Datasheet.pdf.

[3] Martin Haenggi, Twelve Reasons not to Route over many Short Hops, VTC04.

[4] Martin Haenggi and Daniele Puccinelli, Routing in Ad Hoc Networks: A case for Long Hops, IEEE Communication Magazine, Oct. 2005.

[5] ZigBee specifications, "ZigBee Document 053474r15", ZigBee Alliance, Feb 2007.

[6] Wireless Medium Access Control and Physical Layer Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), IEEE Std 802.15.4-2003, IEEE Computer Society, 01 October 2003.

[7] Shuaib, A.H. and Aghvami, A.H. 2009. A routing scheme for the IEEE-802.15. 4-enabled wireless sensor networks. In *Vehicular Technology, IEEE Transactions on*. (IEEE, 2010), 5135--5151. DOI=http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5164970.

[8] RFC 1058. Internet Engineering Task Force, Rutgers University. 1988. DOI=http://www.ietf.org/rfc/rfc1058.txt

[9] RFC 4944. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. 2007. DOI=http://datatracker.ietf.org/doc/rfc4944/

[10] http://mail.millennium.berkeley.edu/pipermail/tinyos-help/2007-August/027614.html

[11] http://mail.millennium.berkeley.edu/pipermail/tinyos-help/2006-April/016195.html

[12] RFC 791 Internet Engineering Task Force, INTERNET PROTOCOL, Darpa Internet Program, Protocol Specification, September 1981, http://tools.ietf.org/html/rfc791

[13] RFC 2460 Internet Engineering Task Force, `Internet Protocol, Version 6 (IPv6),` S. Deering, Cisco, R. Hinden, Nokia, December 1998, http://tools.ietf.org/html/rfc2460