Direct Execution of OLSR MANET Routing Daemon in ns-3

Evgeni Bikov
Institute for Information Transmission Problems
Russian Academy of Science
Bolshoy Karetny per. 19, Moscow, 127994,
Russia
bikovevg@iitp.ru

Pavel Boyko
Institute for Information Transmission Problems
Russian Academy of Science
Bolshoy Karetny per. 19, Moscow, 127994,
Russia
boyko@iitp.ru

ABSTRACT

Direct code execution framework recently developed for ns-3 is tested running open source OLSR implementation. It is shown that current DCE version is capable to run the unmodified production quality MANET routing daemon. Native ns-3 OLSR model is calibrated against DCE with rather didactic results. Finally, large-scale topologies are used for model and DCE comparison. The conclusion about the applicability areas of modeling and direct code execution are given.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design - Wireless Communication

Keywords

Direct code execution, MANET, ns-3

1. INTRODUCTION

Within last years MANET systems are becoming more and more popular.The deployment and the debugging of protocols on a real network can be rather complicated if large networks are considered. That is why simulation is an important tool for improving and validating protocols.The key problem in simulating MANET networks is a great complexity of the internal structure of link- and network-layer protocols. There are several basic directions for reducing such complexity. The first one implies simplification of every mechanism. Systems based on state machines are often used in this area. It's only good for approximate studies of fundamental ideas when developing new protocols. The second way is to model each subsystem element by element. It demands many efforts of the whole community of developers but the results are worth it. The validation of proposed models is the greatest challenge in this area. The third idea involves replacing key elements of the whole system with their real-world analogs. Such method eliminates most of the problems with validation and verification of models. It is considered to give much more accurate and reliable results. In this paper we check this statement and outline areas of applicability of direct code execution and convenient modeling.

This work presents the comparison of two ways of simulating the well known OLSR protocol [1]. The first one is based on using OLSR model (hereinafter referred to as "Model") within ns-3 network simulator [2]. Ns-3 is as an eventual replacement of the popular ns-2 simulator. OLSR model in ns-3 was ported from its predecessor. The second variant consists in direct code execution of the production quality open source OLSRd daemon [3] – an ad hoc wireless MANET routing daemon. It is heavily used by the Freifunk networks in Austria. Since the pure source code of the functioning prototype is used in this case, verification and validation are replaced by model calibration.

Both the projects are said to be based on RFC for OLSR [1]. The problem worth mentioning here comprises the fact that real life implementations of RFC can vary significantly while still following recommended concepts. The key differences come from the legal RFC variation and ambiguity of the description of algorithms. We find such differences rather instructive, so typical variations in implementation are considered at greater length in Section 4. Most of them occurred when DCE (executed source code from [3]), Model (from ns-3 kit) and OLSR RFC document were being compared.

The remainder of this paper is organized as follows. The related work and motivation for our research are given in Section 2. Section 3 presents the setup of the conducted survey. The small-scale comparison and calibration of two models are described in Section 4. The large-scale comparison and more detailed experiments are presented in Section 5. Our conclusions and recommendations can be found in Section 6.

The contributions of this paper are threefold. Firstly, it overviews the applicability of the direct code execution technique for various development tasks like tweaking and validation of models. Secondly, it focuses on the comparison of modeling results from ns-3 built-in model and those obtained by the direct code execution of protocol. Thirdly, the paper advances our understanding of the ways to increase accuracy of modeling.

2. RELATED WORK

Discrete-event simulation is basic and the most popular method for network simulation, see, e.g. [4], [5]. See [6] for additional overview of MANET simulators and related

WNS-3 2011, March 21, Barcelona, Spain

DOI 10.4108/icst.simutools.2011.245558

techniques.

To conduct a simulation experiment one needs to build a model of the system under consideration (in our case – network). The main issue in this area is to define a set of key assumptions to simplify the system and design its model. After successful design, the model must be validated appropriately. Each phase of the model implementation must be closed with the verification; see e.g. [7] for a thorough discussion of validation and verification problems. The main limitation of this method consists in the fact that even after passing through all these stages, the model may still not be sensitive enough to reproduce some feature of the real system behaviour. The use of this technology demands deep understanding of the problem under research before the research itself. Thereby, in some cases the preparational stage of modeling becomes too difficult. The credibility of the simulation results is the problem stated more than five years ago (see e.g. [8]). It's still relevant.

Models based on some specification may lack accuracy in another reason. In some aspects the protocol behaviour may depend upon implementation itself. For instance, it can be tricky to replicate some thread-specific scenario or memory bottleneck in a discrete-event model. There are several approaches to overcome these difficulties.

The first approach is to implement a protocol model at the stage of implementing the protocol itself. In this case the interpretation of a protocol specification isn't arbitrary. On the other hand, such project is hard to maintain. Two associated parts must be constantly kept synchronized. Any modification in the base protocol leads to the need of an additional review of the model.

The second approach consists in manual, automatic or semi-automatic patching of source code of the existing protocol implementation to make possible its further usage within some discrete-event simulation system. Paper [9] presents convincing results of executing slightly modified versions of real protocols within DaSSF [10]. About 3.8 % of the initial source code is said to be modified. Global variables are often encapsulated for this method. System calls such as file, socket and time operations need to be replaced with proper functions. This approach is more appropriate for production needs, but still lots of efforts need to be applied to maintain rapidly developing projects.

Modifying 4 % of the source code, where most changes are stated to be made for the encapsulation, seems to be a solvable task for autopatching. The Network Simulation Cradle project [11] seems to be successful enough for us to conclude that this method can be relied upon. Nevertheless this technique is rather complicated. Probably, it can facilitate the protocol porting for its designer. In other cases, where there is a need to port the preexisting code, this method seems to be unreliable.

There is a way to avoid encapsulation and direct system calls reimplementation. The idea is to integrate a discrete-event simulator with real hardware. In this case the simulator must contain most of the basic network primitives to reach this goal, so only highly developed systems should be considered. In paper [12] most of such systems are considered and compared. One of the proposed variant is to use ns-3 simulator.

In common research scenarios a number of nodes can vary significantly, so building a testbed can become too expensive. In such a case it is more reasonable to use virtualization technology to create hardware implementations, called virtual machines, that execute software as if they were real hosts. This idea is probed by NS community and proved to be rather useful. Using Linux containers as well as Qemu virtual machine based networks is considered to be promising. The key limitations here consist in difficulties in debugging and monitoring the whole system. Also, this method requires more efforts during a deployment process, not saying about serious software requirements. The architecture for combining a discrete-event simulator with virtual machines is presented in paper [13].

Finally, there is another method worth highlighting. It's in some way similar to code patching, but the initial source code stays unmodified. It is referred to as direct code execution. The method is based on the idea of executing basic source code of a protocol with redirecting its system calls externally. Preparatory measures consist in compiling a position independent binary executable file from the source code and reimplementing key functions within the simulator. The related work of researching on-demand routing protocols was conducted in [9], but still the source code was, technically speaking, ported to the simulator. In this work we use the direct code execution system within ns-3 simulator, developed recently by Mathieu Lacage from INRIA [14]. The similar technique was used by Hajime Tazaki. The work was presented on WNS3 March 2010. The author had successfully executed Zebra with the self-written extension for OLSR. The implementation mentioned is based on the concept of addressing dynamic library files for system calls substitution at runtime. The key advantage of such an approach consists in the fact that the source code of protocol stays unmodified. In this work we execute OLSR MANET routing daemon [3] using this system.

3. SETUP

3.1 Simulation parameters

We use a research branch of Network Simulator 3 supporting direct code execution to run all the experiments presented in this paper. The source files for simulator version we used can be taken here http://code.nsnam.org/mathieu/ns-3-dce/. The latest version 0.6 of the OLSR routing daemon was downloaded from [3].

We will present two series of experiments: small-scale and large-scale. The default configurations for large-scale scenarios are presented in table 1. The small-scale topologies will be described in Section 4. As small-scale experiments were conducted mostly for calibration and validation of two systems, mobility models and side applications were not involved for this set.

The number and density of nodes were chosen to guarantee stable connectivity between nodes and the presence of multi-hop routes according to [15]. The specified WiFi model was preferred as the most basic and easy to use. For investigating the protocol performance in dynamic conditions experiments were conducted with the mobility model involved. VoIP application model was used as the user-level detector of the routing performance, see Sect. 5 for more details. The default values of OLSR parameters defined by RFC [1] were used, see Sect. 4 for the detailed discussion of OLSR configuration.

3.2 DCE setup

Setup parameter	Value
Transmission range	$R = 178$ m
Simulation area	6.6 $R \times 6.6$ R
Number of nodes	100
Simulation duration	250 s
Transient period	100 s
Mobility model	Random waypoint
Initial position	Steady state RWP distribution
Average node speed	20 m/s
Average pause time	10 s
Confidence level	90 %
WiFi standard	IEEE 802.11a
Propagation loss model	Fixed range
Max. MAC retry counter	7
Rate control algorithm	AARF
VoIP packet payload	20 bytes
VoIP packet interval	20 ms

Table 1: Simulation setup common for all large-scale experiments

OLSRd daemon was configured to work in RFC-compatible mode with default parameter values. Minimal OLSRd configuration file used is shown in Listing 1.

```
DebugLevel      0
IpVersion       4
FIBMetric       "flat"
UseHysteresis   no
LinkQualityLevel 0
Pollrate        0.05
TcRedundancy    2
MprCoverage     1

Interface "ath0"
{
  HelloInterval       2.0
  HelloValidityTime   6.0
  TcInterval          5.0
  TcValidityTime      15.0
  MidInterval         5.0
  MidValidityTime     15.0
  HnaInterval         5.0
  HnaValidityTime     15.0
}
```

Listing 1: OLSRd configuraion

Since production quality OLSRd routing daemon is a complex software a large number of system calls were reimplemented to make it work under ns-3 DCE framework. The main purpose of reimplementing system calls consists in isolating different copies of OLSRd processes with their attributes from each other. One should also take into account the fact that protocol implementation can include multithreading, which is hard to combine with the logic of the discrete-event simulator. Thereby direct code execution framework has to provide services similar to aprocess scheduler of kernel. The operations with file descriptors were reimplemented to avoid intersections while accessing common files and streams. Then time handling operations come. This stage mostly responds to coupling the protocol with the scheduler of the simulator. This is definitely not the whole list. Moreover, there are some functions that may be considered safe, but still need to be encapsulated. As an example, `random` system function was reimplemented for repeatability of results within the simulator.

It took many efforts to reimplement system functions to make OLSRd work in the direct code execution environment, but the main point here is that reimplementation is needed only once. All our changes were contributed to the ns-3-dce branch [16]. After some efforts DCE framework will be multipurpose enough for executing other routing daemons (and other applications in general) without special preparations.

4. MODEL CALIBRATION

At this stage we are going to compare the operation of OLSR model from default ns-3 kit [2] ("Model") and the direct execution of OLSRd routing daemon [3] ("DCE") Both of them are said to be written in compliance with OLSR specification [1]. Model is the simplification of the real protocol, and DCE is based on the real production quality implementation. Our goals are to compare two methods of simulating behaviour of OLSR, to investigate the level of accuracy of the results and to determine the range of use for DCE and Model. We'll make an attempt to distinguish the most significant differences between the protocol implementations and try to equalize them.

4.1 Observables and "out of box" results

Our initial strategy is to compare unchanged "out of box" versions of DCE and Model with simple topologies and calibrate them if needed. The small-scale topologies used are presented in Figure 1. At the small-scale stage we have an ability to analyse the protocol operation up to the smallest details. Beside values presented below, PCAP files for each node were analysed. The content of separate packets were also investigated for comparison.

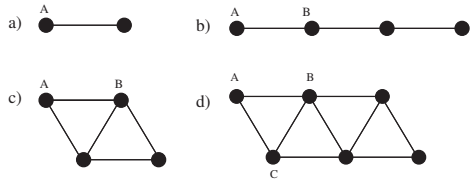

Figure 1: Topologies used for Model and DCE calibration

Nodes are static in all the experiments.

We have identified three observables describing OLSR behaviour and measured them in all the experiments.

Firstly, since route tables are the main product of OLSR operation, they were compared precisely for all nodes using both OLSR implementations. It was observed, that DCE and Model routes appeared to be equal for simple cases. The lengths of the routes and the accessibility of the nodes were investigated for convenience. The results reliably confirm that both systems are capable of building right routes.

Secondly, since the simulated network is based on WiFi equipment, the average packet rate and the average packet

size are considered to be valuable characteristics of the system. There wasn't any additional user-level traffic in the system. The information was gathered by analysing PCAP-files data for the steady state period. The results are presented in Table 2. As can be seen, even for the simplest topologies (without mobility and additional payload) the differences between the simulation results reach more than 100 % of the absolute value for the most of the readings. The causes for such a discrepancy will be discussed in the next section.

Finally, the content of the packets was also analysed to compare the internal mechanisms of the packet completion. Differences occurred in the packet structure and message aggregation logic. They will be considered in details in the next section.

Node	Mean packet size, B		Mean packet rate, s^{-1}	
	DCE	Model	DCE	Model
Topology a)				
A	85.68	120.88	1.15	2.26
Topology b)				
A	89.88	157.78	1.16	2.77
B	89.74	126.75	1.97	3.20
Topology c)				
A	87.93	124.18	1.94	3.27
B	114.96	135.69	2.93	4.82
Topology d)				
A	115.89	132.99	1.94	3.17
B	103.55	214.26	3.31	6.35
C	140.18	217.51	3.66	8.34

Table 2: "Out of box" results, topologies and nodes are shown in Fig. 1

Consider the following as a preliminary conclusion for this section. Built-in models from the simulator kit must not be blindly believed to predict the behaviour of the real world protocol implementation. The models definitely need to be at least calibrated for better predictions.

4.2 Implementation differences

It is evident from Table 2 that there are significant differences between DCE and Model. In this section we investigate causes of such a discrepancy. To do so we have analysed source code for both DCE and Model. The projects were also compared with RFC at the same time. The attendant goal of this stage is to eliminate the most flaring differences between DCE and Model for further investigation. Below there is a list of the most essential discrepancies discovered.

4.2.1 Defaults

It is worth noting that the default values for DCE and Model were not equal for out-of-box kits. Basic constants are mostly the same, but some internal values were noticed to be different. For instance, the limits of message aggregation and duration of data validity were unequal. Such a discrepancy between DCE and RFC specification can be explained by authors' efforts to optimize the performance. The values were equalized for further research.

4.2.2 Message jitter calculation

This example describes a didactic case, when the usage of equal principles with the combination of different architecture concepts leads to unexpected results. RFC [1] defines the default period for sending TC messages to be 5 seconds. It also proposes to use the standard mechanism of avoiding synchronization between sending nodes by introducing jitter. Both the implementations use the following formula to calculate an interval between transmissions

$$actualInterval = TcInterval - jitter, \qquad (1)$$

where $jitter$ is uniformly distributed in $[0, MAXJITTER]$. The difference between DCE and Model consists in the following fact. In Model this formula is used to reduce global timestamps in the scheduler (time from the beginning of the simulation). But in DCE it is used to reduce the local value of the *interval between broadcasts*. So within Model this mechanism just moves all the timestamps in one direction keeping the default average interval between them, when in DCE it brings them together reducing the interval. This difference leads in our case to 13 % deviation of the expectation value for TC interval. The effects of such an order are considered to cause a significant invalidity of model predictions even in small-scale experiments. Note that both the implementations formally follow RFC, moreover, the calculations are written in an equal form. Such kind of deviations are hard to detect or avoid. They arise from different nature of the simulator and real protocol frameworks. This problem might be frequently encountered.

In our case we decided to correct this difference by modifying DCE. The method for calculating the broadcast interval was changed to equate its expectation value to default 5 seconds.

4.2.3 Packet structure

DCE and Model appeared to have different principles of building packets for transmission. RFC for OLSR defines the ability of grouping messages in packets, but doesn't clearly explain all the cases when the designer of the protocol should use it. As the result, Model has just the basic support of message aggregation. DCE, as a successfully deployed and optimized protocol, has an extensive support of grouping messages. It does not just stack messages in packets but regroups their content for further resource economy. This difference immediately leads to a significant discrepancy between overhead ratios.

Model was expanded to support the production level mechanism of aggregation.

4.2.4 Transmission logic

Originally DCE and Model had different mechanisms for initiation of packet transmission. In Model any constructed message initiate scheduling of packet transmission. This leads to unnecessary growth of the packet rate with reducing of the average packet size. DCE developers have optimized this mechanism for reducing overhead. Maximum polling rate is fixed for this reason.

Model were expanded to support similar logic of polling.

4.2.5 Miscellaneous

There are some extra options that can differ the behaviour of two systems. DCE has a function of the topology information redundancy. It is defined in RFC document [1] but is omitted in Model. Another feature is MPR optimization. It's a mechanism preventing a node from broadcasting if it hasn't any MPR selectors. MPR optimization is omitted in

DCE implementation. In both cases efforts were taken to eliminate these differences.

4.3 Calibration results

As the result of the calibration efforts both DCE and Model were modified to implement similar functions and key mechanisms. New results of the performance comparison of DCE and Model are presented in table 3. The same scenarios were chosen to demonstrate the confident similarity of the behaviour at a small-scale.

Node	Mean packet size, B		Mean packet rate, s^{-1}	
	DCE	Model	DCE	Model
Topology a)				
A	99.44	99.43	1.02	1.02
Topology b)				
A	116.33	116.31	1.03	1.02
B	121.91	121.88	1.54	1.54
Topology c)				
A	121.20	116.21	1.52	1.53
B	117.13	114.37	2.03	2.03
Topology d)				
A	142.53	128.52	1.52	1.53
B	150.26	136.96	2.03	2.03
C	150.15	132.45	2.54	2.54

Table 3: Results after calibration, topologies and nodes are shown in Fig. 1

The average packet size values differ slightly. This fact deals with the difference between the internal mechanisms of generation and aggregation of packets. Note that in simple topologies the performance of two models are practically equal. The route tables and accessibility were verified at the same stage for the better confidence. Average packet rates stay close to each other with the increase of the quantity of nodes. This confirms the fact, that both the timplementations were successfully modified to have a similar logic of packet transmission scheduling. Probably, there are cases when such modifications are not available or reasonable. Using pure models based on the direct code execution technique seems adequate for such scenarios.

To conclude this section we state that by close analysis of the source codes of both the projects it appeared possible to calibrate them to a satisfactory level. Note that it's rather a convenient method for validation and extension of existing models. In the next section a large-scale comparison of Model an DCE will be discussed.

5. COMPARISON

Model and DCE calibration presented in the previous section was made on the static small-scale topologies without any background traffic. In this section we proceed with presenting analogous results obtained in the large-scale mobile scenario with background traffic. Scenario details are similar to those in Section 4. They are given in Table 1.

The average length of the known routes and the average ratio of the known destinations as the function of time are presented in Fig. 2. The transient behaviour study serves two purposes: the determination of a steady period for the further comparison and investigating of the transient period of OLSR operation. Differences at this stage can be insightful due to their extreme difficulty of modeling. Despite the fact that this behaviour is not demonstrative enough for making conclusions about the network steady state, variations at this stage can indicate an internal difference, that can after all lead to noticeable effects.

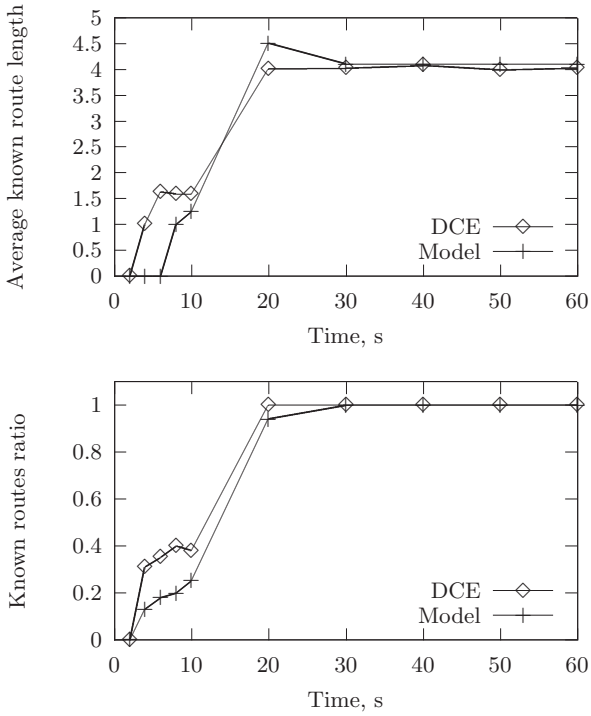

Figure 2: Top: average length of known routes, bottom: average known route ratio as the function of time. Statistical errors are within the symbols

From Fig. 2 we consider $[0; 50)$ seconds as the transient period and $[50; \infty)$ seconds as the steady state period for OLSR operation. The average length of the known routes at a steady state is estimated as 4.10 ± 0.05 for DCE comparing to 4.00 ± 0.05 for Model (90% confidence levels are given). Since 90% confidence intervals do not intersect, we state that internal algorithms in two implementations can differ slightly. They both solve their main purpose of finding the shortest route. Both the implementations find all available routes after 30 seconds of the operation.

The most significant differences between DCE and Model were detected at the stage of analysis of management traffic volumes on large-scale topologies. The average packet size and transmission rate of OLSR management data per node in a *static* random topology are shown in Table 4. Noticeable differences between packet sizes can be explained by a different internal limit for the message aggregation or algorithms of scheduling. A more significant result consists in the variation of total data transmission rates. Differences of this sort can lead to inaccuracy of the results of modeling application performance in the network with a particular routing protocol.

	DCE	Model
Average data rate, kB/s	9.70(25)	4.17(13)
Average packet size, kB	1.48(2)	0.87(3)
Average packet rate, packets/s	6.40(8)	4.83(3)

Table 4: Average OLSR management data rate per node in static topology. Statistical error in the last digits is given

In Table 5 the similar data measured in mobile scenario is presented. Differences between two implementation still exist, but the interesting fact here consists in the degree of changes between static and dynamic topologies. In the case of DCE mobility leads mostly to the packet rate increase while in the Model the packet size grows first. This fact puts forward another argument for doubts about the credibility of models, based on the protocol specifications. Such clear-cut distinctions in the characteristics of broadcast transmissions are considered to be significant enough to influence the results for the related applications performance.

	DCE	Model
Average data rate, kB/s	17.9(7)	14.6(3)
Average packet size, kB	1.6(1)	2.46(3)
Average packet rate, packets/s	11(4)	5.8(1)

Table 5: Average OLSR management data rate per node in dynamic topology. Statistical error in the last digits is given

All the previous results were obtained in the absence of network traffic apart of OLSR management traffic itself. The next experiments were conducted in the scenarios with sparse or dense VoIP traffic. The topology was chosen to be static. The scenario details are presented in Table 1. This research emerged to check if dissimilarity of service packet floods with internal mechanisms for two implementations significantly affects final evaluations. Source and destination of VoIP flows were chosen randomly and do not change. The periodic traffic flows in both the ways. We measured the packet delivery ratio (PDR), the average delay and the jitter[1] for packet delivery sessions. Since the most practical and frequent reason for estimations in this area is determining the maximum number of simultaneous streams available in the network (VoIP capacity), this parameter will be varied through the experiments. The corresponding results are presented in Table 6.

As can be seen from Table 6, the performance characteristics for two ways of modeling differ slightly for the unsaturated channel. Probably, both the sets of data can be considered suitable for estimations for the deployment of weakly loaded network. At the same time the results for DCE and Model in the saturated network differ significantly. The differences between the forms of managment traffic affect the results of the simulation. Moreover, the correlation between corresponding values does not remain constant. The uncertainty in the capacity results can lead to serious problems if simulation is used for network deployment analysis.

[1]VoIP jitter is defined in [17], do not confuse with OLSR jitter discussed in Sect. 4.2.2

	DCE	Model
Unsaturated		
Average delay, ms	5.6(5)	4.0(7)
Jitter, ms	3.1(1)	3.3(1)
Packet delivery ratio	0.96(9)	0.96(9)
Saturated		
Average delay, ms	10.7(5)	18.2(5)
Jitter, ms	5.3(1)	8.3(1)
Packet delivery ratio	0.94(9)	0.76(9)

Table 6: VoIP performance, "unsaturated" – number of VoIP flows is less than network capacity, "saturated" – number of VoIP flows is close to the capacity

In Figure 3 the simulation run time as the function of a number of nodes is presented, the node density was kept fixed for these experiments. It is worth noting that at large topologies DCE outstrips Model. Both the experiments are based on ns-3 simulator, so thedifferences can grow through the internal algorithms of protocol implementations. It was rather expected that while the simplified model executes faster at small topologies, the production quality implementation outperforms at large scale where the deep runtime optimizations of the real code pay off DCE overheads.

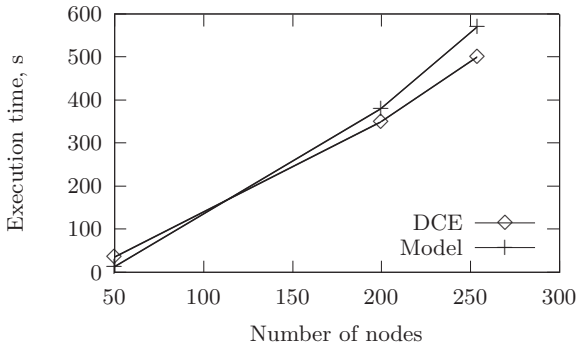

Figure 3: Simulation run time as the function of number of nodes

6. CONCLUSION

In this work we have shown that direct code execution (DCE) framework developed recently for ns-3 (see [14]) is capable to run the unmodified production quality OLSRd daemon. This is probably true for any other MANET routing protocol implemented in user space.

Comparing the behaviour of the native ns-3 OLSR model and the directly executed OLSRd daemon we have found large differences. Obviously enough, built-in models from the simulator kit must not be blindly believed to predict the behaviour of the real world protocol implementation. However, the origin of these differences was identified and after

corresponding *calibration* work DCE and Model show almost identical results running on simple small-scale topologies. This fact can be treated as the validation of the native ns-3 OLSR model against real implementation, but note, that the real OLSRd daemon was also modified in the calibration process to fit OLSR RFC better.

The proposed calibration procedure can be useful for modeling other MANET routing protocols as well. Interesting enough, the calibration procedure can be also be used for *implementing* new routing protocols assuming that corresponding ns-3 model is already built.

Even after the calibration some difference in the protocol implementation still exists as can be seen from the simulating large-scale scenarios.

The last question we cover in this work is what approach to MANET simulation should be used: modeling or direct routing daemon execution?

When the simulation is used to support the protocol development both the approaches can be used. Modeling will be simpler to use at the initial stages of the protocol development, but implementation and direct code execution will likely pay off on the later stages such as testbed and field experimentats or a real-world deployment (assuming that the protocol will reach these stages). The amount of additional efforts for extending DCE[2] framework to support the specific implementation is supposed to decrease significantly in the near future after the completion of the framework implementation.

When the simulation is used to support the deployment of the existing protocol implementation, direct code execution is recommended. As we have shown in this paper, elusive implementation differences can have a visible effect on the network performance.

The case, where using direct execution seems to be the only way out, is modeling the behaviour of prepatched or extended protocol. Most of the default specifications are traditionally modified for further usage for particular reasons. For instance, OLSRd implementation we use in this work supports link quality extensions as well as a flexible plugin system. Modeling these often undocumented "enhancements" is next to impossible.

Below, there is a list and key facts worth considering while making choice of modeling MANET routing protocol.

- DCE system is more suitable for the selection of the best implementation for the chosen protocol type. Simulation models are suitable for the selection of the protocol type itself.
- Ordinary approach of using built-in simulator models lacks accuracy in most cases of customer-level (uncertified) protocols.
- Different interpretation of the protocol specifications while modeling can lead to significant deviations from the original protocol.

Below there is a list of facts worth considering before using direct code execution.

- DCE system is hard to debug. It mostly fits running already debugged implementation rather than developing a new one from the very beginning.
- DCE system is harder to develop at the beginning.
- The amount of maintaining efforts must typically decrease significantly after the completion of the framework implementation.
- DCE can be used for the calibration of the existing models of any nature to better match the implementation under consideration.

[2]Hereinafter term "DCE" refers to Direct Code Execution technique

7. REFERENCES

[1] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626 (Experimental), October 2003.

[2] The ns-3 network simulator. `http://www.nsnam.org/`.

[3] OLSRd: an adhoc wireless mesh routing daemon. `http://www.olsr.org/`.

[4] Josh Broch, David A. Maltz, David B. Johnson, Yih chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. pages 85–97, 1998.

[5] Samir R. Das and Charles E. Perkins. Performance comparison of two on-demand routing protocols for ad hoc networks. pages 3–12, 2000.

[6] Luc Hogie, Pascal Bouvry, Frédéric Guinand, and Hogie Bouvry Guinand. An overview of manets simulation. In *Electronic Notes in Theoretical Computer Science*, 2006.

[7] Robert G. Sargent. Verification and validation of simulation models. In *WSC '07: Proceedings of the 39th conference on Winter simulation*, pages 124–137, Piscataway, NJ, USA, 2007. IEEE Press.

[8] Stuart Kurkowski, Tracy Camp, and Michael Colagrosso. Manet simulation studies: The incredibles. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9:50–61, 2005.

[9] Jason Liu, Yougu Yuan, David M. Nicol, Robert S. Gray, Calvin C. Newport, David Kotz, and Luiz Felipe Perrone. Simulation validation using direct execution of wireless ad-hoc routing protocols. In *In 18th Workshop on Parallel and Distributed Simulation (PADS'04)*, 2004.

[10] DaSSF project: Scalable Simulation Framework. `https://www.primessf.net/bin/view/Public/`.

[11] The Network Simulation Cradle. `http://research.wand.net.nz/software/nsc.php/`.

[12] Conceptual Architecture for Evaluating Inter-Domain Solutions within OneLab2. Technical report.

[13] Jist: an efficient approach to simulation using virtual machines: Research articles. *Softw Pract Exper*, 35(6):539–576, 2005.

[14] Mathieu Lacage. *Experimentation Tools for Networking Research*. PhD thesis, 2010. Universite de Nice Sophia-Antipolis, Supervisor-Dabbous, Walid.

[15] Stuart Kurkowski, Tracy Camp, and William Navi. Minimal Standards for Rigorous MANET Routing Protocol Evaluation. Technical report, Colorado School of Mines, 2006.

[16] The ns-3-dce branch of network simulator. `http://code.nsnam.org/mathieu/ns-3-dce/`.

[17] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for

Real-Time Applications. RFC 3550 (Standard), July 2003. Updated by RFC 5506.