Performance Evaluation of using a Dynamic Shortest Path Algorithm in OLSRv2

Ulrich Herberg
Hipercom@LIX
Ecole Polytechnique
France
Email: ulrich@herberg.name

***Abstract*—MANET routing protocols are designed to scale up to thousands of routers with frequent changes of the topology. In preference, MANET routing protocols should also support constrained low-power devices. One of the bottlenecks of scalability in link-state routing protocols is the performance of the shortest path algorithm (e.g. Dijkstra). In this paper, we investigate the in-node performance of OLSRv2 and, in particular, study the benefits of using a dynamic shortest path (DSP) algorithm for this routing protocol. A DSP algorithm is an algorithm that adds or removes edges in the routing tree incrementally and calculates shortest paths, also incrementally. The performance in OLSRv2 with classic Dijkstra vs. DSP is evaluated, by comparing the CPU time for calculating paths in a large emulated network. Additionally, it is demonstrated that frequent topology changes due to mobility in MANETs lead to frequent routing table recalculations with only few routes updated each time. This property of MANETs makes the use of a DSP in OLSRv2 appropriate.**

I. Introduction

With mobile ad-hoc networks (MANETs) routing protocols becoming more mature, several big community MANET networks have emerged over the last few years. The Freifunk [1] and the Funkfeuer [2] community networks, in Berlin and Vienna respectively, each consists of about 400 routers. In a test, these networks have been successfully joined via a VPN connection, constructing a single MANET of about 1000 routers, and running OLSR [3]. The routers in these networks typically run on "constrained" devices with 200 MHz and 16 MByte of RAM. For allowing such networks to scale in size, it is important to reduce the CPU and memory requirements for the MANET routing protocol as much as possible.

Apart from control traffic handling, calculation of the shortest paths to all destinations in the MANET is costly in terms of CPU time. In particular, due to the ad-hoc nature of MANETs, the topology may be subject to frequent changes. Additionally, the destinations advertized in control messages are received incrementally on each router, and only a small number of addresses are advertized in each control traffic message. OLSR uses a variation of the well-known static Dijkstra algorithm for calculating paths to all destinations in the network. This implies that for each incoming HELLO or TC message with a new topology information, the previously calculated paths are deleted and a new tree is calculated on the router.

This paper suggests the use of a dynamic (i.e. incremental) shortest path calculation algorithm (DSP), and presents an evaluation of OLSRv2 [4] with such a DSP. The evaluation compares the CPU time for calculating shortest paths in a large emulated network between the standard static algorithm and a DSP. This paper shows that large-scale MANETs, such as the above-mentioned community networks, benefit by using a DSP algorithm in OLSRv2, and that the flexible structure of the OLSRv2 specification allows to do so without breaking compatibility with the specification.

A. Outline

The remainder of this paper is organized as follows: Section II presents related work, and section III analyzes the requirements of path calculation in OLSRv2. Section IV presents the methodology that is used for the evaluation. Section V details the results of the evaluation and discusses the consequences of the results. Section VI analyzes why MANETs are particularly well-suited for using DSP algorithms. Section VII concludes the paper.

II. Related Work

Extensive research about increasing efficiency of calculating the shortest path problem exists, notably by using incremental tree calculations. [5] compares 26 different algorithms for solving the shortest path problem. That survey concludes that DSP algorithms perform much faster then repeating static ones (such as Dijkstra). In addition to a theoretical comparison of the complexities of each algorithm, the survey provides experimental results of the required CPU time by each algorithm in a simulator, but not embedded in a real routing protocol. [6] is a similar comparison of dynamic shortest path algorithms, based on experimentation.

[5] suggests that the algorithm that proves to be the fastest in the survey is [7] when using the same data structures as in Bellman-Ford. [7] – henceforth called NST+ by using the first letters of the three authors – is an incremental shortest path algorithm. That means that the algorithm incrementally adds and removes new edges to and from the tree, instead of recalculating the whole tree for every new or removed edge. The paper proposes an algorithmic framework that allows to characterize a variety of dynamic shortest path algorithms including dynamic version of the well-known Dijkstra, Bellman-Ford, and D'Esopo-Pape algorithms, and shows their efficiency in simulations. [7] proves that the complexity of the algorithm

is $O(D_{Max}|\delta_d|^3)$ where D_{Max} is the maximum out-degree of a vertex of a given network, and δ_d denotes the set of affected vertices. NST+, as well as more recent dynamic shortest path algorithms (such as [8] and [9]) claim that using such an algorithm can be very efficient in link-state routing routing protocols like OSPF and IS-IS. However, their evaluation was not based on an OSPF or IS-IS implementation.

[10] describes an implementation of a DSP algorithm in OSPF based on the Quagga open source routing software. The authors show that their DSP algorithm performs better than the default routing algorithm that runs on a particular Cisco router.

However, to the best of the author's knowledge, no study exists for using DSP algorithms in mobile ad-hoc networks (MANETs) routing protocols such as OLSR [3]. Due to the dynamic natures of these networks, the topology typically changes much more frequently than in wired networks running OSPF. In addition, MANET routers usually have less CPU power and memory than routers running OSPF. This makes MANET routing protocols an ideal environment for using DSP algorithms.

III. Routing Set Updates in OLSRv2

The Optimized Link-State Routing protocol version 2 (OLSRv2) [4] is a routing protocol for use in mobile ad hoc networks. The protocol is developed by the MANET working group within the IETF. It contains the same basic mechanisms as OLSR [3], but adds additional features such as IPv6 compatibility, a more flexible message format [11], and simplifies certain mechanisms from OLSR (such as using multiple interfaces on a router). While the specification tries to guide the implementors very closely by detailed descriptions, it keeps the principle of "no-one-size-fits-all". As in MANETs very different deployments can be possible (static or mobile routers, different sizes of the network, different hardware for the routers, etc.), OLSRv2 allows freedom in order to facilitate different deployments. Amongst other, the choice of the routing calculation algorithm is bound by [4] only by:

> "The Routing Set MUST contain the shortest paths for all destinations from all local OLSRv2 interfaces using the Network Topology Graph. This calculation MAY use any algorithm, including any means of choosing between paths of equal length."

An example of such algorithm – a variation of Dijkstra's algorithm – is presented in the appendix of [4].

This means that using a DSP algorithm does not break with the OLSRv2 specification, as long as it provides shortest paths to all destinations given by the topology.

IV. Test Methodology

This section describes the methodology used for evaluating the performance of using a DSP algorithm in OLSRv2.

A. Emulator for large Topologies

In order to create a large topology for testing the scalability of the path calculation algorithm, an OLSRv2 topology emulator has been built, presented in [12]. The basic idea is illustrated in figure 1. Note that the emulator runs on *real* machines, i.e. the addresses depicted in the figure are bound to a network interface in the operating system. In figure 1, a router is depicted with the address 10.0.0.1. This is the router running the OLSRv2 implementation to be evaluated. On the right side, several routers are shown with, in this example, IP addresses from 10.0.0.2 to 10.0.0.5. These routers shall represent the 1-hop neighbors of 10.0.0.1. These four routers are not running on four different machines, but on a single machine with a single interface with four different IP addresses. The emulator running on this machine creates HELLO messages with using these four IP addresses as source address in the IP header, and thus emulates four direct neighbors of 10.0.0.1. Note that the HELLO messages already include 10.0.0.1 in the list of symmetric neighbors, so that the link is symmetric starting from the first HELLO message. In addition, forwarded TC messages of "virtual" routers are created. These "virtual" routers represent routers that are at least two hops away from 10.0.0.1, but are not bound to an IP address on the emulator machine. The four 1-hop neighbors create TC messages and pretend that these originate from routers further away. For example, such a TC message might have a hop count of 4 (i.e. four hops away from 10.0.0.1), and any message originator address different from the four direct neighbors.

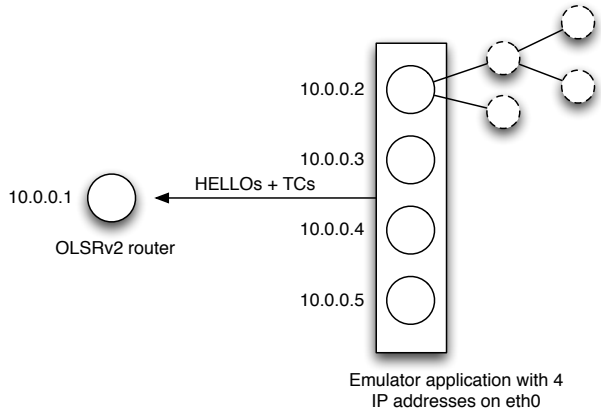

Figure 1. Emulator for creating arbitrarily large OLSRv2 topologies

The emulator thus represents arbitrary topologies to the router under test. An applet has been implemented that allows for creating topologies. In the Java applet, the user can "draw" routers by clicking on an area and creating edges between two routers. Additionally, random graphs using the Eppstein Power Law or Erdos-Renyi can be created in the applet. As these graphs might be separated into several connected components, the applet connects these components randomly. Another feature is to import NS2 [13] tcl files representing a static scenario (i.e. not including any router movements). After having created the graph, a spanning tree with the router 10.0.0.1 being the root is calculated, and the emulator can

send HELLOs and TCs that correspond to the topology of that graph. Currently, the emulator only allows to use static graphs, i.e. it allows to test incrementally adding edges in the routing tree, but not yet removing edges. In a future work, it is planned to extend the emulator for allowing to emulate dynamically changing graphs.

Topologies created with the emulator can easily contain several hundreds of thousands of routers.

B. OLSRv2 Implementation

For the evaluation, OLSRv2, implemented in Java [12] (called JOLSRv2), has been used. This implementation is a fully-compliant implementation of the current specification of OLSRv2, i.e. [4], [14] and [11].

JOLSRv2, per default, implements the shortest path algorithm (a variation of the standard Dijkstra algorithm) proposed in the appendix of [4]. In order to compare the standard algorithm with the DSP algorithm, the Java class representing the routing set has been specialized and the NST+ algorithm [7] has been implemented. NST+ was chosen as an instance of a dynamic shortest path algorithm; other such algorithms (as for example [9]) could be used for an evaluation as well. Due to their similar properties (i.e. dynamically adding or removing edges), it is supposed that the order of magnitude is similar.

C. Time Measurement

In order to compare the static shortest path algorithm with the DSP algorithm, the accumulated time for calculating the routes is measured in each respective algorithm. Whenever the method for updating the routing set is entered, the time is saved in a variable, and the difference of time when leaving the method is added to a cumulative time counter. This, naturally leads to some inaccuracies, since JOLSRv2 is a multi-threaded application, i.e. other time-consuming threads might run in the background. However, in order to reduce the degree of inaccuracies, the following settings have been used:

- The validity time of HELLO and TC messages is set to a very large number, i.e. Link Tuples and Router Topology Tuples do not expire during the experiment, and thus no further actions (e.g. tuple expiration or Routing Set recalculations) have to be performed by the router.
- JOLSRv2 allows to disable sending TC and HELLO messages on a router. During the experiment, the router was not sending any control messages – neither periodic nor triggered – but only listening to the HELLO and TC messages from the emulator.
- The emulator sends the control messages only once. Thus, once the messages are received and processed by the OLSRv2 router, no further messages have to be processed. Thus, no CPU time is needed for further message processing.
- No other time-consuming processes run on the machine.
- The measurement is averaged over 20 runs.

V. Evaluation

This section presents the results of the evaluation. Figure 2 depicts the average time consumption for calculating the routes to all destinations in the emulated network. Random graphs have been created using the Eppstein Power law, from 10 to 10000 routers. The JOLSRv2 routing protocol implementation was tested on an Intel Core2 with 2.13 GHz and 4 GB of RAM with no other time-consuming processes running in the background.

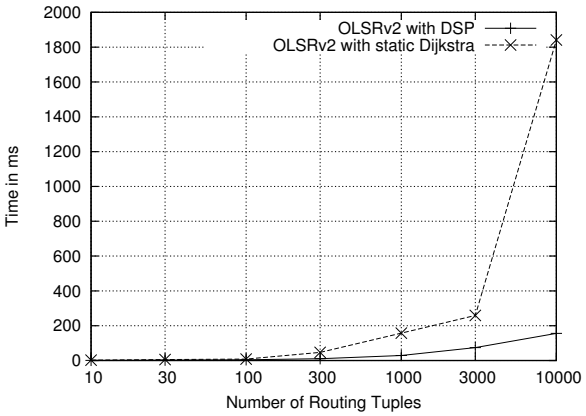

Figure 2. Results of the simulation: Accumulated time for calculating routes to all destinations

As can be seen in figure 2, for less than 1000 routers, the difference of calculation time between the static Dijkstra and the DSP algorithm are negligible. Starting with about 1000 routers, the static algorithm takes significantly longer than the DSP which stays at very low values (in average 156 ms for 10000 routers in the given topology).

It can be assumed that in C++ implementations with intelligent memory management, the time consumption could be further reduced as compared to the Java implementation that was used in this comparison. Further improvements can also be achieved by using more recent DSP algorithms than NST+. For example, [9] claims to be up to three times faster than NST+.

VI. Incremental Route Updates

In section V, it has been shown that using a DSP algorithm for calculation of the shortest paths in OLSRv2 is beneficial for large networks. Since DSP algorithms incrementally add or remove edges in the shortest path tree instead of recalculating the whole tree for every Routing Set update, they are particularly suitable for networks with many changes of the topology, such as in mobile ad hoc networks. In static networks or in networks with very few changes of the topology (such as typically in OSPF or IS-IS networks), DSP algorithms are less advantageous over static Dijkstra than in mobile networks.

This section will analyze the effect of mobility on the number of Routing Set updates in OLSRv2 and the number of routes that are changed in each Routing Set update.

A. Evaluation Settings

The JOLSRv2 implementation, that has been presented in section IV-B, not only runs on real operating systems such as Linux and Windows, but also in the network simulator NS2 [13], using a tool named AgentJ [15] that allows to use Java routing protocol implementations within NS2 without modifying the source code. AgentJ supports the full network stack of Java and provides translators for addresses, timers and threads. The resulting combination means that preexisting Java routing protocols, which run on the Internet and are deployed on operating systems, such as Linux or Mac OS, will work unmodified within NS2.

In order to evaluate the nature of Routing Set updates in MANETs using OLSRv2, a simulation with JOLSRv2 has been performed, using the settings as presented in table I.

Parameter	Value
NS2 version	2.34
Mobility scenarios	Random way point
Grid size	1000m by 1000m
Number of routers	40
Communication range	250m
Pause time	0 secs
Router velocity	0 to 30 m/s (constant)
Radio propagation model	Two-ray ground
Simulation time	100 secs
Iterations	20 times
HELLO Interval	2 secs
TC Interval	5 secs
Interface type	802.11b
Frequency	2.4 GHz

Table I
SIMULATION SETTINGS

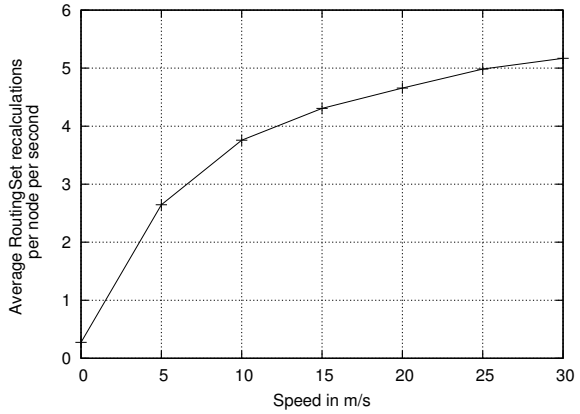

Figure 3. Average Routing Set recalculations per second

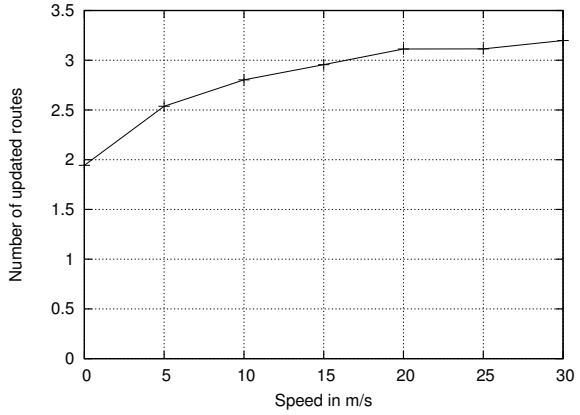

Figure 4. Average number of updated routes per Routing Set recalculation

In the evaluation, only updates to the Routing Set that change (i.e. add, delete or modify) at least one Routing Tuple are counted. In addition to the number of Routing Set updates during the simulation, the number of changed routes in each Routing Set update is also considered.

B. Results

This section presents the results of the network simulation. Figure 3 depicts the average number of Routing Set recalculations in OLSRv2 (that change at least one route) per router per second. As can be seen, for static networks very few updates are performed. This is due to the fact that only at the beginning of the simulation, the Routing Set is updated. Once the network has converged, no more updates are performed. The more mobile the network is, the more Routing Set updates have to be performed due to the changing topology. Figure 3 shows that for mobile networks, several updates of the Routing Set are performed per second, increasing with higher mobility.

Figure 4 depicts the average number of updated routes in each of the Routing Set updates. Slightly growing with mobility, only very few routes are modified each time the Routing Set is updated. This is beneficial when using an incremental DSP algorithm which keeps the previously unchanged routes in memory, and only adds or removes edges.

Figure 5 presents the number of control messages that are sent by each router in average over the simulation time of 100 seconds, both triggered HELLOs and TCs and total HELLOs and TCs (i.e. triggered plus periodic). The amount of control traffic is increasing with the mobility of the routers due to the triggered messages, which are emitted each time a router senses a change in its symmetric neighborhood (for triggered HELLOs) and changes in its MPR selectors (for triggered TCs). Each incoming control message on a router can potentially lead to a Routing Set update if the topology has changed.

Figure 6 depicts the number of advertized neighbors in each control message. Each of such advertised neighbors has potentially to be considered when recalculating the Routing Set if the topology has changed. Due to the MPR mechanism of OLSRv2, TCs messages only advertize a subset of the emitter's neighbors in order to reduce redundant transmissions. As can be seen, the number of advertized neighbors in TCs is rather small, not significantly growing with mobility (since

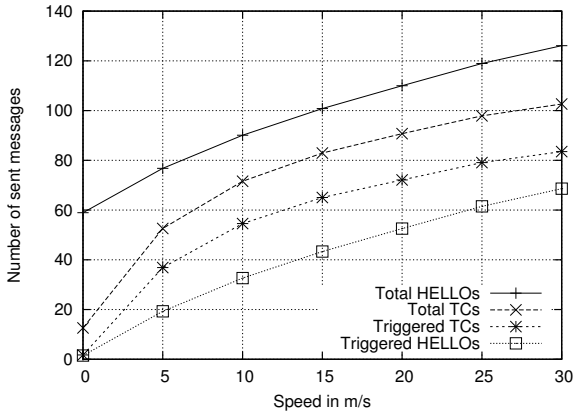

Figure 5. Control traffic emitted by a router

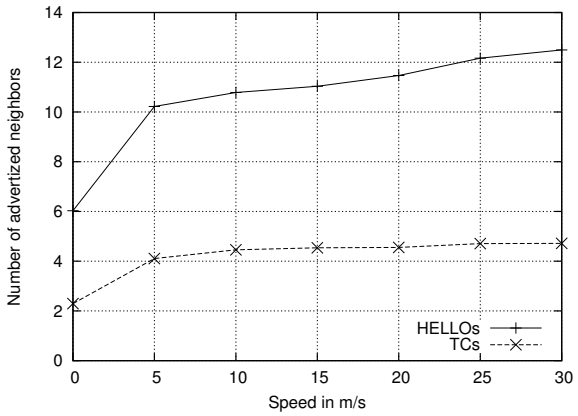

Figure 6. Number of advertized neighbors per control message

the total number of routers stays the same when increasing mobility). Due to the transmission of triggered incremental TC messages in OLSRv2, the number of sent control messages grows from a static to a mobile scenario.

In summary, the topology of mobile ad hoc networks frequently changes with increasing mobility of routers. While routers send many control messages, only a small number of destinations is advertised in each such message. Consequently, a router must often recalculate its Routing Set, but only few routes will change in each recalculation. This is beneficial for incremental shortest path calculations which only consider changed edges, instead of recalculating the whole tree. Note that in scenarios with extremely high mobility, it is possible that most edges have to be modified in the routing tree – in this case, using a DSP may not be more efficient (or even less efficient) than recalculating the tree from the scratch using a static Dijkstra [7]. The investigation of such high mobility scenarios is out of the scope of this paper.

VII. Conclusion

In this paper, it has been experimentally shown that the calculation of the routing table in the MANET routing protocol OLSRv2 is able to scale up to tens of thousands of routers when using a dynamic shortest path algorithm (DSP). A comparison has been performed between the static routing algorithm from the appendix of the OLSRv2 specification and a dynamic shortest path algorithm. The DSP takes significantly less CPU time for calculating the paths, in particular for bigger networks with more than a few hundred routers. Due to the dynamic nature of MANETs, edges are incrementally added by means of control messages and change frequently. This beneficial for the DSP, since it allows incremental updates from the previously calculated graph.

Current community MANETs such as the Funkfeuer and Freifunk networks reach this number of routers in their networks running OLSR. In addition, their routers are often low-power devices with less than 200MHz and very limited amount of memory. Thus, it is crucial to reduce the CPU time for the path calculations in order to guarantee the scalability of these networks. This paper not only provides experimental justification of the efficiency of DSPs on a real OLSRv2 implementation, but also argues that due to the flexible nature of the OLSRv2 specification, full compatibility is kept when using a DSP algorithm instead of the provided algorithm in the specification.

References

[1] "Freifunk website," http://www.freifunk.net.
[2] "Funkfeuer website," http://www.funkfeuer.at.
[3] T. Clausen and P. Jacquet, "Optimized link state routing protocol (OLSR)," Oct 2003, experimental RFC 3626.
[4] T. Clausen, C. Dearlove, and P. Jacquet, "The optimized link-state routing protocol version 2," Oct 2009, Internet Draft, work in progress, draft-ietf-manet-olsrv2-10.txt.
[5] S. Taoka, D. Takafuji, T. Iguchi, and T. Watanabe, "Performance comparison of algorithms for the dynamic shortest path problem," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. E90-A, no. 4, pp. 847–856, 2007.
[6] C. Demetrescu and G. F. Italiano, "Experimental analysis of dynamic all pairs shortest path algorithms," *ACM Trans. Algorithms*, vol. 2, no. 4, pp. 578–601, 2006.
[7] P. Narvaez, K.-Y. Siu, and H.-Y. Tzeng, "New dynamic algorithms for shortest path tree computation," *IEEE/ACM Transactions on Networking*, vol. 8, no. 6, pp. 734–746, Dec 2000.
[8] B. Xiao, J. Cao, Q. Zhuge, Z. Shao, and E. Sha, "Dynamic update of shortest path tree in OSPF," in *Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*, May 2004, pp. 18–23.
[9] B. Xiao, J. Cao, and Q. Lu, "Dynamic SPT update for multiple link state decrements in network routing," *Journal of Supercomputing*, vol. 46, no. 3, pp. 237–256, Jan 2008.
[10] V. Eramo, M. Listanti, and A. Cianfrani, "Implementation and performance evaluation of a multi-path incremental shortest path algorithm in quagga routing software," in *Proceedings of the 7th International Workshop on Design of Reliable Communication Networks*, Oct 2007.
[11] T. Clausen, C. Dearlove, J. Dean, and C. Adjih, "Generalized MANET packet/message format," Feb 2009, standards track RFC 5444.
[12] U. Herberg, "JOLSRv2 – an OLSRv2 implementation in Java," in *Proceedings of the 4th OLSR Interop workshop*, Oct 2008.
[13] K. Fall and K. Varadhan, "The network simulator – NS-2," http://www.isi.edu/nsnam/ns.
[14] T. Clausen, C. Dearlove, and J. Dean, "MANET neighborbood discovery protocol (NHDP)," Oct 2009, Internet Draft, work in progress, draft-ietf-manet-nhdp-11.txt.
[15] U. Herberg, "Integrating Java support for routing protocols in NS2," 2009, INRIA research report 7075.