A new Strategy to Improve the Pathfinding in Wireless ad-hoc Networks

Andreas Redmer
Department of Computer Science
University of Rostock
Rostock, Germany
Email: andreas.redmer@uni-rostock.de

Andreas Heuer
Department of Computer Science
University of Rostock
Rostock, Germany
Email: heuer@informatik.uni-rostock.de

***Abstract*—In link-state computer networks it is usual that every node knows the topology of the entire network and can make the routing decisions based on that. One of the protocols in use is OLSR. The OLSR routing protocol implements the algorithm of Dijkstra to find the shortest paths from the nodes to the gateways of the network. For that purpose, Dijkstra's algorithm has to be executed k times, while k is the number of gateways. In this paper, we present a strategy that generalizes all gateways to one gateway. We call this the General Gateway Strategy. Using this, the Algorithm of Dijkstra has to be executed only one time, which significantly increases the performance of the overall algorithm to find the shortest paths to the gateways.**

Keywords-Wireless mesh networks; Ad hoc networks; Wi-Fi; Shortest path problem.

I. INTRODUCTION

In link-state computer networks it is usual that every node knows the topology of the entire network and can make the routing decisions based on that. One of the common protocols is OLSR (Optimized Link State Routing). It is a protocol for link-state routing in ad-hoc networks and is described in RFC 3626 [1].

The discovery of topology is specified in OLSR by two kinds of messages: HELLO- and Topology-Control (TC) Messages. The discovery of neighborhood is done by HELLO-Messages, which contain the current direct neighborhood of the sending node. TC-Messages contain information of the entire network, encompassing all nodes and routes to these nodes.

By the propagation of TC-Messages in the network, every node can derive the network topology and can make routing decisions using that knowledge. The quality of the link between two nodes is described by two parameters: link quality (LQ), which is the quality from the current node to the neighbor and neighbor link quality (NLQ), which is the reverse direction. These values are not necessarily equal for the same link.

Some of the nodes in the network share their internet connection and make it available to the rest of the network. These nodes are called gateways. Every node must be able to determine the shortest path to the nearest gateway.

OLSR uses the Algorithm of Dijkstra [2] to find the shortest paths between the nodes. The general problem is, that the Dijkstra Algorithm must always be executed k times, where k is the number of gateways.

Originally Dijkstra's Algorithm was described in [2] to find the shortest path between two nodes in a graph (single-pair shortest path). That means the algorithm will terminate as soon as the shortest path is found. For that purpose it is not necessary to explore the entire graph. In OLSR the Algorithm of Dijkstra is used to determine the shortest path from one gateway to all the other nodes (single-source shortest path). So, the algorithm explores the entire graph and can not be terminated sooner. This implies that the algorithm will always have the runtime that is described by the worst-case complexity. It is important to know that many other improvements of the Dijkstra Algorithm (e.g., the A*-Algorithm [3]) are only aiming on terminating sooner, by finding the destination node faster. So, these improvements are not applicable for the needs in ad-hoc wireless routing protocols. The network-graph can be described as a weighted directed graph (V, E) with

- $V := \{\text{a set of nodes}\}$
- $E := \{\text{a set of edges}\}$
- $G(\subseteq V) := \{\text{a set of gateways}\}$
- $(a, b) \in E$ (with $a \in V$ and $b \in V$) describing an edge from a to b
- $f : E \rightarrow \mathbb{R}$ a function describing the asymmetric distance between two nodes. Thereby the value LQ is defined as $f((a, b))$ and NLQ as $f((b, a))$.

Modern implementations Dijkstra's Algorithm use a Fibonacci-Heap [4] to store the nodes, which reduces the amortized complexity of the worst case to

$$O(|V| \cdot log|V| + |E|) \tag{1}$$

Based on that, the complexity of the procedure to find all the shortest path from all gateways to all nodes is

$$O(|G| \cdot |V| \cdot log|V| + |E|) \tag{2}$$

The strategy described in the following section removes the factor $|G|$ and so reduces the complexity significantly. After that, we present the experimental results, that show the actual enhancements of the new strategy. In the last section, a conclusion and future work is given.

 ISBN: 978-1-61208-236-3

II. A NEW STRATEGY

Figure 1 shows a network graph with four gateways (G_1, G_2, G_3 and G_4) printed in blue. The red nodes are non-gateway nodes. The labels of the edges represent examples of the metric, which is used to describe the distance between two nodes. In this case, a simple additive metric is applied. So, higher numbers represent a higher distance and a worse connection. Low values stand for a good connection. Dijkstra's algorithm must be executed four times there. Each run uses one of the gateways as initial node. In the end all four paths that lead from a node to a gateway will be compared and only the shortest one returned. Each run considers the whole graph. In the example it would be determined four times, that the shortest path from C to E is always across D and has always the costs 2. So, calculations are repetitive.

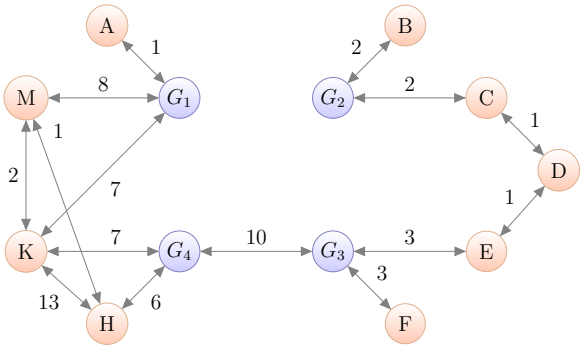

Figure 1. An example for an unmodified network graph

We developed an idea to solve this problem of executing the same calculations multiple times on the same graph. The basic principle for this, is the fact, that a shortest path to a gateway never crosses another gateway. The theoretical perfection would be a graph, where all the gateways are centralized in the network, there are no nodes between the gateways and all the gateways can reach each other with the costs 0. In this case it would be possible to combine all gateways to one gateway. So it would only be necessary to find the shortest paths to this generalized gateway.

In practice that is not the case, because the gateways are spread all over the network and have always costs (> 0) to reach each other. But the direct edges between the gateway are not relevant for our problem, since we have already noticed that a shortest path to a gateway, never crosses another gateway. It is also not necessary to find a path from a gateway to another gateway. So, all the irrelevant edges can be substituted by 0-Edges[1] and so all gateways will be connected by 0-Edges. It does not matter if there has been an edge existing between these gateways before, or not. That also means, that the entire graph does not have

[1]Exact would be "identity-element-edges", which means that always the identity element of the metric should be used. In additive metrics 0, in multiplicative metrics 1, etc.

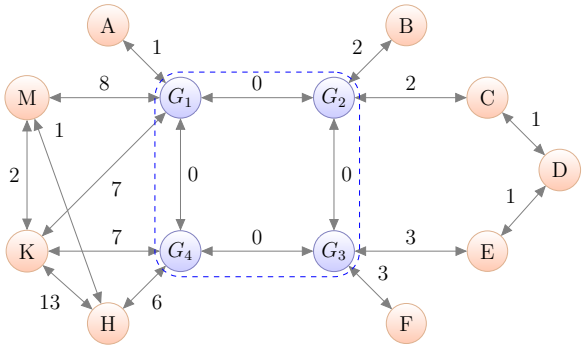

Figure 2. A network graph with generalized gateway

to be connected. It is also possible to apply this technique to separated sub-graphs, whereat each sub-graph contains at least one gateway. It also does not matter how many 0-Edges are inserted. It only must be ensured, that every gateway can reach each other gateway with the cost 0. In Figure 2 these 0-Edges are added and the generalized gateway is marked with a dashed line.

For the newly created graph, the following procedure will be applied:

1) Select a random gateway as initial node.
2) Execute Dijkstra's Algorithm one time.
3) Remove the 0-Edges from the result[2].

For instance, one uses G_1 as initial node in Figure 2. Firstly, Dijkstra's algorithm returns shortest paths from every node to G_1. Amongst others the route:

$$G_1 \leftarrow G_2 \leftarrow C \leftarrow D$$

will be returned for node D. In the beginning of each route there can be several gateways now. All the 0-Edges and the leading gateways have to be removed in the final result e.g., it would be the route:

$$G_2 \leftarrow C \leftarrow D$$

for the node D. This is the shortest path to any available gateway. We called this method General-Gateway-Strategy (GGS). This strategy returns the same result as the multiple execution of Dijkstra's algorithm and the additional selection of the closest gateway (the shortest of all the shortest paths).

III. RESULTS

To evaluate the performance of the GGS we generated one million weighted directed graphs randomly. Each graph was connected and had the characteristics

- $100 \leq |V| \leq 1000000$ and
- $|G| \in \{1, 2, 4, 8\}$.

We implemented the Dijkstra Algorithm using a Fibonacci Heap in Java. For this purpose, a desktop PC was equipped

[2]All 0-Edges in the beginning of a route between two gateways.

 ISBN: 978-1-61208-236-3

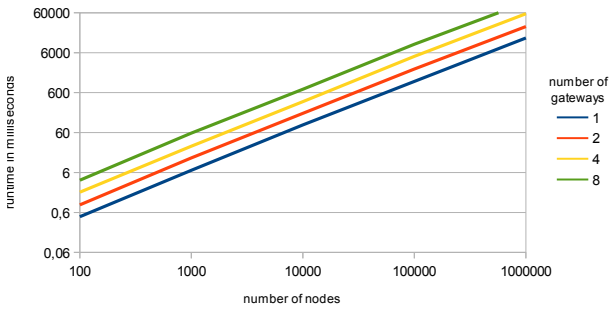

Figure 3. Runtimes without General Gateway Strategy

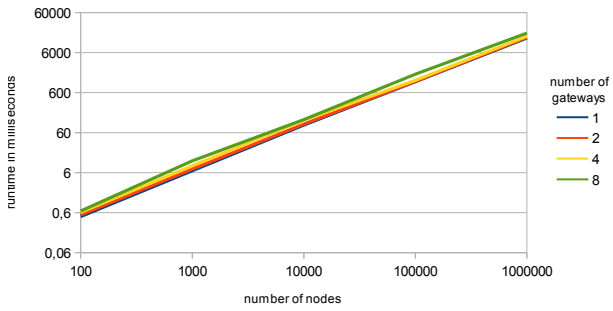

Figure 4. Runtimes with General Gateway Strategy

with a 64 bit dual core 2.2 GHz CPU, 128 kB L1 CPU cache, 512 kB L2 CPU cache and 4 GB DDR II RAM (800 MHz). We ran the Java Virtual Machine OpenJDK 1.9.9 [5] on the operating system Ubuntu Linux 10.04.3 LTS [6] with the 2.6.32-32 Linux Kernel [7].

To measure the runtime we used the Java API Method `System.nanoTime()` [8] directly before and after our algorithm execution. The algorithm was executed several times to avoid deviation due to caching or just-in-time-compilation. The measured times in milliseconds of the endpoints of the test-interval are presented in Table I.

The results of all test runs are plotted in Figure 3. To plot the runtimes (y-axis), a logarithmic axis was chosen in order to provide a better scalability in the plot. The x-axis shows the number of nodes used in the graph. There are four lines representing the number of gateways in the graph ($|G| \in \{1, 2, 4, 8\}$). Based on the complexity of Dijkstra's Algorithm (Formula 1) the results grow as expected exponentially by increasing the number of nodes. Increasing the number of gateways adds a factor growth to the runtimes.

Figure 4 shows the same kind of diagram as Figure 3, but here are the results of the GGS plotted. As expected, all the lines are now very close to the blue line in Figure 3, which represents only one execution of the Dijkstra Algorithm. There is only a very small increment resulting from the raising amount of gateways, which is due to the last step of the GGS. This last step is the removal of the 0-Edges in the final result. In efficient implementations this step might be included in the post-processing of the result and so it might not be noticeable anymore.

As it can be seen in Table I, there are no considerable differences between the runtimes, in networks with only one gateway. But in the case of one million nodes and eight gateways, the GGS takes practically only around 17% (theoretically 12.5%) of the runtime compared with the conventional method that executes the Dijkstra Algorithm eight times. That makes it 5.9 times faster than before. Generally, there is always a small overhead, which results from the postprocessing of the final result. The theoretically possible improvements by the factor k ($= |G|$) can not be reached.

Table I
RUNTIMES OF ENDPOINTS OF THE INTERVAL IN MILLISECONDS

	without GGS		with GGS	
nodes	1 gateway	8 gateways	1 gateway	8 gateways
100	0,47	3,86	0,48	0,66
1000000	14022,50	108703,70	13934,73	18494,70

IV. CONCLUSION AND FUTURE WORK

We have presented a strategy that generalizes all gateways in an ad-hoc wireless network to one gateway. We call this the General Gateway Strategy. Using this, the Algorithm of Dijkstra has to be executed only one time, which significantly increases the performance of the overall algorithm to find the shortest paths to the gateways.

Our new strategy improves the overall complexity by the factor k (number of gateways). This value can be very high for larger scaled mesh networks. Our measurements showed, that even for small scaled networks (not more than eight gateways), there are improvements in the performance of more around 600%. For larger mesh networks a higher enhancements can be expected. This highly depends on the degree of connectivity of the network graph and on the amount of gateways.

In the future, the GGS can be practically used in data analysis algorithms. In one of our research projects we have captured the topology data over more than one year in the wireless mesh-network. For that purpose one of our nodes in the networks saved the topology data one time per minute into a relational database. To analyze this data it is always the first step to use the Dijkstra Algorithm to calculate the current routes at a particular instant of time. This enables the analyst to perform a risk analysis, a bottleneck analysis, evaluate the existence and quality of alternative routes and more. This information can be used for further network planning, to find measures for the importance of nodes and edges and to enhance the network quality. In the past, we passed other tries to increase the performance of the data analysis, e.g., by using cloud computing [9]. The approach

 ISBN: 978-1-61208-236-3

presented in this work, brings an additional performance boost to the data analysis.

Furthermore, the GGS can directly be implemented on devices, which implement the OLSR routing protocol. Since it is fully compatible to the conventional implementations of OLSR, updated devices can be used in existing networks without any restrictions. This will lead to reduced CPU load on the routing node and so a higher throughput of user data.

REFERENCES

[1] T. Clausen, P. Jacquet, C. Adjih, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L. Viennot, "Optimized Link State Routing Protocol (OLSR)," *Network Working Group Request for Comments : 3626 Category : Experimental*, 2003.

[2] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[3] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics SSC*, vol. 4, no. 2, pp. 100–107, 1968.

[4] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *J. ACM*, vol. 34, pp. 596–615, July 1987. [Online]. Available: http://doi.acm.org/10.1145/28869.28874

[5] OpenJDK, "Openjdk website," 2012. [Online]. Available: http://openjdk.java.net/ [retrieved: July, 2012]

[6] Canonical, "Official ubuntu linux 10.04.3 download page," Canonical Group Limited, 2012. [Online]. Available: http://mirror.eftel.com/ubuntu-dvd/10.04.3/ [retrieved: July, 2012]

[7] L. Torvalds, "The linux kernel archives," 2012. [Online]. Available: http://www.kernel.org/ [retrieved: July, 2012]

[8] Oracle, "Java api online documentation of the class system," 2011. [Online]. Available: http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/System.html [retrieved: July, 2012]

[9] T. Mundt and J. Vetterick, "Network topology analysis in the cloud," in *ICOMP'11 - The 2011 International Conference on Internet Computing*, July 2011.

 ISBN: 978-1-61208-236-3