Department of Computer Science
Aalborg University

The Optimized Link State Routing Protocol

Performance Analysis through Scenario-based Simulations

M.Sc. Thesis

by

Lars Christensen

&

Gitte Hansen

Spring 2001

Supervisor: Thomas Heide Clausen

Department of Computer Science

Aalborg University

Title:
The Optimized Link State Routing Protocol – Performance Analysis through Scenario-based Simulations

Project:
Master's Year
Dat 6, Spring 2001

Project group:
E3-101b

Group members:
Lars Christensen, larsch@cs.auc.dk
Gitte Hansen, gitte@cs.auc.dk

Supervisor:
Thomas Heide Clausen

Number of copies: 7

Number of pages: 138

Appendices:
Simulation Overview
Simulation Data

Abstract

In this project we perform an empirical study of the performance of the Optimized Link State Routing Protocol with exhaustive scenario based simulations in Network Simulator 2. We propose the use of enforced jitter and piggybacking on the transmission of control messages. Furthermore we test a simple link hysteresis and adjust the message control intervals. We show that the use of jitter has a substantial effect on the performance of the protocol and that using piggybacking, link hysteresis, and adjusting the control message intervals does not have a significant effect. Finally, we perform a comprehensive comparison of OLSR with AODV that uncover the types of scenarios in which each of the protocol excel. The result of the comparison is that OLSR perform equal to AODV in many scenarios, but substantially better in networks with low mobility, high load, high density and/or sporadic traffic.

To assist us in performing this evaluation we have developed a framework for performing the simulations. This framework includes a scenario generator that generates random scenarios within the constraints of predefined parameters that characterize the scenarios. The complete framework includes the simulator, the scenario generator, and a set of utilities to gather descriptive measures for the simulator output.

Institut for Datalogi
Aalborg Universitet

Titel:
The Optimized Link State Routing Protocol – Performanceanalyse med udgangspunkt i scenariebaserede simulationer

Projekt:
Specialesemester
Dat 6, Foråret 2001

Projektgruppe:
E3-101b

Gruppemedlemmer:
Lars Christensen, larsch@cs.auc.dk
Gitte Hansen, gitte@cs.auc.dk

Vejleder:
Thomas Heide Clausen

Antal eksemplarer: 7

Antal sider: 138

Bilag:
Simulation Overview
Simulation Data

Synopsis
I dette projekt foretager vi et empirisk studie af ydeevnen af routningsprotokollen 'Optimized Link State Routing Protocol' omtømmende scenariobaserede simulationer i Network Simulator 2. Vi foreslår brugen af påtvunget jitter og piggybacking på udsendelse af kontrolbeskeder. Desuden tester vi en simpel link-hysterese og justerer kontrolbeskedintervallerne. Vi viser, at brugen af jitter har en betydelig effekt på protokollens ydeevne og at brugen af piggybacking, link-hysteresen og justering af kontrolbeskedintervallerne ikke giver en tydelig effekt. Vi foretager en analyserende sammenligning med AODV, der viser i hvilke tilfælde hver af protokollerne yder bedst. Resultat af dette er, at OLSR yder lige så godt som AODV i mange scenarier, men betydeligt bedre i netværk med lav mobilitet, megen og sporadisk traffik og/eller høj densitet.
Som hjælp til at udføre denne evaluering, har vi udviklet et framework til at afvikle simulationerne. Dette framework indeholder en scenariegenerator, der kan opstille tilfældige scenarier baseret på en foruddefineret mængde af scenarieparametre der karakteriserer scenarierne. Frameworket består af netværksimulatoren, scenariegeneratoren og et sæt af værktøj til at indsamle beskrivende målinger fra outputtet af selve simulationerne.

Preface

This report documents our work on the master's year at the Department of Computer Science, Aalborg University, Denmark. The thesis documents the results of the work done from September 2000 to July 2001 under the thematic frame of distributed systems.

The formal purpose of the report is to document our ability to work autonomously with a project encompassing empirical and/or theoretical investigation of one or more problem areas relating to central subjects within the area of distributed systems, and to apply theories and methods on a scientific level.

To do this, we have evaluated the performance of the Optimized Link State Routing (OLSR) protocol alone and in comparison with the Ad Hoc On-Demand Distance Vector Routing protocol (AODV). We have implemented OLSR for Network Simulator 2 [nsh] and created a scenario generator. We have developed a framework for simulation and analyzing the results hereof. We introduce the use of enforced jitter and piggybacking as enhancements to OLSR and test a method for using link hystereses. We test and describe the performance of OLSR and AODV in various scenario settings.

References are shown in brackets and refers to the bibliography at page 100. (for example [JMQ+01]). The bibliography contains the sources and references we have used trough out the project. We have included a vocabulary with special expressions used in this report, starting at page 97. This is to prevent misinterpretations in the different contexts.

We would like to thank the research unit Project Hipercom, INRIA Rocquencourt, France for their hospitality during our stay in the fall of 2000 and their cooperation throughout the project, our supervisor Thomas Heide Clausen for extensive and helpful support and critique during the project, and the Mindpass Center for Distributed Systems for allowing us to use their cluster for running simulations.

Lars Christensen

Gitte Hansen

Contents

1. Introduction

For at least the last quarter of a century, research in wireless data communication and networks has been ongoing. In the past, wireless networks were mainly studied in defense research under the name *packet radio networks*, for example [JT87]. The advances in the computing power of mobile computers, and in wireless communication, have increased the applications of and hence the commercial interest in this field. During recent years there has thus been substantial development in the field of wireless data communication. For example GSM is widely spread. Other examples of wireless technologies are: Bluetooth [Blu01], HIPERLAN [ETS95], and IEEE 802.11 [LAN99a]. Bluetooth includes specifications for medium, data/link, and transport layers (plus additional functionally such as service discovery). HIPERLAN, which is an ETSI standard for mobile LANs, includes medium access layer routing. The IEEE 802.11 standard includes specifications of the physical and medium access layers. These new technologies are convenient alternatives to traditional wired networks – users do not need to connect wires to be on the network. An example of this convenience is printing on a network printer. A person can print from his laptop without having to connect physically to the network. Likewise, he will be able to surf the web or to synchronize his PDA wirelessly.

1.1. Network Organization

There are two fundamentally different ways of organizing a wireless network.

Cellular networks

An existing LAN is extended with base stations which allow mobile devices to connect over a wireless medium. The base stations and the attached LAN work as a backbone to the mobile devices. The mobiles devices never communicate directly but always through a base station. Some of the problems in these network are security problems, and transit between different base stations (especially minimizing the 'hand-off' period).

Self organizing networks

A replacement of LANs with self organizing wireless, mobile devices – nodes[1]. There is no wired infrastructure and the hosts communicate directly or by multiple hops

[1] In the following all nodes are routers, and may also have one or more hosts associated.

using each other as routers. The network may be connected to other networks through gateways. Such a network is also called a Mobile Ad-Hoc Network (MANET). The main problem in a MANET is how to maintain connectivity, that is, how to route data through this ad-hoc infrastructure. The network is more dynamic and unreliable than in wired networks, so routing is not as simple as in the latter.

An alternative use of multiple hop wireless communication is as *transit networks* – a group of small inexpensive devices used only for establishing contact between two nodes out of each others radio range. As an example, assume two military units in the field wishing to communicate. Using a multihop transit network with low power transmitters would allow them to conceal the communication, while the use of a single powerful transmitter to establish a single-hop path would make the network more vulnerable, as there is only one point of failure, and one point that the enemy has to detect and supervise.

In this project we will be working only with self organizing networks, in particular MANETS.

1.2. Issues Related to Manets

In this section we will describe the issues and considerations that are related to MANETS. The purpose is to expose the areas that may be problematic in MANETS and which should be taken into consideration, when working with this type of network.

Mobility

Nodes in a wireless network may be mobile. When they move, new links will be created and others will break causing the topology of the network to change. The problem, when mobility exists, is how to maintain connectivity between devices when the topology changes continuously and, potentially, rapidly.

Distributed operation

A MANET should work without any central authority because a node cannot rely on connectivity to such an authority. For a MANET to be functional, even if any subset of nodes are down or out of radio range, all nodes must be equivalent: they must all provide the ability to route data to other nodes, and be able to be self organizing.

Bandwidth

Bandwidth is typically low compared to wired LAN networks. In IEEE 802.11 the maximum bandwidth is 2 Mbit/s [LAN99a], in IEEE 802.11a it is 54 Mbit/second [LAN99b], and in IEEE 802.11b the maximum bandwidth is 11 Mbits/second [LAN99c]. Furthermore, the radio frequencies used in these standards are "public frequencies". This means that they may be used by other devices which may impact the available bandwidth as a result of interference. Interference is especially a problem, because wireless communication channels are not shielded as cables may be. The lower bandwidth of wireless networks is a problem because people using it as a replacement for a LAN will expect the same performance.

Security

The lack of a shielded channel in wireless communication implies that MANETS do not have the inherent physical security as assumed in wired networks. It is easy to eavesdrop on wireless data communication because gaining unauthorized access to the media is simple: radio waves may be intercepted directly whereas it is necessary to gain physical access to wires. For instance, communication on a wireless network in an office environment could easily be eavesdropped on by a person sitting in a car in the parking lot. Therefore, the use of encryption and secure authentication, for example using public key cryptography, is very important.

Routing

A MANET that allows wireless, mobile devices to communicate by multiple hops to nodes beyond their radio range, requires a routing protocol. This should either update the routing tables in each node to reflect the continuous changes in the topology, or have a method of finding a route to a specific node, when it is needed.

Traditional routing protocols which as specifically designed for wired networks, perform poorly in MANETS. Such protocols are designed for highly reliable, high bandwidth networks with a relatively static topology. In contrast to this, MANETS typically have low available bandwidth, are much more unreliable, and may have a highly dynamic topology. Hence, routing protocols designed specifically for MANETS are needed.

Address assignment

For MANETS to be completely autonomous and self organizing, some sort of address assignment scheme needs to exist. This is a problematic requirement, because no central authority can exist. A simple scheme to handle address assignment has been suggested in [RBP00], but there are a lot of possible complications such as healing of network partitions, authenticity etc. that the approach does not handle.

In this project, we are working only with the problems of routing in a MANET.

1.3. Manet Routing

Design of protocols to handle routing in MANETS involves many considerations. The IETF has established a MANET working group [IET] whose focus is to develop and evolve MANET routing specification(s) and introduce them to the Internet Standards track. The MANET working group defines a MANET as:

> *A "mobile ad hoc network" (*MANET*) is an autonomous system of mobile routers (and associated hosts) connected by wireless links – the union of which form an arbitrary graph. The routers are free to move randomly and organize themselves arbitrarily; thus, the network's wireless topology may change rapidly and unpredictably. Such a network may operate in a stand-alone fashion, or may be connected to the larger Internet.* [IET]

There are two general methods of providing routing in a MANET. Either topology information is continuously diffused into the network in order for each node to continuously maintain routes to all other reachable nodes (proactive routing). Alternatively, each node should be able to request a route to any other node when it is needed (reactive routing). The "Optimized Link-State Routing Protocol" (OLSR) [JMQ+01] is an example of a proactive routing protocol for MANETS, while the "Ad-Hoc On-Demand Distance Vector Routing Protocol" (AODV) [PRD01] is an example of a reactive routing protocol. Both protocols have been proposed under the IETF MANET working group. We will be working with OLSR in this project, using AODV for comparison.

There are a number of issues that must be taken into account in the design of a MANET routing protocol. In the following, we list a selection hereof:

Topology dynamics

As described in section 1.2, the topology of a MANET is often far more dynamic that conventional wired networks. The density and size of a MANET also varies.

Bandwidth

As described in section 1.2, bandwidth in wireless network is typically low. Hence it is important for the routing protocol to avoid generating unnecessary overhead in order to maximize the amount of bandwidth available to data traffic. To use the least bandwidth the protocol must also provide the shortest routes (to avoid unnecessary retransmissions of data packets), and provide routing over stable links (to avoid too many packet losses due to a low quality link which causes retransmissions of packets).

Link stability

As described in section 1.2, the links in a wireless network are much less reliable than those of a traditional wired network, because of radio interference from objects and other radio communication on the same frequency band. A link may have a low throughput rate because of transient interferences, or may appear to switch between being available and unavailable because of periodic interferences. Furthermore, under some circumstances, links can be uni-directional. For example, if one of the transmitters is more powerful than the other.

Security

As described in section 1.2, security, and in particular authenticity, is a problem in wireless networks. In connection with routing in MANETS, taking over another nodes' identity and transmitting invalid request and responses into the network is an easy task. For example, a node could transmit incorrect topology information in order to confuse other nodes relying on this information to be true.

1.4. Related Work

A number of routing protocols for MANETS have been proposed under the IETF MANET working group (prime July, the number of proposed unicast routing protocols is 9). Each of the protocols uses different methods and strategies for routing data packets through the network. Only few performance analyses have been performed, be that analytical modeling, simulations, or practical experiments. The number of comparisons of the methods in the different protocols are even rarer and the main works are simulations. This section will describe an analytical modeling of OLSR, simulations and comparisons of MANET routing protocols, and finally a practical experiment with a MANET routing protocol.

1.4.1. Analytical Modeling

"Overhead in Mobile Ad-hoc Network Protocols" [JV00] is a theoretical comparison of the overhead in mobile ad hoc network in terms of control traffic and overhead due to route suboptimality. The article's conclusion is in favor of OLSR when the number of active routes is high and when there is relatively low mobility.

1.4.2. Simulations

Most simulations that do exist are scenario based and performed using Network Simulator 2 (NS2) [nsh]. This includes [BMJ+98], [JLH+99] and [Sam00], which are the three main works in simulations of MANETS. Furthermore this section describes a simulation of OLSR in a custom made simulator.

Broch, Maltz, Johnson, Hu, and Jetcheva

"A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols" [BMJ+98] compares AODV, DSDV, DSR, and TORA using NS2 with up to 50 nodes in a MANET and speeds up to 20 m/s. The following scenario parameters were varied: the movement pattern (7 different node rest times) and the communication pattern (3 different numbers of Constant Bit Rate (CBR) sources). The reason for not using TCP sources is that TCP offers a conforming load to the network and the authors therefore found it to be unsuited for comparison. 10 scenarios of each movement pattern were generated, and 210 simulations for each protocol were performed, in all 840 simulations. With no mobility, DSDV delivers almost all packets, but fail to converge when the mobility is high. TORA is the worst performer. DSR and AODV perform best, but have different expenses, in terms of overhead, with different scenario parameters.

Johansson, Larsson, Hedman, Mielczarek, and Degermark

"Scenario-based Performance Analysis of Routing Protocols for Mobile Ad-hoc Networks" [JLH+99] uses scenario-based performance tests for the comparison of AODV, DSDV, and DSR with the network simulator NS2. Results are presented as a function of a mobility metric designed to reflect the relative speed of the nodes and are based on up to a maximum of 50 nodes in a MANET. The following scenario parameters were varied: the mobility metric (8 different values corresponding to from 0 to 20 m/s) and the traffic load (4 different packet rates, all with CBR sources). Furthermore, 3 specific scenarios were simulated: a conference scenario, an event coverage scenario, and a disaster area scenario. These are intended to model realistic scenarios. The tests were performed with varied mobility, and with varied mobility and load. One scenario with each scenario parameter set was simulated, and 43 simulations for each protocol was performed, in all 129 simulations. The main result is that the reactive protocols, AODV and DSR, perform better than the proactive one, DSDV, at different loads of traffic, and that AODV performs best.

Das, Perkins, and Royer

"Performance Comparison of two On-demand Routing Protocols for Ad Hoc Networks" [Sam00] uses scenario-based performance tests for the comparison of AODV and DSR with the network simulator NS2 with 50 or 100 nodes in a MANET. The following scenario parameters were varied: the movement pattern (7 different node rest times), the communication pattern (4 different numbers of CBR sources), and the traffic load (7 different loads). 5 scenario of each scenario parameter set were generated, and 245 simulations for each protocol was performed, in all 490 simulations. The main result is that in the "less stressed" situations, that is, small mobility, small load, small number of nodes, DSR performs best, while AODV performs best in "highly stressed" situations. DSR, however, generates the smallest overhead in all situations.

Qayyum

Part of "Analysis and Evaluation of Channel Access Schemes and Routing Protocols in Wireless LANs" [Qay00] concerns the performance evaluation of OLSR through simulations. The simulator used is custom made with models of the physical layer, signal propagation, traffic, and queuing. The simulator is simplified and does not take into consideration such factors as reflections, interface queues, MAC overhead, etc. The evaluation has character of theoretical and analytical modeling due the perfectionism of the behavior in the simulator. Basic protocol behavior, protocol performance in a static network, with and without varying load conditions, and performance in a mobile network was evaluated. One scenario with each varied parameter was simulated. The results and modeling showed that the theory behind multipoint relays (MPRs) is very effective (MPRs are explained in section 3.1.1), that OLSR is best suitable in dense networks with frequent route request for new destinations, and that OLSR creates optimal routes. A minor comparison with a simplified DSR was made arguing in favor of OLSR. The simulated networks were static and no expiration of routes was used in any of the protocols. Simulations were run in two steps: first, DSR made route discovery between all nodes. Second, simulations were run with data traffic, with DSR and OLSR, respectively. The main conclusions were that OLSR creates better routes and hence delivers packets with lower latency, and that OLSR is better in dense networks.

The general conclusion of these articles comparing protocols is that of the tested protocols, AODV is the one that performs best in the widest range of scenarios. Not all protocols have been tested however. Especially the OLSR protocol has not yet been compared to others in simulations other than [Qay00].

10 scenarios were generated for each set of scenario parameters in [BMJ+98], 1 scenario for each set in [JLH+99] and [Qay00], and 5 scenarios for each set in [Sam00]. They all examine only CBR traffic. Some of the scenarios used in these simulations have parameters that are distributed randomly, while 3 of the scenarios in [JLH+99] were modeled to be realistic.

Our Evaluation

We find the conclusions in [JLH+99] problematic, since a protocol might show better results based on chance (or a lucky pick of scenario), when only simulating one scenario with each set of scenario parameters. This is seen by the fact that the graphs in [JLH+99] are ambiguous or show no tendencies. [BMJ+98] and [Sam00] perform 10 and 5 scenario of each set of scenario parameters, respectively, but only vary 3 parameters. Though better than only one test of each situation, we find, however, that 5 and 10 are still too few to average out lucky cases. According to [Mit97], at least 30 of each situation should be performed in order to get a representative set of samples. We also find that variation of three parameters is too few to make exhaustive simulations.

It is important to take the nature of the traffic into consideration, when evaluating the results, but it is not essential that the scenarios created from each set of scenario parameters are identical, as long as the lucky cases are averaged out by the number of tests. Furthermore, the simulations test only CBR traffic. We find this problematic as well, since TCP traffic is most likely used where it would be relevant to have a MANET, for example file transfers, downloading of files, surfing[2] etc. The argument for not using it in [BMJ+98], that TCP traffic is conforming, is to general.

We do not find the simulations in [Qay00] comprehensive enough to reveal all the required properties and find that the simulator is too simplified. However the results from the simulations and the analytical modeling indicates areas of importance to examine when evaluating the performance of OLSR. Furthermore, we find the comparison between OLSR and DSR problematic as DSR does not have the ability to act reactively in the simulations,

[2]Measurements on the MCI backbone show that about 25% of the bytes carried across the network are carried by TCP. Of these 50-70% are HTTP messages [TMW97]

because it has non-expiring and non-changing routes ready beforehand. Thereby the true nature of DSR is not revealed making it difficult to conclude upon the results.

It may be difficult to compare "best performance" from different simulations, as this may be measured in numerous ways. Best may be "minimum overhead", "minimum latency", or "maximum throughput" depending on the measurements used. And likewise the conclusions may be very different. It is therefore important to take the different measurements into consideration when evaluating the results. The measurements we use are described in sections 2.3 and 5.3.

1.4.3. Practical Experiences

"Quantitative Lessons From a Full-Scale Multi-Hop Wireless Ad Hoc Network Testbed" [MBJ99] test the performance of DSR in a full scale testbed. The testbed consists of 5 moving nodes and 2 stationary nodes. Each node was equipped with WaveLAN-I radios and GPS receivers to determine each node's location at a given point. The main results from the test is that jitter has to be introduced in the network and there is a need for hysteresis to prevent using transient routes.

1.5. Previous Work

This section will describe the results of the work on our previous semester [CEH01] that have influenced this project.

Scenario Generator

During our previous semester, we designed and partially implemented a scenario generator to enable us to generate random scenarios with certain characteristics. This was to ensure that we were able to generate numerous scenarios with the same set of scenario parameters. We need to generate a large quantity of scenarios with identical scenario parameters to ensure the validity and generality of the results. The scenario generator was finished during this project, and is described in chapter 4.

Practical Experiments

When performing practical experiments, we discovered some idiosyncrasies of MANETS and MANET routing protocols. The implementation of OLSR used for these experiments was developed by [BHJ+00] and reworked by Peter Jensen and ourselves.

Our experiments showed that under high load, a lot of control messages are lost due to collisions. This results in poorer performance, because there is not enough topology information diffused into the network. Hence, not all the nodes have information of all other nodes and data packets are dropped due to route unavailability. Our experiments indicated that the collisions were due to synchronized transmissions of control messages by neighboring nodes. That is, using fixed control message intervals may impact the performance of the protocol because nodes synchronize and, therefore, loose in the order of 10 consecutive control messages due to collisions. By introducing jitter on the transmission of control messages, the number of messages lost due to collision were significantly reduced. Therefore our experiments indicated that performance may be substantially improved by enforcing jitter on the transmission of control messages. This phenomenon was also experienced in [MBJ99].

A possible explanation is, that the probability of collisions is large if two neighbors begin transmitting control messages simultaneously. If the interval between transmitting control messages is the same at all nodes and at all times, the messages will keep on colliding until one of the nodes either moves out of range or gets out of sync.

Furthermore, our experiments showed that the links in a MANET are unstable when the nodes are relatively far from each other. The experiments indicated that OLSR handles unstable links badly, which resulted in route flapping, and that the protocols performance may be improved by detecting bad links and using this information in routing and/or link state determination. A solution to this could be to use a conservative link hysteresis, for example by only using links where 2 out of 3 control messages arrive. Another scheme to solve this is to evaluating the stability of the links thereby avoiding the use of less stable links. This has been suggested in [BCCH01], where experiments have shown that performance can be improved by only using less stable links for routing when these are the only links available.

Preliminary Simulations

To perform preliminary simulations, we implemented OLSR for NS2. We tested OLSR against AODV, but the results indicated that a quantification of the results is necessary to ensure validity and generality in the results.

Our simulations furthermore indicated that piggybacking control messages can improve the diffusion of control messages into the network because more messages get through with piggybacking than if they were transmitted individually. This has also been confirmed by experimental results in [BCCH01].

1.6. Goals

Besides the experimental and simulation results, there are aspects of OLSR which have yet to be investigated. This includes the frequencies of control messages.

The goal of our project will be to perform a comparison of the Optimized Link State Routing protocol and the Ad Hoc On-Demand Distance Vector Routing protocol (AODV), in order to find out whether OLSR is actually better in dense networks with sporadic traffic as claimed in the protocol specification [JMQ+01]. Furthermore, we wish to examine the problems of control message loss and route flapping further, especially in order to evaluate the proposed solutions' impacts on the protocol's performance. We want to perform exhaustive simulations to ensure the validity and generality of the results.

The goals are to:

- compare the performance of OLSR with the performance of AODV in a wide range of scenarios.
- examine the effect on the performance of OLSR of introducing jitter on the transmission of control messages.
- examine the effect on the performance of OLSR of introducing piggybacking.
- examine the effect on the performance of OLSR by changing the frequencies of control messages.
- examining the effect of using conservative link detection to handle route flapping and improve the performance of OLSR.

The next chapter will state the work process, theses, methods, and structure of this report.

2. Methods and Structure

In this chapter, we will describe the methods of this project and the structure of the report. First, we describe the work process that this M.Sc. thesis is based on, and how we have arrived at using the applied methods to confirm or reject the theses. Next, we briefly restate the problems described fully in section 1.5 and argue their relevance. Furthermore we describe the applied methods and measurements. Finally, we give an overview of the rest of the report.

2.1. Work Process

Our main goal for this and the previous semester has been to evaluate the performance of OLSR. We want to evaluate large test beds and perform a large number of tests. To do this we first studied the functionality of OLSR. We updated an existing implementation for Linux from [BHJ$^+$00] with the help of Peter Jensen. This implementation was used to make preliminary investigations. Furthermore, we have studied the functionality of AODV, as this was the protocol we wanted to use for comparison.

Generally, there are three main performance evaluation methods; analytical modeling, simulation, and practical experiments. We have chosen to use simulations for evaluation rather than practical experiments and analytical modeling. Analytical work such as [JV00] is at the risk of neglecting important features and properties of a real world network, because simplifications and assumptions are required to enable the modeling. It is not always practically possible to evaluate large scale situations with practical experiments alone, because they have high resource requirements in form of equipment and manpower etc. Hence, practical experiments are not applicable in our situation as we want to perform numerous, repeatable tests to ensure the validity and generality of the results. With simulations, it is possible to repeat tests which are performed in controllable environments. This makes it easier to evaluate specific situations. However, according to [Jai91], when choosing an evaluation method, it is important to take into considerations the contributions that the two other methods may add to the evaluation. We use practical experiments to reveal areas of relevance for further investigations and furthermore use the results from the analytical modeling in [JV00] for finding scenarios of interest.

We have used Network Simulator 2 (NS2) [nsh] for simulating the wireless networks in this project as this is the simulator used in the majority of other performance evaluations of MANETS as described in section 1.4.

During our investigations of related work, we found that in much work, results were based on single or few instances of random scenarios while other work was based on specific scenarios, not necessarily impartial to the protocols. We want to create a large number of scenarios with specific characteristics, but still impartial to any protocol. Furthermore we want to be able to test the protocol under different conditions and different behaviors of a MANET. To fulfill this, we have created a scenario generator that takes a set of scenario parameters and create random scenarios within the constraints of the parameters. Furthermore, the scenario generator automates the process of creating scenario files for NS2, which has aided us in running a large number of simulations.

To ensure that our results are valid and general, we want to eliminate the possibility of results appearing by chance. We have achieved this by running numerous simulations and analyzing the results with the aid of statistical methods to assure representativity.

2.2. Theses

This section describes the different theses we advance. The first 4 exclusively concern the performance of OLSR and enhancements hereof. The last thesis concerns the performance of OLSR in comparison with AODV.

Jitter

In our practical experiments and in simulations, we discovered that a lot of control packets were lost due to collisions, when a fixed control message interval was used.

We anticipate that introducing jitter on the transmission of control packages will improve the performance of OLSR. If the number of dropped control messages is lowered, more data packets will arrive at their destination because of higher route availability.

We will simulate scenarios with and without enforced jitter on the control message intervals in order to determine the effect of enforcing jitter.

Piggybacking

In some simulations we experienced that piggybacking control messages increased the performance of OLSR. We wish to verify whether piggybacking, in general, improves the performance of the protocol.

We will simulate scenarios with a variable holdback time. The holdback time is the time a message is held back in an attempt to piggyback it with other messages.

Control Message Intervals

Values for control message intervals used in OLSR are suggested in the draft. Although these values may be reasonable it has not been determined whether these values are optimal. We want to determine whether better performance can be obtained by adjusting these control message intervals.

We will simulate scenarios with variable control message intervals.

Handling Unstable Links

Practical experiments have shown that unstable links in a MANET affect the performance of the protocol negatively. We want to investigate whether the simple method of conservative link detection described in section 1.5 can improve the performance of the protocol.

We will simulate scenarios with and without conservative link detection.

Performance Comparison with AODV

It has only been evaluated through analytical modeling and simplified simulations how well the OLSR protocol performs in comparison with other MANET routing protocols. We want to gain a general picture of when OLSR performs well – and when it does not. For comparison, we will use AODV (the protocol that has performed best in other simulations). We will use simulations to test OLSR in a wide range of scenarios with variable mobility, node density, and traffic characteristics.

2.3. Method

Our main method for verifying the theses and showing the effect of the various changes to the protocol, is to simulate wireless networks with different scenario parameters. For each thesis we generate various different scenarios with the same parameters for each possibility that is to be tested. We simulate the scenarios in a network simulator and analyze the results from the simulation using statistical tools.

Measurements

We use the following measurements for evaluating the protocols:

- Throughput: The number of data packets that reach their destination. That is the number of received packets.

- Overhead: The amount of bandwidth occupied by control traffic. This may be measured in number of packets or bytes.

- Packet delay: The time between a packet is transmitted by an application and until it is received. That is, the time from source to destination.

An elaboration of the concrete measurements can be found in section 5.3.

2.4. Overview of the Report

Chapter 3 describes the OLSR protocol in detail with emphasis on the functionality of the protocol. It furthermore contains a description of AODV, the protocol used for comparison in this project, and a discussion of the protocols. Chapter 4 describes the scenario generator that we have built to automate the generation of scenarios. The scenario generator allows us to generate a wide range of random scenarios with the same set of parameters and hence avoid simulating only scenarios that give good, or bad results by chance.

The simulator and method of simulation is described in detail in chapter 5. We simulate the wireless networks using Network Simulator 2 (NS2) [nsh]. This simulator is able to simulate all network layers from the physical layer to the transport layer, and should therefore provide a reasonable and realistic picture of the performance of the network. Furthermore, the chapter contains a section about technical issues concerning NS2.

Chapter 6 describes statistical utilities. To analyze the results of the simulations, we extract data such as throughput, delay, and control overhead and examine these using statistical tools. We use both measures of central tendencies and dispersion. In some cases we also use the chi-square test of independence to calculate the probability that results may appear by chance. To lower this probability, we run at least 30 different scenarios with the same set of scenario parameters for each test.

In chapters 7 and 8 we present the results of the simulations we have run to observe the performance of OLSR and the comparison of OLSR and AODV, respectively. The chapters contain the test configuration of the scenarios, and for each test set the following will be described: the thesis that is to be tested, the parameters which are varied, and the results. Each test set is concluded by an analysis of the results.

Chapter 9 concludes and summarizes the report. We have included appendices to give an overview of the simulations we have run, and the data extracted from the results.

3. Study of Two Manet Routing Protocols

In this chapter we will describe two MANET routing protocols. We have studied the protocols to understand their functionality, to be able to perform exhaustive comparisons between them. And furthermore, to be able to fully implement the Optimized Link State Routing (OLSR) protocol in both a simulator (NS2) and for the Linux operating system. The Optimized Link State Routing protocol [JMQ+01] is a proactive link-state routing protocol and the Ad Hoc On-Demand Distance Vector (AODV) routing protocol [PRD01] is a reactive routing protocol. Currently, OLSR and AODV are Internet drafts in the MANET working group [IET] and thus proposals for a MANET routing protocol standard. They are as such to be considered as work in progress. OLSR is currently in the 4th version and AODV in the 8th version. First we describe OLSR with emphasis on the functionality. Next, we will give an overview of AODV. The chapter is concluded by a comparison of the two protocols and the anticipations we have for their performance when conducting tests and simulations.

3.1. Optimized Link State Routing Protocol

The Optimized Link State Routing protocol [JMQ+01] (OLSR) is an optimization over the pure link state protocol. OLSR is a proactive routing protocol which employs periodic message exchange to update topology information in each node in the network. The protocol uses control messages for neighbor sensing to discover the neighborhood and to establish knowledge of the link status between the node and all of its neighbors. This knowledge is then, through the use of Multi Point Relays (MPRs), flooded into the network, providing each node with partial topology information, necessary to compute optimal routes to all nodes in the network. Only nodes selected as MPRs flood topology information into the network. The use of MPRs combined with local duplicate elimination is used to minimize the number of retransmissions in the network and thereby reduce overhead. Likewise optimal routes reduces overhead in the network as described in section 1.3.

3.1.1. Multi Point Relay

OLSR optimizes the process of flooding control messages by using Multi Point Relays (MPRs). Each node selects a set of MPRs among its neighbors. The role of the MPRs is to retransmit the selecting node's control messages. The MPR set is selected so that all two-hop neighbors can be reached through nodes in the MPR set. Only neighbors with symmetric links[1] are considered when choosing MPRs. Computation of the MPR set is triggered by changes in the neighborhood or two-hop neighborhood.

The collection of nodes that have selected a particular node as MPR is the node's MPR selector set.

A minimal MPR set exists, however the computation hereof is an NP-hard problem as there is no known polynomial time solution. Therefore, an heuristic selection algorithm is used. First, all the neighbors which provide the only path to one or more two-hop neighbors are selected. Next, one of the neighbors that can reach most of the two-hop neighbors, not yet covered by the MPR set, is selected and added to the MPR set. This step is repeated until all two-hop neighbors can be reached. The last step in the algorithm is an optimization of the MPR set: Each node in the MPR set is examined. If the MPR set without the particular node still covers the two-hop neighborhood, the node is removed from the set.

If the minimal MPR set is found, fewer packets are retransmitted in the network. It is, however, more important to cover the whole two-hop neighborhood than to have a small MPR set. This is because it is necessary to construct a partial topology graph with a subset of all links, yet with all nodes, to gain enough topology information to make routes from all nodes to all nodes.

Only MPRs retransmit a control message and only if the message comes from a node in its MPR selector set. Other nodes will process the packet, not retransmit it.

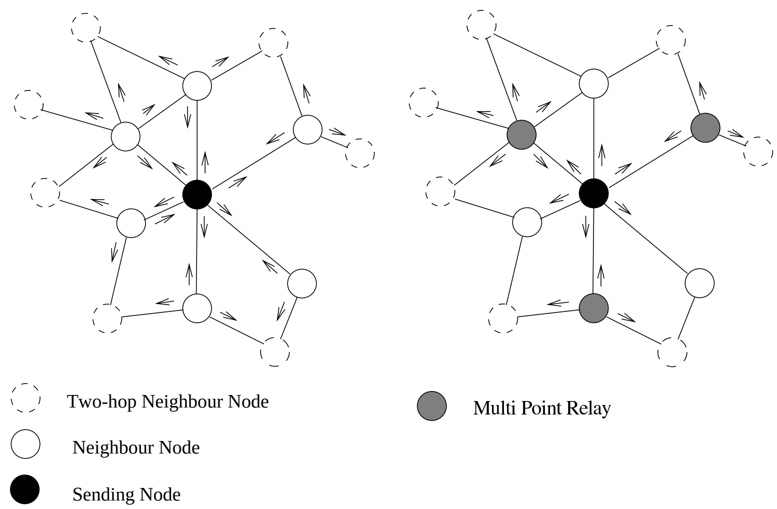
Using MPRs therefore results in a significant reduction in the number of retransmissions in the network. Figure 3.1a illustrates transmission of a packet in a small MANET using pure flooding, whereas figure 3.1b illustrates the same situation, but with the use of MPRs. Each arrow represents a transmission.

The load of control traffic is minimized in part because only nodes selected as MPRs transmits topology information, and in part because only MPRs retransmit control messages for other nodes. Furthermore, the topology information only consists of links to the nodes that have selected the particular node as MPR. This means that the control packet is smaller than if information about all links' states were diffused into the network.

3.1.2. OLSR Messages

There is only one type of OLSR packet. All OLSR messages are sent as payloads in this packet. The packet may contain one or more messages providing the possibility of piggybacking control messages.

[1] Symmetric links are links between nodes, where it is confirmed that both nodes can receive packets from each other.



(a) Pure Flooding [HJR00] (b) MPR Flooding [HJR00]

Figure 3.1.: *A small network with full flooding and MPR flooding.*

In the current version of OLSR there are two types of messages: hello messages and topology control messages.

Hello Messages

Hello messages are broadcasted to the neighborhood at regular intervals. They contain information about the node's known neighbors and the link status between the originator of the hello message and its neighbors. That is, the hello message contains the information from the node's neighbor-table. Each entry in the table is assigned a timeout value, the neighbor hold time. OLSR operates with three kinds of neighbors: Asymmetric[2], symmetric and MPR. Neighbors with the link type MPR are the nodes, to which there exists symmetric links, and that the transmitting node has selected as MPRs.

Upon receiving a hello message, the receiving node updates its neighbor-table. If the transmitting node is asymmetric in the receiving node's neighbor-table and this node find itself in the hello message, it upgrades the status of the link assigned to the neighbor to symmetric. If the receiving node has MPR status in the message it upgrades its MPR selector set accordingly.

[2] A link between a pair of nodes is asymmetric if it is confirmed that data can be received in one direction, but not in both.

Topology Control Messages

Nodes with a non-empty MPR selector set flood topology control (TC) messages into the network within a minimum and maximum interval as defined by the draft [JMQ+01]. The purpose is to inform the other nodes of the status and changes in the topology so they have enough information to construct routes to all other nodes. A TC message contains the address of the originating node and a list of its MPR selector set.

Upon reception of a TC message, the node saves topology information in a topology table, where each entry is assigned a timeout value, the topology hold time. Furthermore, if it is the MPR of the node from which the message was received, the TC message is retransmitted.

3.1.3. Routing

Based on the information in the topology table each node calculates the routes to all other nodes using a shortest path algorithm, for example Dijkstra's algorithm [Dij59], using hop-to-hop routing.

OLSR maintains the routing tables, but leaves it up to the underlying operating system to take care of packet forwarding. Thereby OLSR is not a part of the protocol stack, but only calculates routes and changes the routing tables in the operating system.

3.2. Ad Hoc On-Demand Distance Vector Routing

This section describes the Ad Hoc On-Demand Distance Vector Routing protocol (AODV). Currently, the AODV routing protocol is an Internet Draft in the 8th version in the MANET charter and is to be considered as work in progress.

The presented description is of draft version 6 [PRD00a] as it is this version which is used in the implementation of NS2 used in this project. First, the main functionality of AODV version 6 will be described and followed by a description of the differences between AODV version 6 and the current version 8 [PRD01].

3.2.1. Functionality

The AODV routing protocol is a reactive routing protocol. A node, utilizing AODV, acquires routes only when they are needed for data transmission, and caches them for a predefined period before they time out and are removed. Because AODV is reactive, a node does not maintain routes to all destinations as for example OLSR.

When a route is needed for transmitting a packet, the source node floods a *Route Request* with information about the destination and a hop count which is initialized to 0. Upon reception of a *Route Request*, a node examines whether it has a fresh route to the destination in its route cache[3] If not, it forwards the *Route Request* after incrementing the

[3] A fresh route is a route that has not timed out yet.

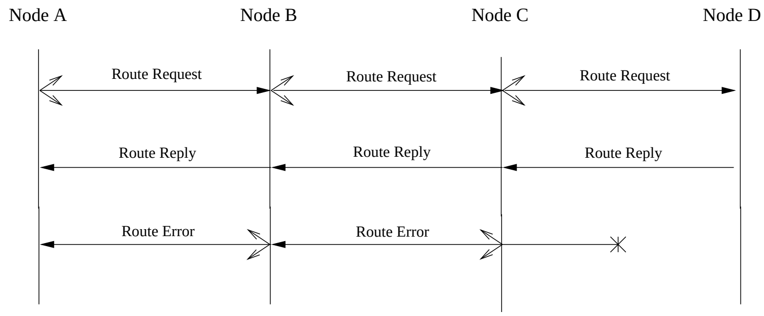


Figure 3.2.: *AODV Messages.*

hop count (See figure 3.2). Otherwise, if a node has a fresh route (or is the destination), it unicasts a *Route Reply* to the source node with information about the new route.

To optimize the search, AODV uses an expanding ring search. A *Route Request* message is first flooded with a time to live (TTL) of TTL_START. If no *Route Reply* arrives within a predefined amount of time, the *Route Request* is flooded again with a TTL that is incremented with TTL_INCRE. The last step is repeated until TTL reached the constant NET_DIAMETER, which is also predefined. This means that a *Route Request* may be flooded several times. TTL_START, TTL_INCRE, and NET_DIAMETER are all defined in the draft [PRD01].

In AODV, when a node receives a *Route Reply*, it saves the information in its routing table before forwarding it in order to optimize future route requests.

The nodes may use neighbor sensing by transmitting periodic hello messages (*Route Reply* with a time to live set to one hop) and that way detect broken links. It is also possible to use link layer notification. If a node detects a broken link, it transmits a *Route Error* message to the neighbor that has recently used the broken link illustrated in figure 3.2. When the transmitting node receives a *Route Error*, it either stops transmitting or transmits a new *Route Request* to "repair" the broken route.

3.2.2. AODV Updates

This section will describe the differences between AODV draft version 6 [PRD00a] and AODV draft version 8 [PRD01].

The difference between AODV draft version 6 and AODV draft version 7 [PRD00b] is the introduction of multiple interfaces. In version 7, handling of multiple interfaces is added, for example if a node has both a wired and a wireless interface. However, in our simulations all nodes have similar wireless interfaces and only one per node.

The difference between AODV draft version 7 and AODV draft version 8 is the introduction of support for unidirectional links. However, in our simulations all links will be bidirectional.

None of the updates affect the basic functionality of the protocol. Therefore it will not have any influence on the performance of the protocol, and the results of our comparison will valid even though the results are based on an earlier version of AODV.

3.3. Protocol Discussion

In this section we will discuss the differences between OLSR and AODV, and how we anticipate that these differences will affect the protocols' performance. In the following an AODV node is a node that utilizes AODV and, likewise, an OLSR node is a node utilizing OLSR.

The basic difference between OLSR and AODV, that OLSR is proactive and AODV is reactive, indicates that OLSR will perform better when traffic is sporadic and that AODV will perform better when traffic is static. That is, when the traffic has long duration.

When talking about a protocol being better, we mean that the general evaluation of throughput, packet delay, and control overhead in networks utilizing the particular protocol is in favor of that protocol. The notions are described in section 2.3.

It is speculated in [JMQ+01] and [JV00] that OLSR will perform best under sporadic traffic where the protocol can benefit from having found the routes proactively. Furthermore, it is anticipated that OLSR will perform better than reactive protocols such as AODV when the network is rather dense because OLSR generates less control traffic due to the use of MPRs.

Route Optimality

AODV bases its routes on the path the initial *Route Request* packet takes to reach the destination node. This path may not be the shortest route, but it will be near optimal, as it is the route that takes the shortest time. Due to randomness in the retransmission of the flooding messages, this may not correspond to the route with fewest hops. OLSR, on the other hand, will provide shortest routes given that the nodes providing the route have sufficient topology information.

Using the routes with fewer hops may not always be an optimal strategy because the route with the fewest hops may also be the route with longer distances between nodes and hence risk being a route with more unstable links.

AODV will not adapt to newly created links that may provide a shorter route through the network. It will only react on broken links. This means that if the network "bends" such that a short route is created, AODV will continue to use the old route. OLSR will adapt and use newly created links as soon as the new topology information is diffused. AODV will detect broken links either using link layer notification or using hello messages and send a notification to the source node, or repair the link locally. OLSR will also detect the broken link (when enough hello message are not received or alternatively link layer information can be used if accessible), and flood new topology information.

Route suboptimality may cause more overhead, because of the number of retransmissions of data packets is higher than if routes are optimal, thereby ensuring the smallest number of hops.

Control traffic

AODV nodes request routes when they are needed while OLSR nodes gather topology information proactively. This means that the amount of control message traffic that an OLSR node generates is constant (with a constant number of nodes), while the control traffic that an AODV node generates depends on the traffic in the network. When there is no traffic in the network, AODV nodes do not generate any control traffic, except if it uses neighbor sensing and transmit hello messages, while an OLSR node generates the same amount as when there is traffic. When there is a high number of active/new routes in the network, the AODV nodes will transmit a lot of *Route Request* and *Route Reply* messages, until it reaches the level where enough routes are cached. Meanwhile, OLSR nodes will keep the amount of control traffic constant.

This indicates that when traffic is highly sporadic with bursts of activity, the AODV protocol's performance will suffer because the network will be highly loaded with control traffic.

AODV uses full flooding when diffusing *Route Request*s into the network. This generates much more control traffic than using MPR flooding such as OLSR, as explained in section 3.1.1. In a fixed size network, the “cost”, in terms of control traffic transmitted, for performing a full flooding increases linearly with the number of nodes, as all nodes retransmit the packet. With MPR flooding, the number of retransmissions with 100 nodes are only 1/5 of the retransmissions with full flooding, and the number hardly increases at all with the number of nodes when above 70 nodes [Qay00]. Furthermore it may take longer time for the control messages to cross the network with full flooding than with MPR flooding. If a control packet is transmitted with full flooding to two nodes, which are each other's neighbors, they are not able to retransmit simultaneously because they use the same medium. If the same situation occurred with MPR flooding, only one of the nodes would have to retransmit it, unless they were both in the MPR set of the transmitting node. The difference is that only when both nodes are MPRs to the transmitting node will there be a problem of simultaneous attempts of retransmitting.

Furthermore, AODV floods *Route Request* packets using *expanding ring flooding*, where the packets are flooding with an increasing Time-To-Live starting at 1 and increasing with 2 each time the request times out. The constants are defined in the AODV draft [PRD01]. Hence, if two node at each side of the network tries to communicate, large parts of the network will be flooded multiple times.

Latency

The latencies in the network are of high importance to the performance. The time it takes for a packet to reach its destination from when it arrives in the IP stack of the source node will have high effect on the end users experience of the network. With OLSR, the latency will be near optimal because of the shortest-path-routing (given that the routes can be found). With AODV, nodes will often have to request routes before packets can be transmitted. This can take multiple seconds because of the expanding ring flooding strategy.

Anticipations

To test the performance of OLSR and AODV, we will vary the test scenario parameters concerning mobility, density, and traffic.

We expect that both AODV and OLSR will have a better performance with low mobility than with high. However, scenarios with OLSR will have a constant amount of control traffic, while scenarios with AODV will have an increasing amount of control traffic, because of the need to transmit *Route Error* messages every time an active route breaks.

We expect OLSR to perform better than AODV in dense networks, because the network will be overloaded with AODV control traffic, whereas the use of MPRs in OLSR should keep the control message overhead at an acceptable moderate level.

The performance of both protocols depend on the nature of the traffic. With lower duration and a constant number of simultaneous streams, we anticipate that OLSR is better because OLSR nodes have routes available when they are needed, while AODV nodes will need to request them. With long duration we expect AODV to perform better. We anticipate that the time used to make a bulk transfer of data from one node to another will be higher for AODV nodes than OLSR nodes. This because an AODV node will need to transmit a *Route request* and wait for the *Route Reply* before the data can be transmitted, while OLSR will have the routes available beforehand.

4. Scenario Modeling and Generation

In this chapter, we will describe the scenario generator that we designed and implemented. We have created the scenario generator to be able to generate a large number of scenarios with the same set of scenario parameters. First, we motivate the creation of the scenario generator. Next, we describe the requirements for such a scenario generator. Finally, we describe our scenario generator with examples of use. The chapter is concluded by a summary.

4.1. Motivation for Creating a Scenario Generator

As described in section 1.4, we found that simulations of MANETS in related work with scenario based simulations have been very few, either random or specific, scenarios. We find it problematic that only few, and in the case with specific scenarios only one, simulations of each was run. This makes it possible to pick a scenario (intentionally or by chance) which gives one protocol advantages over others. Furthermore, only few parameters are varied, and none of the related work test TCP traffic.

We want to create a series of random scenarios that have certain characteristics, but still are impartial to any particular protocol. We want to test the protocol under different conditions and different behaviors.

Furthermore, we want to automate the creation of scenarios because we want to create a large number of scenarios with the same set of scenario parameters in order to get more valid, general, and representative results.

4.2. Requirements of Modeling

To create a scenario generator to fulfill our motivation, we set up the following requirements:

- First of all it is important to be able to specify different parameters for the nodes and the area in the wireless network in order to model the conditions and behaviors. That is, simulation area, number of nodes, movement, and traffic. It is important to be able to specify different kinds of traffic, both streaming and bulk traffic.

- Furthermore, to create scenarios to model realistic situations, it is important to have groups of nodes with their own set of parameters as it is possible that not all nodes have the same behavior. This could be the case at a conference where the speaker has one behavior (stands at the same place) while the spectators may move around, for example, when they enter the room.

- Finally, it is important that the scenarios are impartial to any specific protocols and that it is possible to generate a number of different scenarios with the same characteristics.

To fulfill these requirements, we build a scenario generator that takes a set of parameters and generates a scenario from these. The parameter types are described in the following section, including a semi-formal description. The scenario generator generates random scenarios from the set of parameters by, for example, placing the nodes randomly within the simulation area.

By generating random scenarios from a set of parameters, we are able to generate series of random and different scenarios which still have the same characteristics. This way we ensure that the resulting data, collected from the simulations, are not based on a coincidence. Instead, the results can be averaged over all of the simulations in order to get a representative result.

4.3. The Scenario Generator

The wireless scenario generator was introduced in [CEH01] and has been extended and completed during this project. The scenario generator was used to generate all of the scenarios used in the simulations, presented in this report.

The scenario generator takes a set of scenario parameters as input. The parameters include the number of nodes, the size of the simulation field (a flat ground rectangle of x by z meters), the duration of the simulation, the movement of the nodes, and characteristics of the traffic. The scenario generator then produces a scenario description that includes the nodes, their position and movement, and the traffic in the network. The elements in the scenario description are created randomly based on the scenario parameters. As an example, the positions of the nodes are random, but within the limits of the scenario parameters of the number of nodes, the field size, and the movement. The scenario description is finally converted into a Tcl script, which can be given directly to NS2.

Movement

The movement model used by our scenario generator is a *random movement* model. Each node selects a direction and a distance, moves, and rest at the waypoint where it has arrived. When a node's direction will cause it to move out of the simulation field, it is reflected off the border, like a ball hitting the side of a pool table.

Traffic

Our scenario generator can generate two types of traffic; streaming and bulk data transfer. The streaming traffic is simulated as a constant bit rate transfer of equally

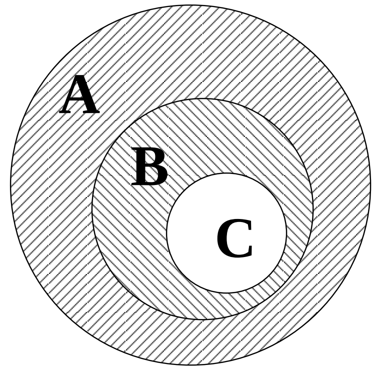


Figure 4.1.: *Groups of nodes*

sized UDP packets being transmitted with constant interval from one node to another. Bulk transfers are simulated by sending a fixed amount of data over a TCP connection.

Groups

The scenario generator can create scenarios with groups of nodes, called clusters, characterized by their own set of parameters describing their field size, number, movement, and traffic. Such groups of nodes can be created recursively. For example, it is possible to have a group A of 20 nodes with a subgroup B with 10 nodes with a subgroup C with 5 nodes. This is illustrated with sets in figure 4.1.

A semiformal description of the parameters that can be specified for a scenario is shown in table 4.1. The parameters are listed in groupings that are needed, or may optionally be included, to specify a scenario.

For explanatory reasons, this is a semiformal description and not the actual syntax used. We create a simpler syntax in order to make the implementation of the parser easier. Figure 4.2 shows an example of a parameter set specified with our syntax. This example will create a scenario with 40 nodes moving randomly in a field of 1000 by 1000 meters with a speed of between 5 and 10 m/s and no rest time. Of these 40 nodes, 10 generate traffic in form of bulk transfers being sent to random nodes selected from all 40 nodes.

In the example, all parameter values are constant except speed which has range values. We have made it possible to create more diverse scenarios by specifying *constant set* or *range* values to parameters. A 'constant set' is a list of constants. The scenario generator will then select one of these value each time a value is needed. For example, if the speed parameter is specified as '1,4,5', each time a node chooses a new direction and speed, it will choose a speed of either 1, 4, or 5 at random. A 'range' argument to a value is specified as a minimum and maximum value. For example, if the bulktransfer_amount is specified as '4096-8192', the bulk transfers will be from 4 to 8 kilobytes at random.

The following are the elements of the scenario generator that have been extended and completed during this project. The use of groups has been implemented. Furthermore

`scenario-spec = {`
`  simulation-time,`
`  field-size,`
`  group-spec,`
`  [ group-spec*, ]`
`};`
	To create a scenario, it is necessary to specify the time that should be simulated and the groups of node that should be included in the scenarios generated. At least one group of nodes must be specified. Finally, the size of the field must be specified. We only use rectangular fields, so the *field-size* parameters is specified as the length and width of the field.

`group-spec = {`
`  number-of-nodes,`
`  node-speed,`
`  node-rest-time,`
`  node-distance,`
`  [ stream-spec, ]`
`  [ transfer-spec, ]`
`  [ group-spec*, ]`
`  [ group-speed, ]`
`  [ group-rest-time, ]`
`  [ field-size, ]`
`};`
	A specification of group of nodes consists of a number of nodes, the speed with which the nodes should travel, what distance they should move at the time, and the time they should rest at waypoints. Optionally, traffic can be specified in form of streams or bulk transfers. Also optionally, a number of subgroups can be specified. Each subgroup is specified with the same parameters as *group-spec*. The *field-size* parameters is used to set the size of the field in which the nodes can move around. Optionally, the group movement can be bounded by specifying *field-size*, and how the group should move, *group-speed* and *group-rest-time* is stated.

`stream-spec = {`
`  destination-group,`
`  number-of-streams,`
`  packet-interval,`
`  packet-size,`
`  stream-duration,`
`};`
	A stream specification consists of a *destination-group* which is the group of nodes that the streams should flow to. The number of streams, will be the average number of streams that are active at any point during the simulation. The duration of each stream is also specified which gives a total number of streaming sessions of $\frac{number-of-streams \times simulation-time}{stream-duration}$. Finally, the packet size and the interval of packet transmission are specified.

`transfer-spec = {`
`  destination-groups,`
`  number-of-transfers,`
`  transfer-amount,`
`};`
	A specification of bulk data transfers consist of the destination group to which the data should flow, the amount of data to transfer, and the total number of transfers to perform throughout the simulation.

Table 4.1.: *The parameters that a scenario specification consists of.*

```
field_size 1000 1000          # Simulation area of 1000 by 1000 meters
simulation_time 250           # Simulate 250 seconds
group A 40                    # Create a group ‘A’ with 40 nodes
A.speed 5-10                  # All nodes, move between 5 and 10 m/s
A.resttime 0                  # Don’t stop and rest on waypoints

group B 10 A                  # Create a subgroup of A, B, with 10 nodes
B.bulktransfers_to A 100      # Let B create 100 bulktransfers to nodes in A
B.bulktransfer_amount 10000   # Send 10000 bytes in each bulk transfer
```

Figure 4.2.: *Parameter Set Example*

we have implemented the possibility to pick a range or a constant set for the value of the parameters. Finally, we have changed the movement model of the scenario generator. In [CEH01] each node selected a waypoint to move to. Now the node selects a distance and a direction, and moves accordingly. A node is reflected of the border, if it was to move out of the simulation area. This is to give a better distribution of nodes in the simulation area.

4.4. Summary

We use a scenario generator to generate test scenarios. The scenarios have certain characteristics obtained by a set of scenario parameters. The scenarios are random within the constraints of the set of scenario parameters.

The scenario generator enables us to create a wide range of random scenarios, which may, and often will, be different but yet conforming to the same scenario parameters. Thereby we ensure the generality of the results and remove the possibility of obtaining a good or bad result by chance.

5. Simulator, Setup and Simulation Procedures

In this chapter, we describe the simulator used (NS2) and the data extracted from the simulations. We use NS2 to simulate the wireless networks defined in the scenarios generated by our scenario generator, and use the data for analyses. First, we motivate the use of simulations and the choice of NS2. We describe the characteristics of the simulated universe, how a simulation is created, and the output from the simulator. Following we describe limitations concerning the way NS2 works. Finally, we describe how useful and applicable data is extracted from the output of the simulator. The chapter is concluded by a summary.

5.1. Motivation for using Network Simulator 2

We have used Network Simulator 2 [nsh] for simulating the wireless networks in this project. NS2 is the simulator of choice of other performance comparison of MANET routing protocols by simulations as described in section 1.4. One reason for using NS2 is that it performs complete enough simulations of all network layers from the transport layer through all layers to the physical layer. Furthermore, an implementation of AODV for NS2 already exists and we have an implementation of the OLSR protocol for NS2, primarily done in our previous project.

Finally, NS2 is free of charge and available for download.

5.2. Simulator

Network Simulator 2 is a discrete event network simulator, which is able to simulate many different kinds of networks, including both wired and wireless networks. In this section, we will only describe the parts of NS2 that we use to simulate wireless networks.

5.2.1. The Extent of the Universe

The physical layer simulated by NS2 is radio transmission. NS2 simulates the propagation of radio signals. In our simulations we use a *two-ray ground reflection model*. In this model,

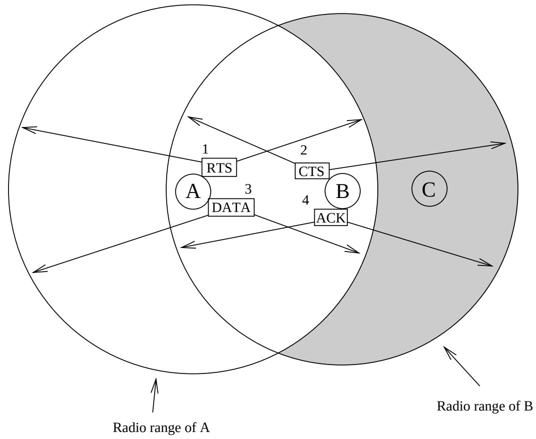


Figure 5.1.: *Hidden node scenario.*

the radio signals propagate directly between nodes and are also reflected off the ground. Nodes are placed 1 meter above the ground.

The media is set to emulate the Lucent Wavelan cards operating on the 914Mhz band. The preset values for transmitter power and receiving thresholds give a radio range of 200 meters when there are no obstacles to interfere with the transmission. The bandwidth is set to 2 Mbits/second. This is equivalent to the IEEE 802.11 standard.

The Medium Access scheme is the IEEE 802.11's distributed coordinated function (DCF). This is basically a Carrier Sense Multiple Access / Collision Avoidance access scheme (CSMA/CA). Collision avoidance is used, because collision detection is not possible in a radio network. This is because the node cannot "hear" anyone but itself when it transmits data. The collision avoidance is implemented as a RTS/CTS scheme, where the source node transmits a request-to-send signal which is answered with a clear-to-send signal from the destination node. Then, data is transmitted and the session is concluded with an acknowledgment from the destination node. This way, all nodes in radio range of the communicating nodes know that they should not begin transmitting, even if the can not hear the actual data, because transmitting would cause collision. This would occur for a so called "hidden node" as illustrated by figure 5.1 where node C cannot hear the data transmission but refrains from initiating a transmission because it hears the CTS from node B.

For broadcasting, data packets are simply transmitted on the media unless the node is waiting for other nodes to finish transmitting.

NS2 implements the Address Resolution Protocol (ARP) [Plu82] for IP to MAC address resolution.

On top of this, NS2 implements an entire TCP/IP stack with a variety of protocols. We use UDP for streaming traffic and TCP for bulk transfers.

We have implemented OLSR for NS2 and used an existing implementation of AODV. Neither of the two implementations use link layer notification, though both AODV and

OLSR can make use hereof, it does impose assumption and hence dependence on the link layer, and we therefore choose not to use this information.

5.2.2. Setup

Each simulation is run from a Tcl script describing the simulation parameters (such as radio propagation model and simulation area), the nodes that participates in the network and the traffic they generate. Each node's initial location and the time and type of movement is specified. The time and type of traffic is also specified before the simulation is run.

5.2.3. Output

The output from a simulation is a trace file containing a line for each event that has occurred during the simulation. The possible events are transmitting, receiving, dropping and forwarding of a packet. Events are recorded for all layers in the networking stack of each node. For example, the transmission of an UDP packet from one node to another, results in lines for transmitting from agent layer (transport layer), network layer, and medium access layer, and receiving on each of the layers in the destination node. Furthermore, receiving on the MAC layer, forwarding on the network layer, and transmission on the MAC layer again, is recorded for nodes that route the packet.

Each packet generated within NS2 during a simulation is assigned a unique identifier which allows the packet to be followed through the trace file. This allows us to measure packet delay as the time it takes for the packet to reach its destination from its transmission from the source node's application layer.

Furthermore, each protocol is assigned a "type", depending on which entity generated the packet. In our simulation there will be agent packets for normal data traffic, OLSR or AODV packets for routing control traffic, ARP packets for address resolution, and MAC packets for medium access control information.

An example of an extract from a trace file is shown in figure 5.2. The figure shows the trace of a single packet (packet number 3774) as it is transmitted from node number 5, routed through node number 27 and received at node number 25. To the right, the corresponding events in the network stack of each node is illustrated.

5.2.4. Limitations

The simulations performed by NS2 are completely deterministic. That is, for the same scenario, the output trace is always the same. In a real world MANET, there would be a lot of factors such as processor speed, memory latencies, cosmic ray etc. that would make the events occur in a slightly random fashion. Building the scenario generator in order to simulate many similar scenarios helps us avoid that the deterministic behavior of NS2 results in a unrepresentative data set.

The environment, simulated in NS2, is simplified. There are no physical obstacles for nodes and/or radio signal nor are there any external interference with the radio signals.

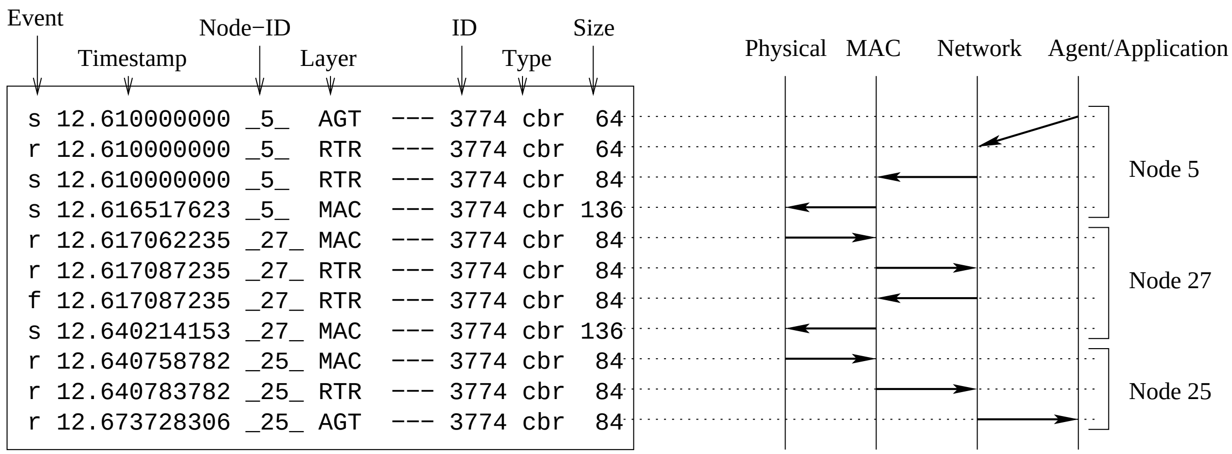


Figure 5.2.: *Example of an extract from a trace file.*

This presumably makes the transmission of data more reliable in the simulation than in the real world and causes all links to be bidirectional if both nodes have the same transmitter power.

NS2's scheduler does not time computation time used by the networking stack and routing protocols. However, in most reasonable cases, the computation time is negligible compared to the latencies in the transmission of packet over the wireless medium.

5.2.5. Technical Issues

During our work with the simulator, we found a bug in the routing queue code used in AODV when packets are queued during a route request. If the route request timed out the simulation would loop endlessly due to the non-removal of the first element in the queue, in a loop supposed to remove all timed-out packets. This occurred in approximately 2 out of 3 simulations. If the user of NS2 handled this situation by stopping and restarting the simulation, hoping for a scenario that would not cause the lock, he would leave out simulations where AODV performs badly, that is, when there are route timeouts.

Furthermore, we found bugs in AODV that would make the application crash under certain conditions due to assertions about the existence of entries in the routing table when a route breaks down. We fixed this by simply inserting routing entries with "down" status and let the existing AODV implementation decide whether a route should be found or not.

We also fixed a smaller issue affecting only the implementation of OLSR in some of the otherwise unused parts of the NS2 code.

Our Evaluation

Our general evaluation of Network Simulator 2 is that it is a comprehensive tool that simulates enough of a real network stack to provide a realistic picture of how a network would function in the real world.

However, there are some issues about NS2 that are important. The simulator package lacks a test suite and the code is of very varying quality because of the many contributors. This is confirmed by the number of bugs we have as described in section 5.2.5. On the other hand, NS2 has a widely established user base that should have caught some of the more grave model errors and general bugs in the code.

5.3. Measured Variables

In this section, we describe the data that we retrieve from the trace files from NS2. We have used custom made utilities to extract this data from the trace files.

Packet delay

The packet delay is measured as the time from the packet leaves the source node's agent layer until it is received at the destination node's agent layer. This includes time spent in queues and the time for transmission over the medium. Computation time required to process the packet is not included in the delay as NS2 does not take processing time into account in its scheduler.

Throughput

The throughput is measured as the number of application layer packets, data packets, that reach their destination as a fraction of the number of packet that are transmitted. Data packets that do not reach their destination may be dropped for any reason whatsoever.

Drop reasons

We have also recorded the reasons for dropped data packets. The most interesting drop reason is route unavailability, that is, a 'no-route-drop'. When using OLSR, a packet will be dropped if a route for the packet's destination is not already available when the packet is to be routed. When using AODV, a packet will be retained until the route request succeeds or times out. If the route request times out, the recorded drop reason is *no route-drop*.

Bandwidth

We count both the packets and the bytes of all data and control packets transmitted on the MAC layer, as these are data actually transmitted over the medium and hence consuming bandwidth. This allows us to get data such as the amount of bandwidth used on control traffic (control overhead) and how much application layer data is actually transmitted over the medium.

Transfer delay

For some of the simulations, we have recorded the total transfer time of a TCP bulk transfer. The time is measured from the initiation of the transfer, that is, when the node wants to begin the transfer, and until the last acknowledgment is received at the source node. Therefore, these measurements include the time used to set up the route (when applicable). The transfer delay is more comparable to the user's experience of delays than the packet delay, because it will be almost equal to the delay that the user experiences from, for example, when he clicks on a link in his browser and until the web page is completely loaded.

5.4. Summary

We use a discrete event network simulator, NS2, for numerous simulations of Manets. Each simulation is run from a Tcl script and outputs a trace file from which results are extracted.

We extract information about the packet delay, the throughput, the bandwidth, the drop reasons, and in some cases the transfer delay, and evaluate upon these results.

6. Statistical Methods

In this chapter, we will describe the statistical methods we use to analyze the simulation data. We use statistical methods to ensure the validity and generality of the results. First, we motivate the use of statistics. We then describe the descriptive measures used to analyze sets of sample data. Finally, we describe the chi-square test of independence used to analyze the results' dependencies of the various simulation parameters. This chapter is concluded by a summary.

6.1. Motivation for using statistics

Our aim is to model scenarios with certain characteristics, but at the same time eliminate the possibility of results appearing by chance through a favorable or unfavorable pick of a specific scenario. We achieve this by running numerous simulations of random scenarios with the same specific characteristics and using statistical tools to analyze the data extracted from the simulations.

According to [Mit97], at least 30 scenarios of each situation should be performed to have a good approximation of the population[1] and provide enough information for a set of sample data. In our case the population is the total number of all possible scenarios with the specific characteristics. That is, with the same scenario parameters.

We use descriptive measures[2] to describe each set of simulations with the same parameter set, a test set. In some cases it is relevant to test the results of dependency of the varied parameter to compare the results from different test sets, for example with and without enforced jitter on the transmission of control packets. For this we use the chi-square test of independence to calculate the possibility of results appearing by chance. Unless stated otherwise the formulas are from [ASW96]. We developed custom made utilities to calculate the descriptive measures and to perform the chi-square test.

6.2. Descriptive Measures

We perform a number of scenario simulations of each situation with particular characteristics, for example different degree of mobility. From these simulations we obtain a set

[1] A population is the entire group of all possible situations from which the measures are taken.

[2] A descriptive measure is a single number that provides information about a set of data.

of sample data which is analyzed using descriptive measures. We use measures of central tendencies and of diversion.

Measures of Central Tendency

The purpose of these measures is to determine the sample mean, $\overline{x}$, which is the numeral average of the data.

Measures of Diversion

The purpose of these measures is to show the dispersion in the results. That is, the tendencies of data values to scatter about the mean. We determine the variation, s^2, and the standard deviation, s, as follows:

$$\text{sample variance} = s^2 = \frac{\sum (x - \overline{x})^2}{n-1}$$

$$\text{sample standard deviation} = s = \sqrt{\frac{\sum (x - \overline{x})^2}{n-1}}$$

Because we do not examine the entire population, (that would be all possible scenarios that comply to a specific set of scenario parameters) but only samples (a random subset of the population), we use the formula for the sample variance and standard deviation and not the formula for population variance and standard deviation[3]. Because the data sets may have different means, we also calculate the relative variation within each set, the coefficient of variation, CV. The coefficient of variation is determined as:

$$\text{coeffiecient of variation} = CV = \frac{s}{\overline{x}} \times 100$$

We use the standard deviation and the coefficient of variation to describe the dispersion in the results.

6.3. Chi-Square Test of Independence

When comparing different results it is important to be able to determine the probability that the results appear by chance, that is, if there exists a dependency between the results and the varied parameters or not. We use the chi-square test of independence for this purpose in this project. To illustrate the use of the chi-square test of independence, we will use a fictive example of simulations with and without the use of enforced jitter. When testing the dependencies we always advance the following two general hypotheses:

- H_0: The results are independent

[3] The formulas are $\sigma^2 = \frac{\sum (x-\mu)^2}{N}$ and $\sigma = \sqrt{\frac{\sum (x-\mu)^2}{N}}$, where N is the number of the entire population and μ is the population mean.

Noroute drop				
Interval	0-1000	1001-2000	2001-3000	Row total
With jitter	60 (60)	20 (30)	40 (30)	120
Without jitter	40 (40)	30 (20)	10 (20)	80
Column total	100	50	50	200

Chi-square	16.66
Degree of freedom	2
$Q(\chi^2 \vert df)$	0.9998

Table 6.1.: *Contingency table and dependency calculation for a fictive jitter test.*

- H_a: The results are dependent

The results of the tests are summarized in a contingency table with the number of occurrences, O. Table 6.1 shows such a table with the number of packets dropped because of route unavailability from our fictive example. The numbers in parenthesis are the expected values, E, with no dependencies.

First the expected value is calculated as $E = \frac{(row\ total)(column\ total)}{n}$ where the *row total* and *column total* are those of the particular cell, and n is the total number of occurrences. In the example the expected value of occurrences with jitter in the interval 0-1000 will be: $E = \frac{120 \times 100}{200} = 60$.

Chi-square, χ^2, is then calculated as $\chi^2 = \sum \frac{(O-E)^2}{E}$. In the example χ^2 is 16.66. Next we calculate the probability that H_0 is true – that is, the probability that the results are independent. To do this we need the degree of freedom, df[4]. This is calculated as $(k-1)(m-1)$, k being the number of columns and m being the number of rows. In the example df is 2. The probability integral is $P(\chi^2|df)$ and is calculated as follows[5].

$$P(\chi^2|df) = 2^{\frac{-1}{2}df}\{\Gamma(\frac{1}{2}df)\}^{-1}\int_0^{\chi^2} e^{-\frac{1}{2}x}x^{\frac{1}{2}df-1}dx \text{ [PH76]}$$

The probability that the results are dependent and that H_a is true, $Q(\chi^2|df)$, is $1 - P(\chi^2|df)$, because the two hypotheses are mutually excluding. In the example this probability is 0.9998, which means that there is a 0.02% chance that the results are independent of the use of jitter.

We use this value to determine whether the results appear by chance or whether the change in scenario parameters affect the results.

[4]The degree of freedom expresses the number of options available within a variable or space.
[5]The Γ function is $\Gamma(\alpha) = \int_0^\infty y^{\alpha-1}e^{-y}dy$ [BL96].

6.4. Summary

This chapter briefly describes the statistical methods we will use to analyze the results of the simulations. For each set of sample data, we calculate descriptive measures: the mean, $\overline{x}$, the standard deviation, s, and the coefficient of variation, CV. We will furthermore use the chi-square test of independence to calculate the probability of results' dependencies of the parameters in some cases.

We use the descriptive measure to make the large amount of data produced by the simulations comprehensible and to have a representative result to conclude upon. The measures of central tendency are used to get an idea about the average performance of the network/protocol. The measures of dispersion are used to get an idea about the stability of the results from a particular type of scenario. We use the chi-square test of independence to ensure that there is a low probability that our conclusions are invalidated because of results that has appeared by chance.

7. OLSR Performance

In this chapter, we will present the results of the simulations which are designed to test the impacts of the improvements to the OLSR protocol. Namely, the introduction of jitter, piggybacking, link hysteresis, and adjustments of the control message intervals, as described in section 2.2. First, we describe the default test configuration of the tests. Next, we describe the tests with jitter, piggybacking, control message intervals, and link state detection, respectively. Each test section will describe a test set with the following elements: the thesis that is to be tested, explanation of the varied parameters, results of the simulations, and finally, an analysis of the results. The chapter is concluded by a summary of the results of the analyses.

7.1. Test Configuration

The theses tested in this chapter, are those stated in chapter 2 which exclusively concern the performance of OLSR. To examine these theses, a test set of at least 30 scenarios (as described in chapter 6) for each set of scenario parameters were generated and run in NS2. The scenarios were generated by the scenario generator described in chapter 4 from the default parameters stated in table 7.1, unless otherwise stated. The measures used in the result sections are those stated in section 5.3, although we may leave our some measures in tests when they are not relevant. For each test set we state the results used for analyzing the particular set. For explanatory reasons, the results are shown as numerals, graphs, and figures depending on the context. The complete set of the results in numerals, is included in appendix B.

7.2. Jitter

When using fixed intervals between transmitting control messages, we observed in our practical experiments that numerous packets were lost due to collisions. We anticipate that introducing jitter in the transmission of control messages will have effects on the performance of OLSR.

Jitter is enforced on both types of control message, that is on both the hello and the TC messages. The jitter is implemented by adding a random amount of time, α, to the control message interval, I, and transmitting the control message after $I + \alpha$ seconds. Test sets with jitter and without jitter were performed. In the simulations with jitter, α was

Number of nodes	50 nodes
Field size	1000 × 1000 meters
Simulation time	250 seconds
Node speed	1-5 meters/second
Node resttime	0-6 seconds
Node distance	1000 meters
Number of streams	25 streams
Packet size	64 bytes
Packet interval	0.10 seconds
Stream Duration	250 seconds

Table 7.1.: *Default parameters used in the simulations*

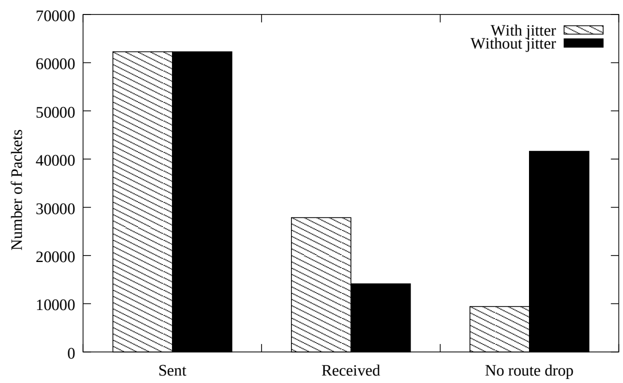


Figure 7.1.: *Number of packets sent, received and dropped due to route unavailability with and without jitter.*

chosen from the interval $[-0.5; 0.5]$. In the simulations without jitter, α was 0. The hello message interval is 2 seconds and the TC message interval is 5 seconds as recommended in the OLSR draft [JMQ+01]. At the beginning of the simulation, the nodes are "turned on" (and begin transmitting hello messages) randomly within the period of a hello interval.

Results

Figure 7.1 shows the average number of packets that were sent and received, as well as those dropped because of route unavailability. Without jitter half as many packets reached their destination as with jitter, while the amount of packets lost due to route unavailability was more than four times the amount of the simulations with jitter.

Table 7.2 shows the descriptive measures of packets dropped because of route unavailability and packets received, respectively. The standard deviation and the coefficient of variance are consistently higher without jitter than with jitter, meaning that both the regular dispersion and the relative dispersion is higher without jitter.

	Noroute drop		Received	
	With jitter	Without jitter	With jitter	Without jitter
Mean	9,430	41,600	27,900	14,100
Standard deviation	2,630	14,100	3,810	6,780
Coefficient of variation	27.92 %	33.93 %	13.67 %	48.00 %

Table 7.2.: *Descriptive measures from jitter tests.*

	Without jitter	With jitter
TC flooding number	12.8 nodes	32.1 nodes
OLSR overhead	2560 bytes/second	3850 bytes/second
Packet delay	0.174 seconds	0.596 seconds

Table 7.3.: *Selected results from jitter tests.*

Analysis

The higher throughput is consistent with the lower number of packets lost due to route unavailability. The fact that the drop rate caused by route unavailability is significantly lower with enforced jitter indicates that more nodes in the MANET have enough topology information to have routes to all nodes, because that more TC messages arrive at the nodes. In table 7.3 the average TC flooding number is shown. The TC flooding number is the average number of nodes a TC message reaches in the network. It is clear that the topology information is diffused to a greater part of the network and thereby nodes will have knowledge of a larger number of nodes in the network. This also means that the overhead OLSR produces will be higher with jitter than without. This overhead is, however, a desired and necessary overhead, because we want the topology information to be diffused as far out in the network as possible. The overhead with jitter is 50% higher, as seen in table 7.3, but still relatively small.

The average packet delay without and with jitter is show in table 7.3. The larger delay with jitter can be explained by the larger throughput of data packets. The packets that get through without jitter are those with short routes, which are not as dependent on topology information, as those with long routes. Not only do more packets get through with jitter, but packets destined for nodes farther away get through (which they would not, otherwise). Thereby, the average delay is larger than without jitter.

Figure 7.2 shows the number of simulations as a function of the number of packets dropped due to route unavailability to visualize the dispersion of results without jitter compared to that with jitter. The figure shows that without jitter, the dispersion is high, that is, the simulations are scattered around in many intervals, while with jitter, the simulations are concentrated in few intervals.

Likewise figure 7.3 shows the dispersion of number of packets received with and without

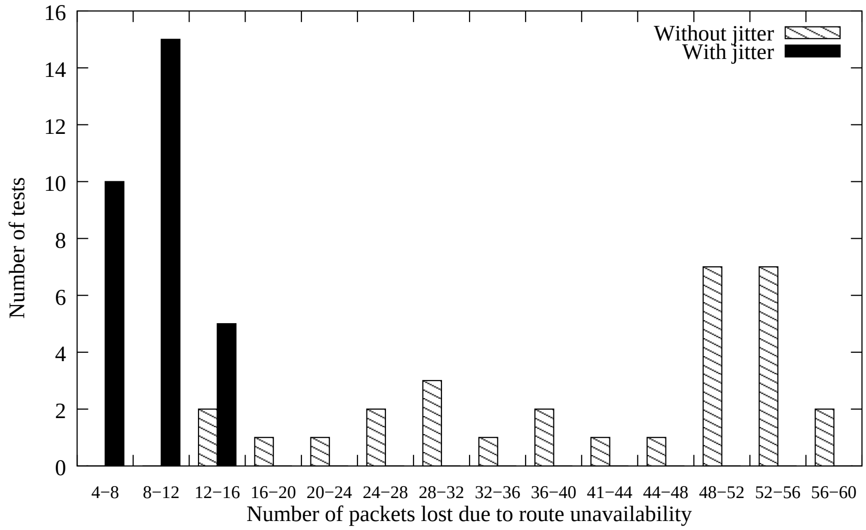


Figure 7.2.: *The number of tests as a function of number of packets lost due to route unavailability. The number of packets is showed in intervals of thousands.*

jitter. The figure shows that with this, the number of received packets is concentrated in four intervals, while without jitter, they are dispersed in 7 intervals.

The figures and the measures in table 7.2 show that the dispersion of the results without jitter was substantially larger than those with jitter. This means that the performance with enforced jitter is not only better, but also more stable than without jitter.

The chi-square test of independence on both the number of received packets and the number of packets lost due to route unavailability shows that the probability that the results are dependent on the use off jitter 1.0000 (when rounded off due to calculation imprecision). This indicates that there is a very high dependency between the throughput and the use of jitter. See tables B.3 and B.4 for a calculation of the results.

The general conclusion is that jitter improves the throughput and lower the number of packets dropped because of route unavailability. The cost of enforcing jitter only the effort to implement it.

7.3. Piggybacking

The preliminary simulations indicated that piggybacking control messages had an impact on the number of control packets dropped collisions, and thereby indirectly on the throughput. We therefore anticipate that the simulations will show an improvement in the performance of OLSR when piggybacking control messages. That is, higher throughput and lower overhead.

Piggybacking of control messages is enforced by holding back incoming control messages that are to be retransmitted for up to a predefined amount of time, holdback time, before

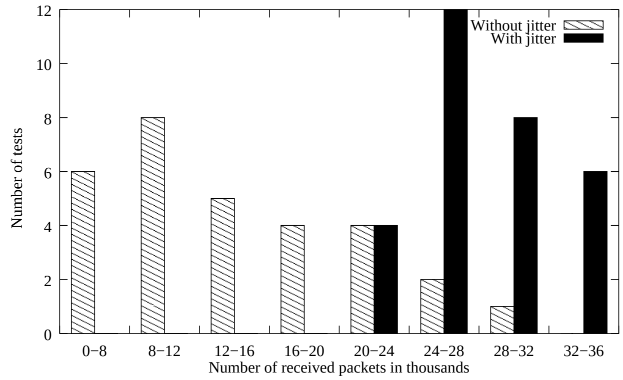


Figure 7.3.: *The number of tests as a function of number of packets received. The number of packets are showed in intervals of thousands.*

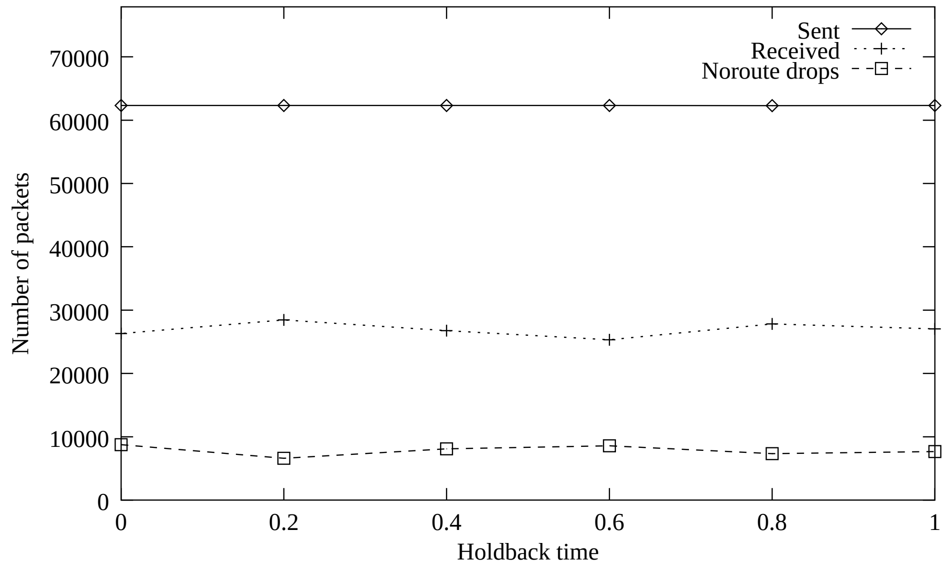
retransmitting them. If a locally generated control message is transmitted from the node before the end of the holdback time, the incoming messages in the buffer are transmitted, piggybacked with the outgoing message. Test sets with holdback time of 0.0, 0.2, 0.4, 0.6, 0.8 and 1.0 seconds were performed. A holdback time of 0.0 is equivalent to not implementing piggybacking because incoming message are retransmitted as soon as they arrive, if the node is MPR to the nodes, where the messages are sent from. To make sure that the impact shown in the results were due to piggybacking, three test sets were run: One with jitter and with piggybacking, one with only piggybacking, and one with jitter and piggybacking and no mobility.

Results

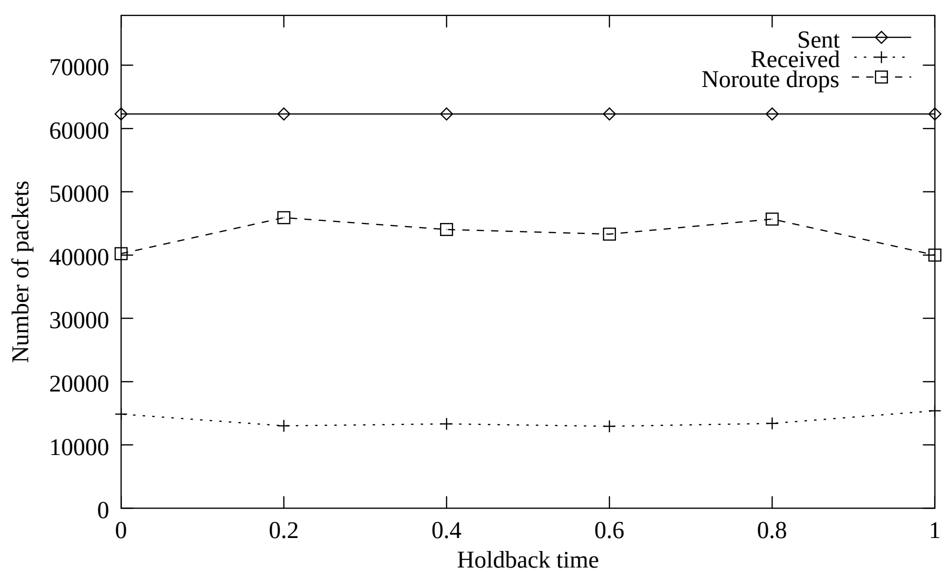
Graph 7.1 shows the average number of packets that were sent, received, and dropped due to route unavailability in the situation where the nodes enforced both jitter and piggybacking with variable holdback time. The graph shows some fluctuation. The throughput is 8% and 6% higher at holdback times of 0.2 and 0.8 seconds than with no holdback time. The number of packets lost due to route unavailability are 22 and 16% lower than without piggybacking.

Graph 7.2 shows the average number of packets that were sent, received, and dropped due to route unavailability with variable holdback time, in the situation where the nodes enforced only piggybacking and no jitter. The number of lost packets due to route unavailability is high at all holdback times with peaks at holdback times of 0.2 and 0.8 seconds and lows at no and maximum holdback times.

Graph 7.3 shows the average number of packets that were sent, received, and dropped due to route unavailability with variable holdback time, in the situation where the nodes enforced both jitter and piggybacking, and without mobility. The graph shows slight fluctuation. However, there is a 5% higher throughput and 24% lower droprate due to route unavailability than with no piggybacking.



Graph 7.1: *The number of packets sent, received, and dropped due to route unavailability with both jitter and piggybacking with variable holdback time.*



Graph 7.2: *The number of packets sent, received, and dropped due to route unavailability with only piggybacking with variable holdback time.*

	Received					
Holdback time	0.0	0.2	0.4	0.6	0.8	1.0
Mean	26300	28400	26800	25300	27800	27000
Standard deviation	4480	4380	3970	4000	4660	4850
Coefficient of variation	17.0 %	15.4 %	14.5 %	15.8 %	16.8 %	17.9 %
	Noroute drop					
Holdback time	0.0	0.2	0.4	0.6	0.8	1.0
Mean	8740	6610	8100	8560	7330	7650
Standard deviation	2840	2720	2250	3010	3090	2650
Coefficient of variation	32.5 %	41.1 %	27.8 %	35.1 %	42.1 %	34.6 %

Table 7.4.: *Descriptive measures from piggybacking tests with jitter.*

	Received					
Holdback time	0.0	0.2	0.4	0.6	0.8	1.0
Mean	14900	13000	13300	13000	13400	15500
Standard deviation	7790	6580	6360	8000	6180	8870
Coefficient of variation	52.3 %	50.5 %	47.7 %	61.6 %	46.1 %	27.0 %
	Noroute drop					
Holdback time	0.0	0.2	0.4	0.6	0.8	1.0
Mean	40200	45900	44000	43300	45700	39400
Standard deviation	15600	11500	13500	16300	11600	17200
Coefficient of variation	38.7 %	25.1 %	60.6 %	37.6 %	25.4 %	43.8 %

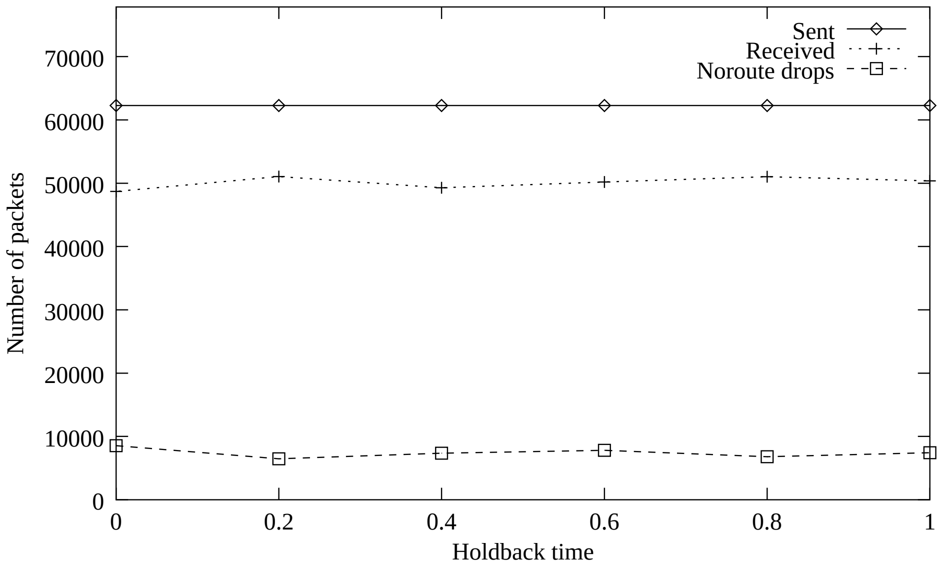
Table 7.5.: *Descriptive measures from piggybacking tests with no jitter.*

Analysis

The simulations show that there is a small effect in throughput of piggybacking as seen in tables 7.4, 7.6, and 7.6. This can be explained by the fact that fewer packets are lost due to collisions, but each packet contains more messages, so the same amount of messages are lost.

Graphs 7.4 and 7.6 show that the amount of control traffic becomes smaller, as expected, when the holdtime becomes longer in the scenarios where jitter is enforced. This is expected, as there are more messages in each packet, making the overhead smaller, because fewer packet headers are transmitted on the medium. As seen in graph 7.5 the amount of control traffic without jitter becomes smaller when piggybacking is used, but shows little fluctuation at different holdback times.

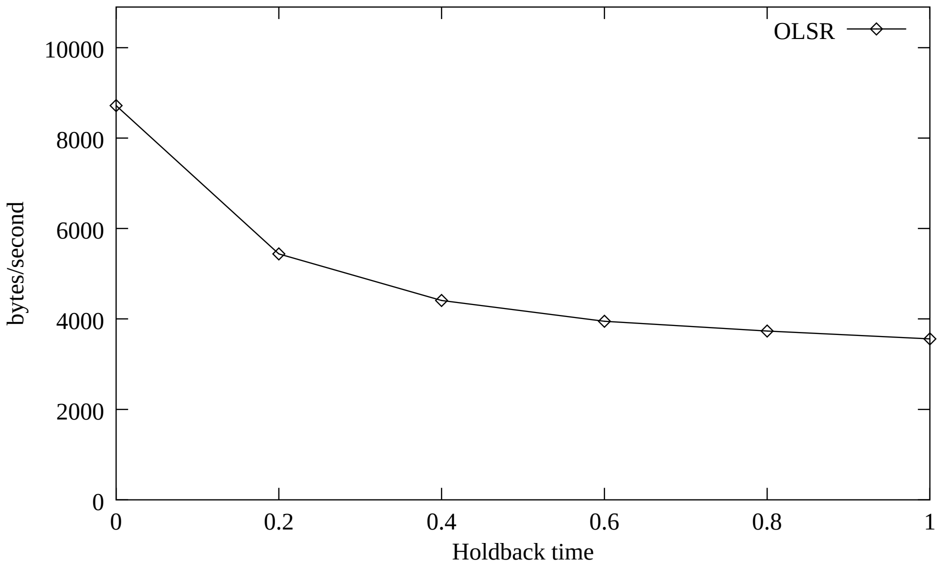
We have performed the chi-square test of independence on the number of received packets and the number of packets dropped due to route unavailability, and the percentages range from 60% to 96%, thus the numbers do not lead to any solid conclusions. In the test



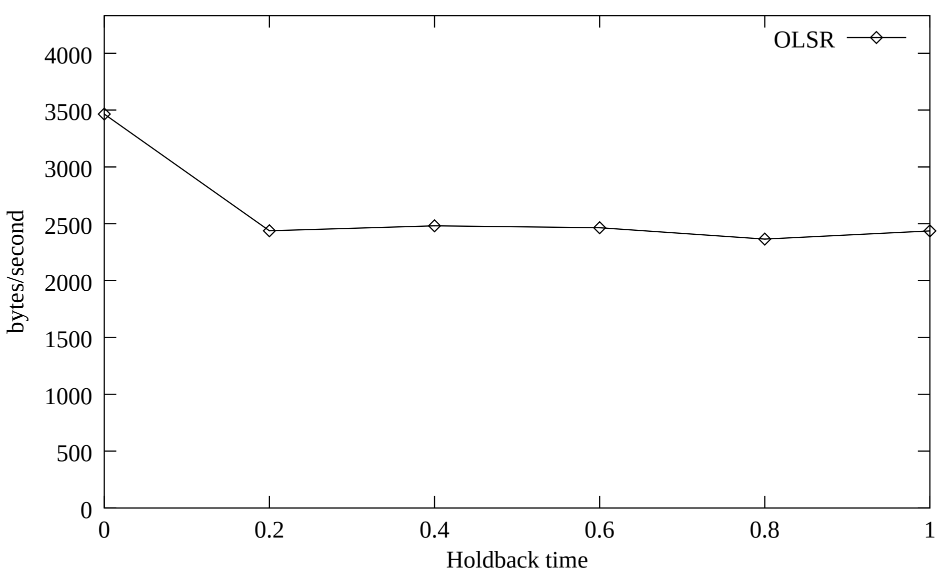
Graph 7.3: *The number of packets sent, received, and dropped due to route unavailability with piggybacking and jitter in a network without mobility and with variable holdback time.*

	Received					
Holdback time	0.0	0.2	0.4	0.6	0.8	1.0
Mean	48700	51100	49300	50200	51000	50400
Standard deviation	5560	6440	5780	5790	6520	7150
Coefficient of variation	11.4 %	12.6 %	11.7 %	11.5 %	12.8 %	14.2 %
	Noroute drop					
Holdback time	0.0	0.2	0.4	0.6	0.8	1.0
Mean	8530	6480	7340	7800	6800	7432.6
Standard deviation	4500	4110	4120	5190	5010	6360
Coefficient of variation	52.7 %	63.5 %	56.1 %	66.5 %	73.7 %	85.6 %

Table 7.6.: *Descriptive measures from piggybacking tests with jitter and no mobility.*



Graph 7.4: *The amount of control traffic with piggybacking and jitter.*



Graph 7.5: *The amount of control traffic with piggybacking and without jitter.*

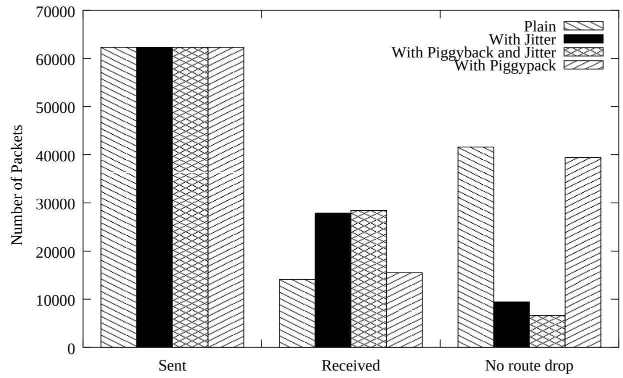


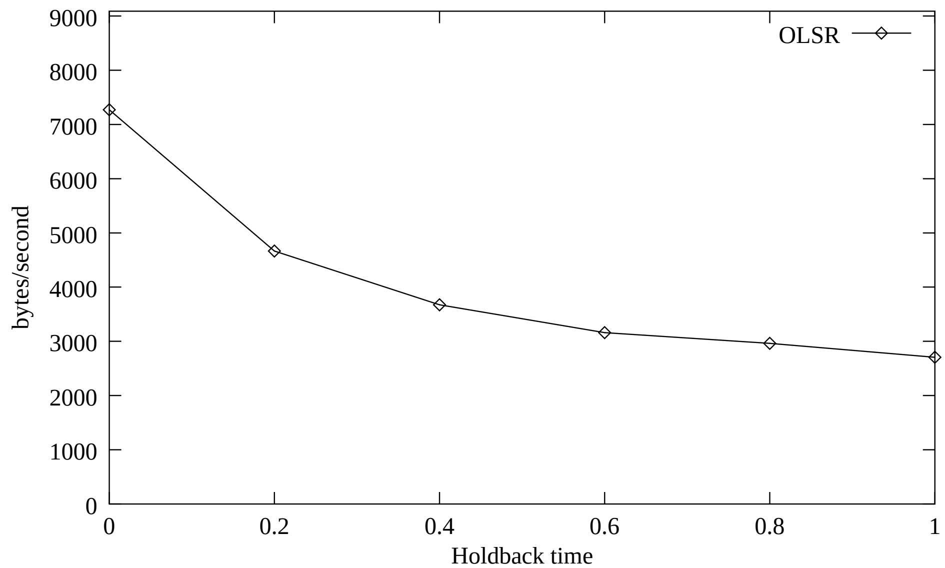
Figure 7.4.: *The average number of packets from test sets with jitter, with piggybacking, with both piggybacking and jitter, and with no jitter or piggybacking.*

with piggybacking and with jitter the dependency of the number of packets dropped due to route unavailability on the use of different holdback times is 96%, meaning that when applying piggybacking on a network, which already use jitter, an effect is very likely to appear. See tables B.7, B.8, B.11, B.12, B.15 and B.16 for a calculation of the results. It seems that the best effect of piggybacking is when it is enforced together with jitter and with mobility.

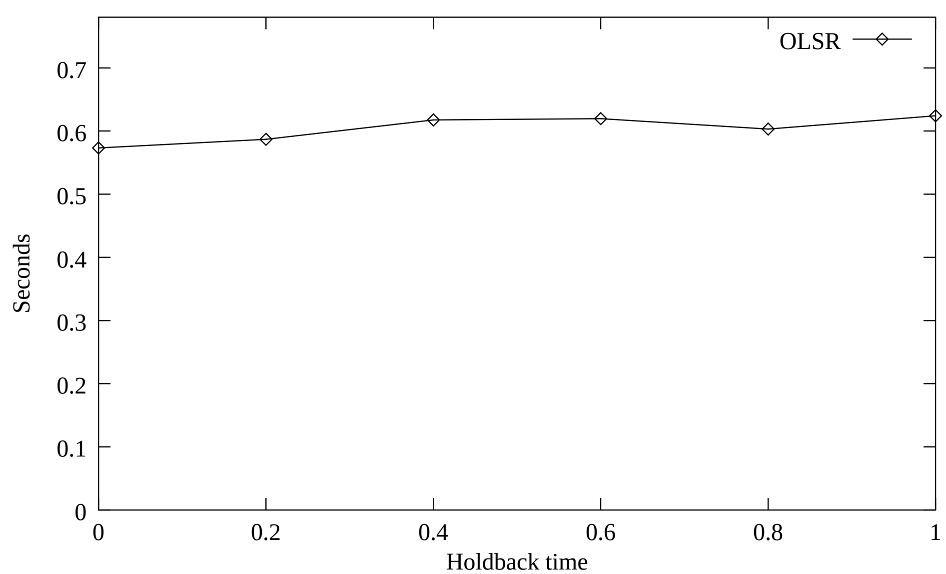
Graph 7.7 shows that that the average packet delay increases slightly when applying piggybacking in a setting with jitter. The increase from not using piggybacking to a holdback time of 1.0 second is 9%. Graph 7.9 show a slight decrease in packet delay as the holdback time get larger. This could be because the short routes are more stable, but the long routes take longer time to be discovered, and as the packet delay is larger for packets that are destined farther away, the average packet delay will be smaller if a smaller fraction of packets with long routes reach their destination. However the fluctuation in the graph makes it hard to conclude upon. Graph 7.8 shows great fluctuation in the average packet delay, which just confirms the earlier result that jitter is very important to get a stable result.

Figure 7.4 shows the average number of packets sent, received, and dropped due to route unavailability from the test sets with plain OLSR, with enforced jitter, with jitter and piggybacking (holdback time: 0.2 seconds), and finally without jitter but with piggybacking (holdback time: 1.0 seconds). We have chosen to show the results with those holdback times, because they gave the best results in the respective tests. The figures shows that piggybacking works best in combination with jitter.

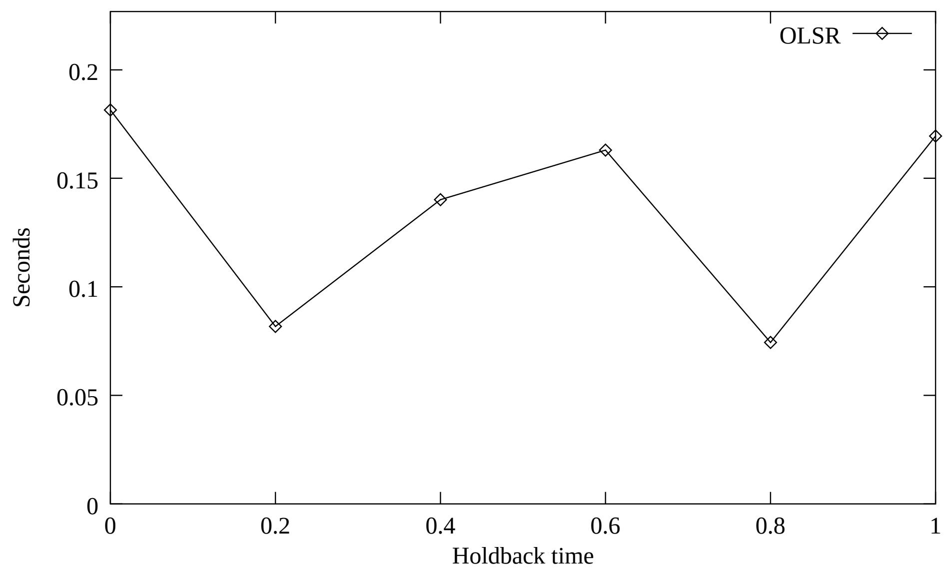
We recommend that piggybacking is applied. The positive effect on the throughput is small, but there is no cost of enforcing piggybacking and the overhead becomes smaller.



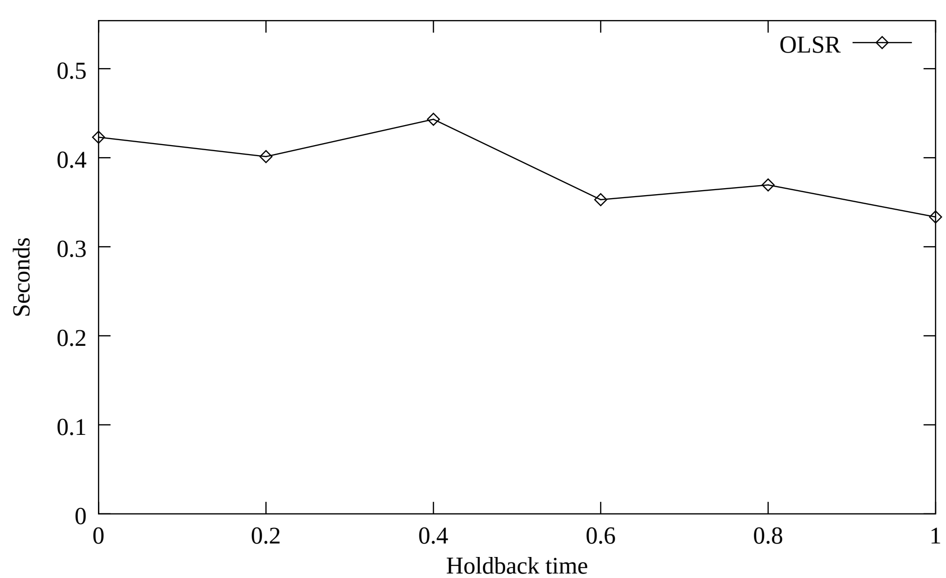
Graph 7.6: *The amount of control traffic with piggybacking and jitter and without mobility.*



Graph 7.7: *The average packet delay with piggybacking and with jitter.*



Graph 7.8: *The average packet delay with piggybacking and without jitter.*



Graph 7.9: *The average packet delay with piggybacking and jitter and without mobility.*

7.4. Control Message Intervals

The constant values for hello and TC intervals specified in the OLSR draft [JMQ+01] are chosen mainly on the analytical evaluation of the advantages and disadvantages of having higher or lower intervals. We have tested the OLSR protocol with different settings for these constants in order to check if more optimal values exists.

7.4.1. The Hello Interval

We performed simulations where the OLSR protocol uses a variable hello interval. The hello interval is the interval between two hello messages. Test sets with hello interval of 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, and 3.5 seconds were performed. The neighbor hold time was kept at 3 times the hello interval, which is 6 seconds when the interval is as defined by the OLSR draft [JMQ+01]. The test was made with jitter on the transmission of control packets because of the results stated above. At a hello interval of 0.5 seconds the jitter interval is ± 0.25s, and ± 0.5s at the other intervals. Piggybacking was not enabled for hello messages in this test. That is, hello message are sent immediately when they are generated.

Simulation Results

The number of sent and received packets per second is shown in graph 7.10. The graph shows that the throughput is not affected much by changing the hello interval, except at high hello intervals where it drops a little.

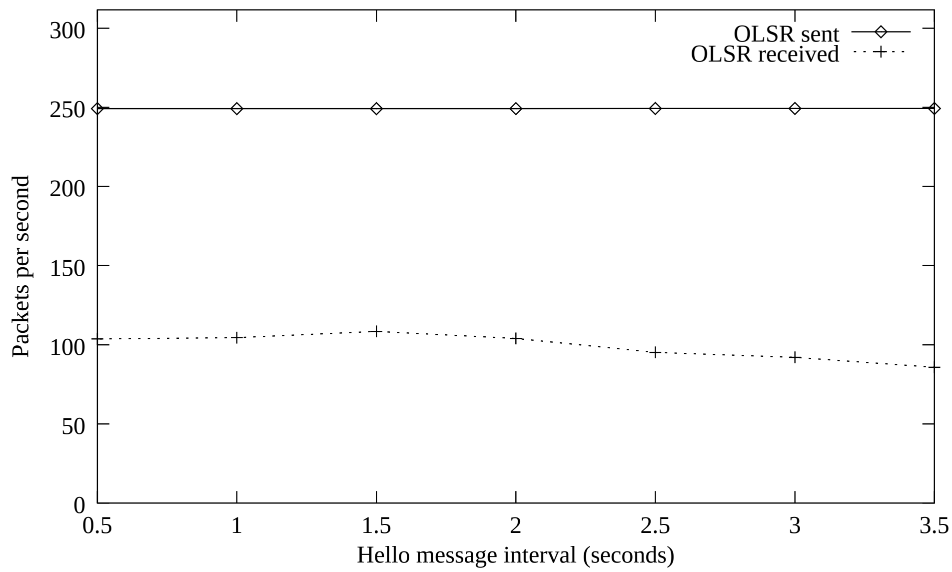
The average packet delay is shown in graph 7.11. It is hard to see any tendency in the packet delay when the hello interval is changed.

Analysis

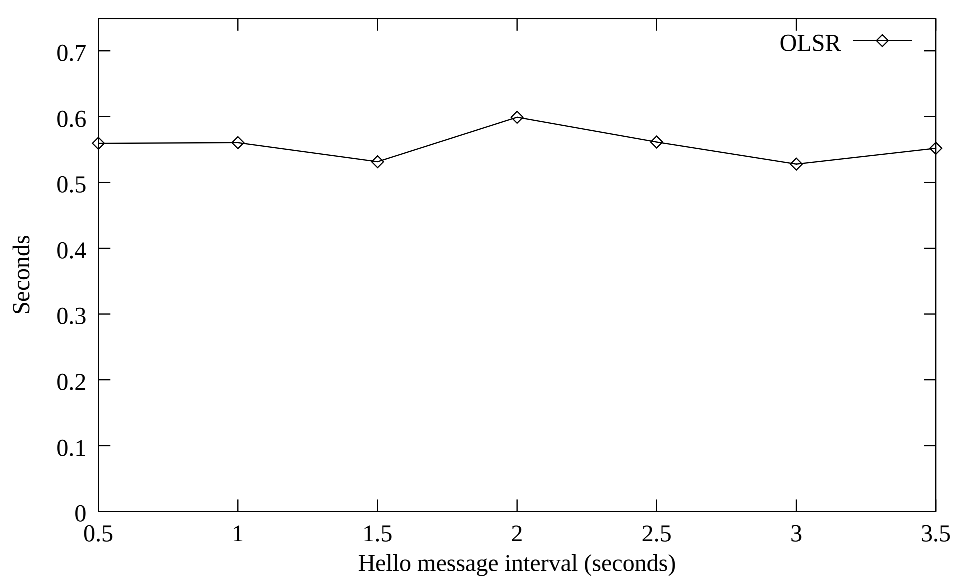
The graphs shows that the throughput and average packet delay is not affected much by changing the interval. At a higher hello interval, the throughput drops a little. This is because the protocol adapts slower to topology changes. At a lower interval than 2.0 seconds, the throughput neither increases nor decreases. This may be because the increased ability to adapt to topology changes is outweighed by the additional load in control traffic put on the network. The control traffic in number of bytes per second is shown in graph 7.12. At a hello interval of 0.5 seconds, the amount of control traffic is around 14 kilobytes which is a little more than 1/20 of the available bandwidth.

7.4.2. The TC Interval

We have performed simulations with variable TC message intervals to test whether performance can be improved by using other values than the 5 seconds specified by the OLSR draft [JMQ+01]. Test sets with TC message intervals of 1 to 12 seconds in intervals of 1 second were performed. In all cases, a random jitter of at maximum 25% of the TC



Graph 7.10: *Number of sent and received packets with variable hello message interval.*



Graph 7.11: *Average packet delay with variable hello message interval.*

message interval was used. Locally generated TC messages were never piggybacked in this test, while incoming TC message from other nodes could be held back in an attempt to piggyback for up to 0.2 seconds. The topology hold time was set to 3.2 times the TC message interval in all settings. This corresponds to the settings defined by the OLSR draft of 5 seconds TC message interval and 16 seconds topology hold time.

Results

The throughput for the simulated networks is plotted for each tested TC message interval in graph 7.13. The throughput is not affected much by changing the TC message intervals, except when the interval is high and the throughput drops a little.

The average packet delay is plotted in graph 7.14. The graph shows that the average packet delay increases slightly when the TC message interval is increased.

The amount of control traffic sent on the medium is shown in graph 7.15.

Analysis

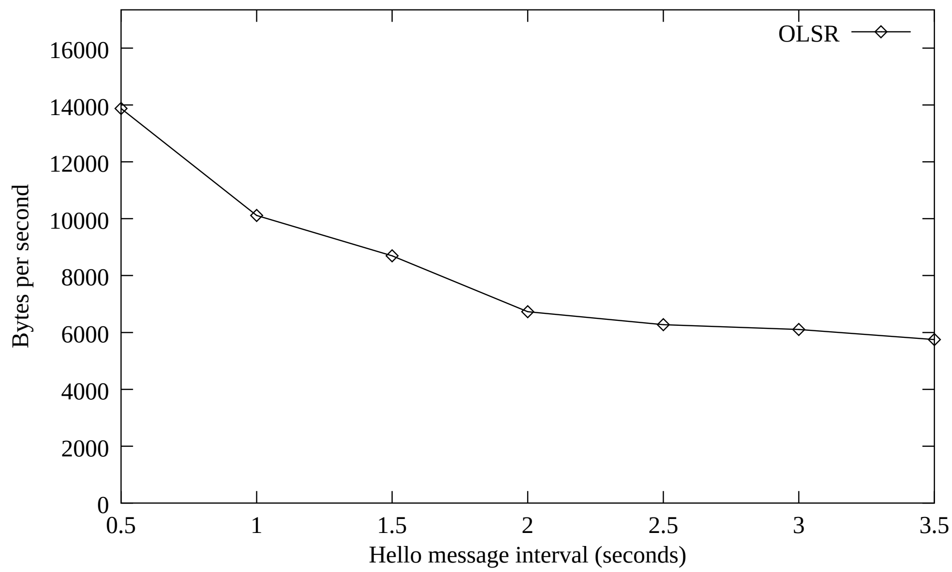
The throughput is mostly unchanged when adjusting the TC message interval. At high TC intervals, the throughput drops only a little. The reason that the throughput drops with longer TC message intervals is because the topology information in each node is updated less frequently and hence is more likely to be outdated.

The average packet delay is also very little affected by changing the TC message interval, but it does get a little lower value when the topology information is updated more often and a little higher when the information is updated less frequently. This is expectable: The optimality of the routes depend on how correct topology nodes has. When this topology get updates less frequently, the routes gets less optimal and hence the transmission delays get longer.

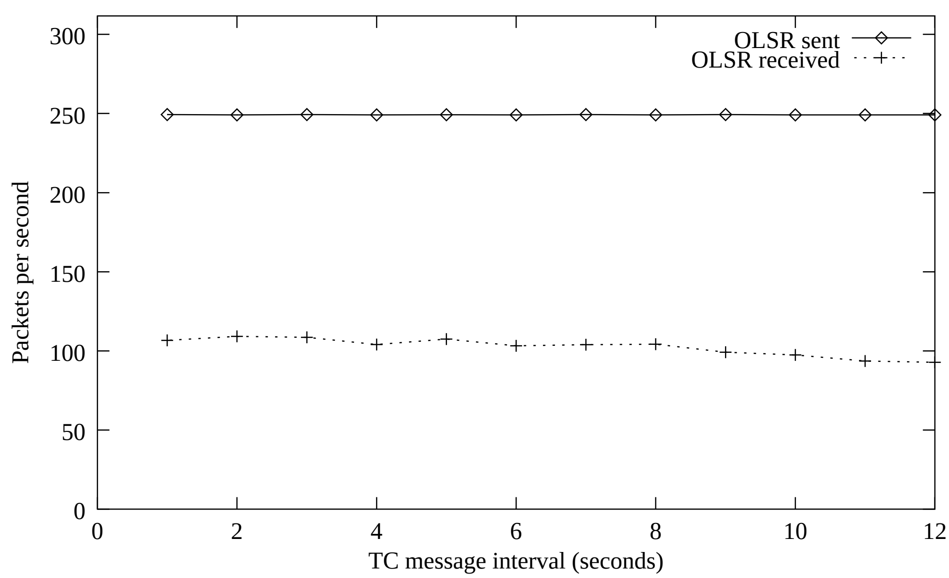
The gain from lowering the TC interval could be expected to be more significant when there is a more dynamic topology in the network because then it is more important to get correct topology information faster. We ran the same test with TC intervals from 1 to 7 seconds with mobility 12.5 meters per second. The throughput is shown in graph 7.16. Not even in this situation is the throughput affected by lowering the TC message interval.

7.5. Link Stability

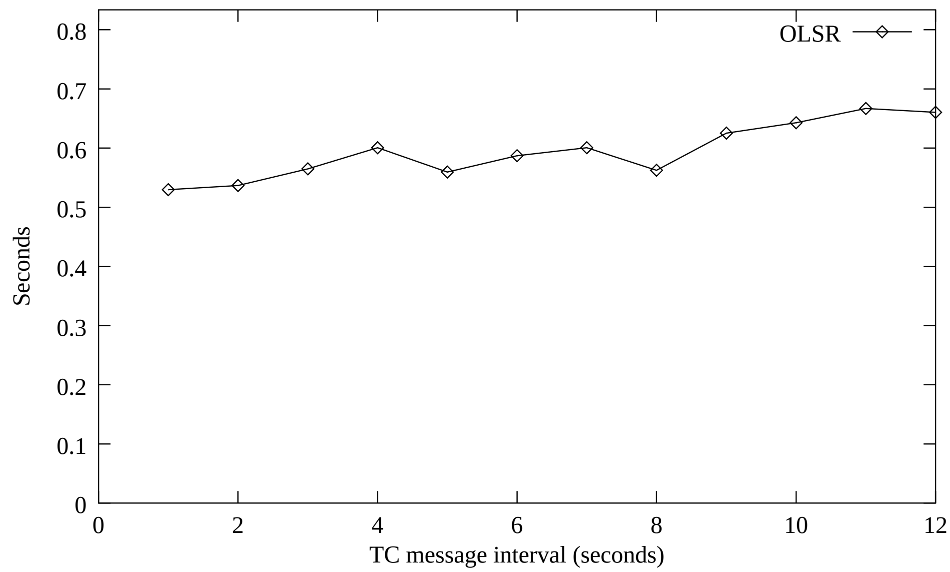
Because the links in a wireless network are relatively unstable and can be very sporadic if the distance between the two nodes is near the radio range of the antennas, it is important to investigate routing methods that take the quality of link into account. A method, suggested to us by one of the designers of the OLSR protocol, is to use a simple link hysteresis where it is required to receive more than 1 hello message in order to qualify the link as usable (after which the usual asymmetric/symmetric negotiation is done). With the current OLSR draft, only 1 hello message must be received in order to qualify the link as asymetric or symmetric. If instead we require that at least 2 hello messages within 3



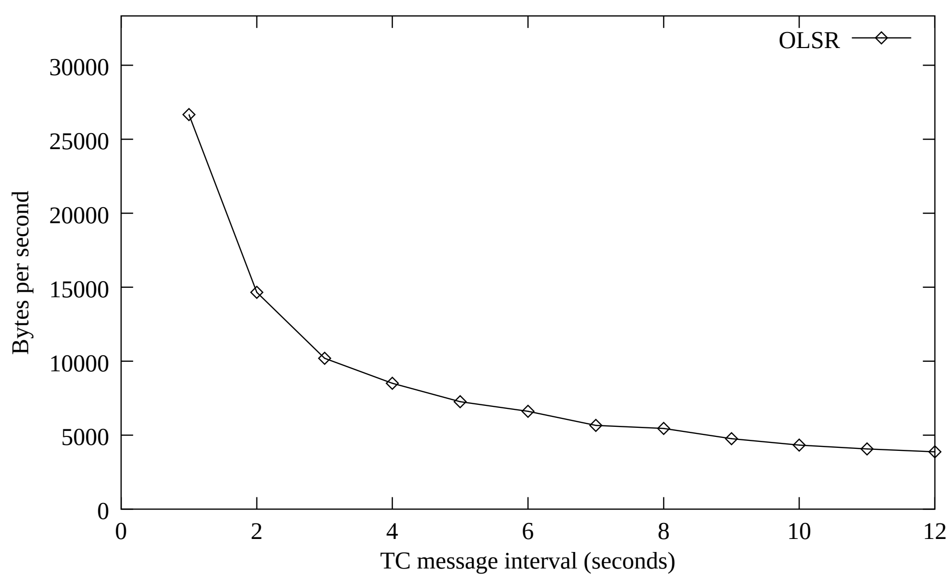
Graph 7.12: *Amount of control traffic with variable hello message interval.*



Graph 7.13: *Number of sent and received packets with variable TC message interval.*



Graph 7.14: *Average packet delay with variable TC message interval.*



Graph 7.15: *Amount of control traffic with variable TC message interval.*

	Received		Noroute Drops	
Hysteresis	1/1	2/3	1/1	2/3
Packets per second mean	101.5	97.4	53.9	52.8
Standard deviation	9.87	7.33	12.18	11.29
Coefficient of variation	9.73%	7.53%	22.58%	21.37%

Table 7.7.: *Descriptive measures for scenarios with and without the simple 2/3 link hysteresis.*

hello message intervals must be received, we might be able to avoid using links that are only sporadicly available. If a single hello message arrives, we simply ignore it.

The simulations done here were performed with a stream duration of 50 seconds.

Results

The measures of tendency and dispersion for the number of packets per second that are received or dropped due to route unavailability are shown in table 7.7. 4.2% fewer packets get through the network with the 2/3 hysteresis and 2% fewer packets are dropped due to route unavailability.

Analysis

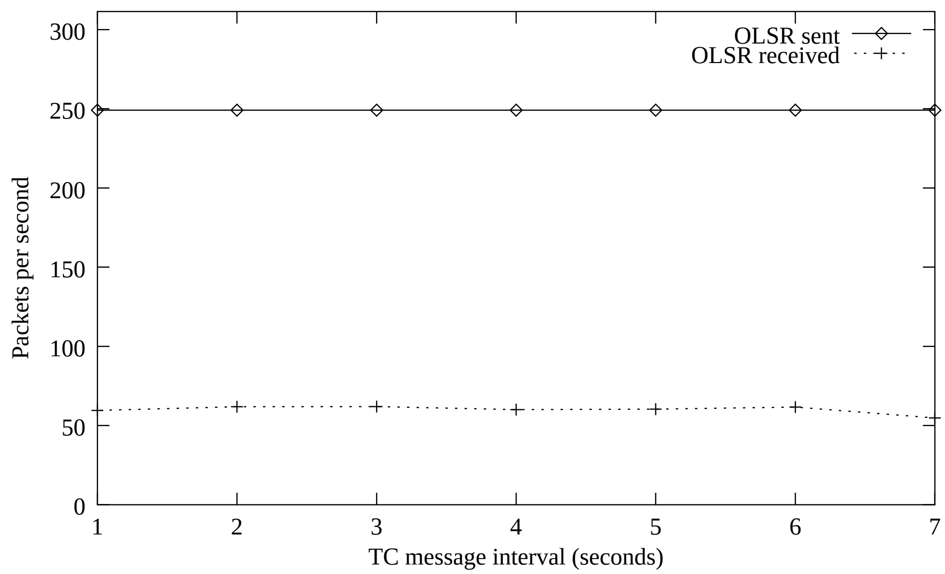
The simple link hysteresis tested here does not seem to make much of a different for the throughput in the network. However, note that the coefficient of variation for received packets is a little lower with the hysteresis than without. This means that the networks using the link hysteresis are a little more stable than networks using the plain OLSR protocol, even though a little fewer packets get through the network.

7.6. Summary

These tests have shown that the use of enforced jitter on the transmission of control packets in OLSR is of very high importance to the performance and stability of the network. The effects of piggybacking control messages with each other are not significant, although the network tends to be a little more stable when using piggybacking.

There is little effect of changing the control message intervals. Even with a highly dynamic topology, lowering the TC message interval does give a readable effect.

The simple link hysteresis of requiring 2 out of 3 hello messages before qualify a link as usable does not give a significant performance improvement.



Graph 7.16: *Number of sent and received packets with variable TC message interval and high mobility.*

8. Comparison of OLSR and AODV

In this chapter, we will present the results of the simulations comparing the performance of OLSR with that of AODV. First, we describe the default test configuration of the simulations. Next, we present the thesis that is to be testes and the assumptions we have made about the results. Then, we describe the test with varied mobility, density, and traffic, respectively. The chapter is concluded by a summary that summarized the results of the analyses.

8.1. Test Configuration

The thesis tested in this chapter, is the one stated in chapter 2 which concern the performance of OLSR compared to that of AODV. To examine this thesis, a test set of at least 30 scenarios for each set of scenario parameters were generated and run in NS2. The scenarios were generated by the scenario generator described in chapter 4 from the same default parameters stated in table 8.1, unless otherwise stated. The measures used in the result sections are selected from those stated in section 5.3. For each test set we state the results used for analyzing the particular set. For explanatory reasons the results are shown as numerals, graphs, and figures depending on the context. The complete set of the results in numerals, is included in appendix B.

Number of nodes	50 nodes
Field size	1000 x 1000 meters
Simulation Time	250 seconds
Node Speed	1 to 5 meters/second
Node Rest Time	0 to 5 seconds
Streams	25 streams
Packet Size	64 bytes
Packet Interval	0.10 seconds
Stream Duration	10 seconds

Table 8.1.: *Default parameters used in the simulations.*

8.2. Thesis and Assumptions

To our knowledge, there has been no comprehensive, simulation based comparison of OLSR and other Manet routing protocols. [Qay00] performs various comparisons of OLSR with the DSR protocol, but he uses a custom build simulator with several issues as described in section 1.4. We will compare it to AODV because this is the Manet protocol that have consistently shown the best performance in related works (for elaboration, see section 1.4). In section 3.3 we have described our assumptions on how we expect OLSR and AODV to perform in different types of scenarios.

We have performed a number of simulations to show how well OLSR performs when compared to AODV. We have varied the mobility in order to test how well the protocols perform when the topology changes frequently. We have varied the density of the network to see how well each protocol performs in large and dense network, and small and sparse networks. We have varied the traffic to see how the protocols perform under sporadic traffic and under more static traffic patterns. In other words, our work aims at uncovering when OLSR and AODV, respectively, excel.

8.3. Performance with Variable Mobility

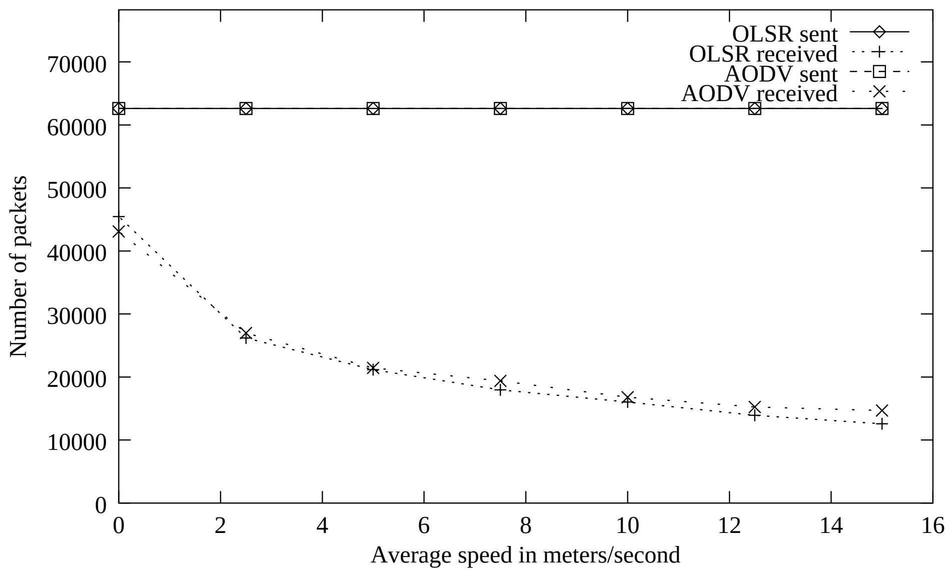
In order to determine how OLSR and AODV perform under variable mobility, 7 test sets with node mobility from 0 to 15 m/s were performed. 0 m/s is no mobility and 15 m/s corresponds to a slow moving car (54 km/h). Higher mobility is impractical for this particular type of medium (IEEE 802.11). With a speed of 15 m/s, a node moving through the radio range of another stationary node would only have radio contact for less than half a minute.

Results

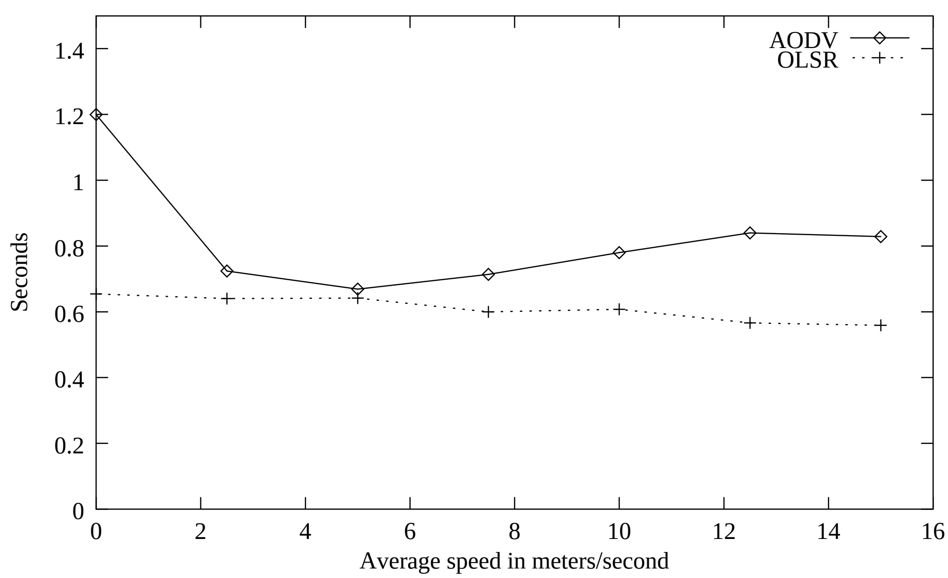
The number of sent and received packets for the simulation is shown in graph 8.1. The throughput for each protocol is nearly equal in both scenarios with little or no mobility and with high mobility. With little mobility, OLSR seems to perform slightly better than AODV, with 5% more packets received at no mobility. In networks with high mobility, AODV seems to have a little advantage over OLSR with 21% more packets received at an average speed of 15 meters per second.

The average packet delays are plotted in graph 8.2. The graph shows that the average packet delay always is higher in networks using AODV than in networks using OLSR. However, in networks with medium speed (2 to 8 meters/second), the average packet delay in AODV networks is not significantly higher than that of OLSR networks.

The amount of bandwidth used for control traffic for each protocol are plotted in graph 8.3. The graph shows that the control traffic of OLSR is the same at all levels of mobility, while that of AODV increases with increasing mobility. At no mobility, the control traffic of AODV is significantly lower than with mobility, but it is still higher than that of OLSR.



Graph 8.1: *Number of sent and received packets with variable mobility.*



Graph 8.2: *Packet delay with variable mobility.*

Analysis

AODV manages to get more packets through the network than OLSR when there is a high mobility (links break and are created more frequently). This is expectable – the AODV protocol reacts faster than OLSR to changes in network topology. In networks using OLSR, the new or newly broken links will have to be detected (two-way negotiation), MPRs selected, and new topology information diffused into the network before the routing can utilize the changes to the topology. AODV will only have to detect that the link has broken before performing a local route repair. In case AODV chooses to send a *Route Error* packet back to the source node, the route will have to be requested, which will take substantially more time than performing a local route repair.

The average packet delay is higher with AODV than with OLSR. This can be explained by the sub-optimal routes that AODV provides (as described in section 3.3). Another possible explanation is the that the first packets sent in a stream are delayed while AODV requests the route and waits for the route reply. It is interesting to see that the packet delay with AODV is lower with a moderate mobility than with no mobility. It may be explained by that the extra packets that does at no mobility (according to graph 8.1 are those that have to take the longest routes through the network.

The big difference in control traffic with AODV between no and some mobility can be explained by the introduction of link breakage and creation with mobility. With no mobility, links are static and a route will only have to be requested once, while with some mobility, a route may have to be repaired or re-requested during the session, hence the extra control traffic. OLSR's control traffic is constant as it is not affected by the creation or breakage of links.

8.4. Performance with Variable Density

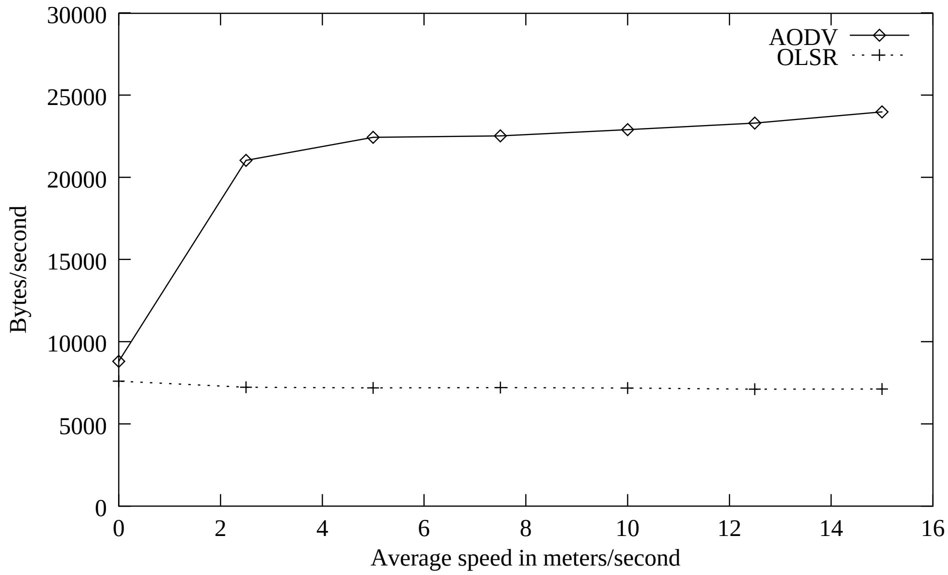
This test was designed to show how each protocol operates in networks with different node density. In a simulation area of 1000 by 1000 meters, test sets with the following number of nodes were performed: 10, 20, 50, 75, 100, and 125.

8.4.1. Variable Amount of Traffic

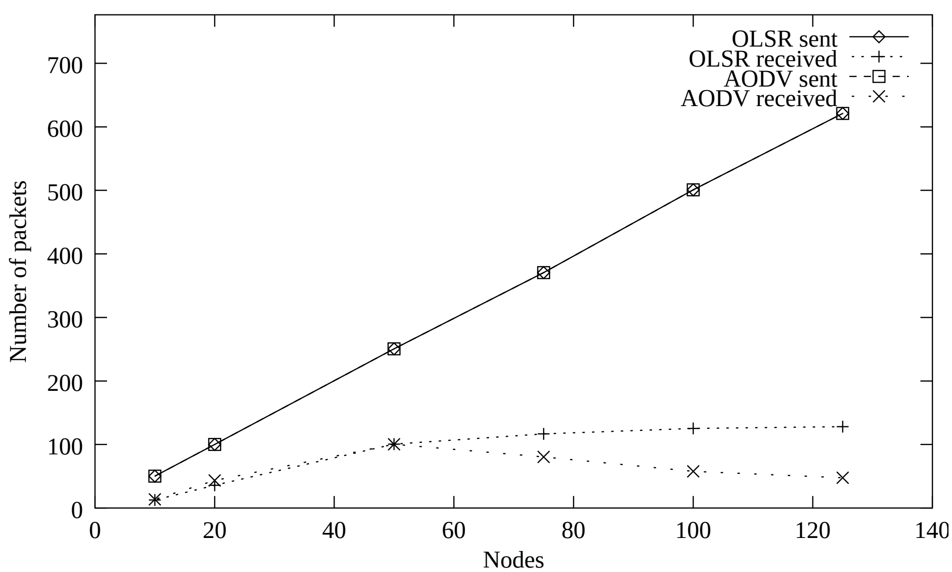
The number of streams in this test is dependent on the number of nodes, namely 50 streams per 100 nodes. We have done this, because in most real networks, we assume that each node would, on average, generate the same amount of traffic. Hence, the total amount of traffic in the network would increase linearly with the number of nodes.

Results

Graph 8.4 shows the number of sent and received packets, with variable density. The graph shows that the number of sent packets climbs linearly with the number of nodes, as we have defined. The number of received packet with AODV and OLSR is almost equal in networks



Graph 8.3: *Amount of control traffic with variable mobility.*



Graph 8.4: *Number of sent and received packets with variable density.*

with less than 50 nodes. In networks with more than 50 nodes, AODV throughput drops while OLSR throughput remains nearly the same.

Graph 8.5 shows the packet delay with each protocol and with variable density. The graph shows that the average packet delay increases with the number of nodes in the network for both protocols. With OLSR, the average packet delay climbs faster than that of AODV.

Analysis

The main reason that the throughput with AODV drops significantly is because of the control traffic that the protocol generates. The control traffic is plotted in graph 8.6. OLSR control traffic increase with the number of nodes in the network, which is expectable since each node generates extra hello and TC message. At 125 nodes, OLSR control traffic is around 26000 bytes per second. AODV's control traffic increases much more than that of OLSR. At 100 and 125 nodes, the AODV control traffic is 5 times that of OLSR. The amount of control traffic that AODV generates is mainly determined by the traffic in the network, and since we have increased the amount of traffic linearly with the number of nodes, we would expect the control traffic to increase.

The average packet delay with OLSR is lower than with AODV in networks with low density. The can be explained by that AODV queues packets while requesting routes and may choose inoptimal routes as explained in section 3.3. In high density networks, networks using OLSR has a higher packet delay than networks using AODV. This can be explained by AODV's lower throughput in these networks. The packets that does get through the network are most likely the packet following shorter routes, while in OLSR networks, the throughput is higher and hence more packets travel longer routes and hence the average packet delay is higher.

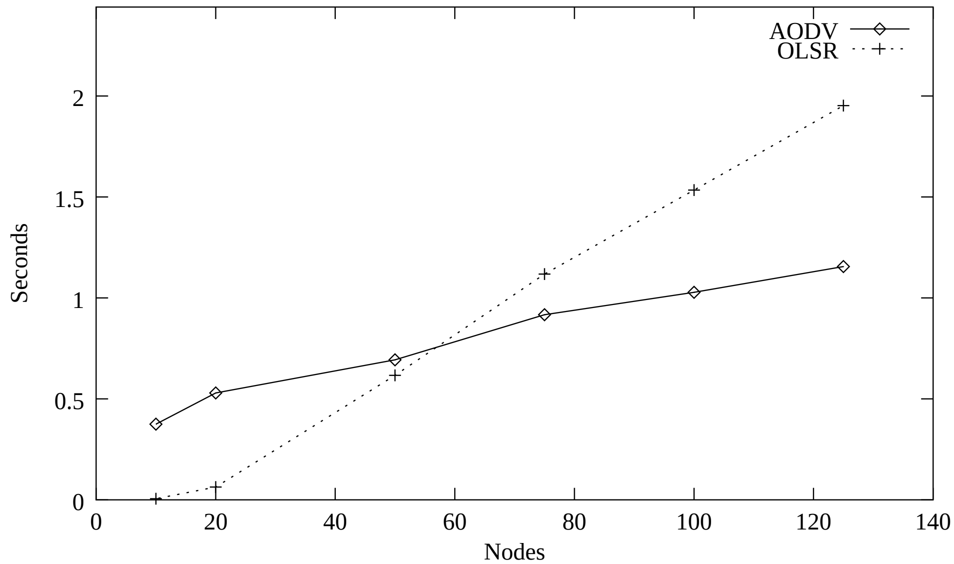
8.4.2. Constant Amount of Traffic

In order to test whether the difference in throughput is caused only by the extra traffic in the network performed the simulations again, but this time with the same amount of traffic in all scenarios (25 streams).

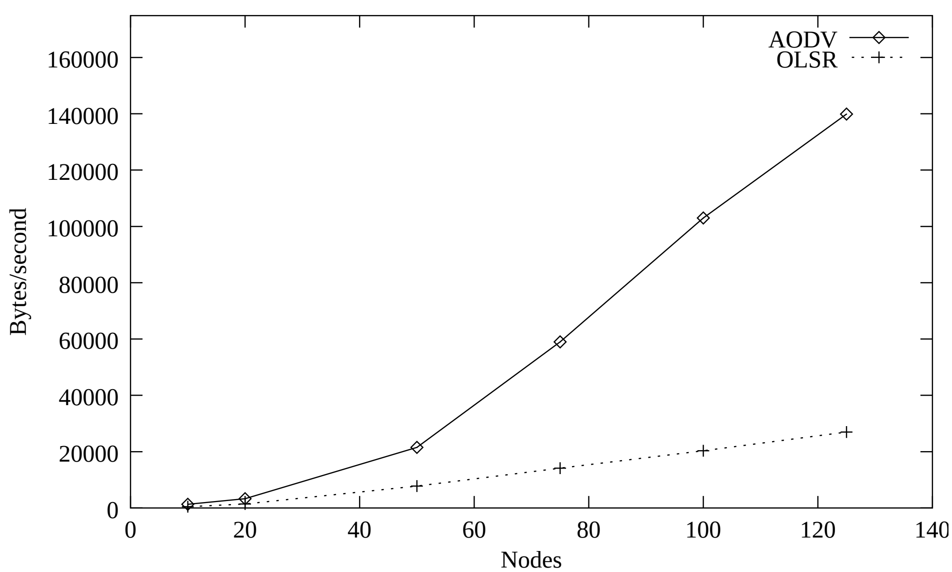
Results

Graph 8.7 shows the throughput with variable density and constant amount of traffic. The graph shows that the protocols compare up to about 50 nodes. In networks with more than 50 nodes OLSR is able to get more packets through the network than AODV. At 100 nodes, AODV gets 26% less packets through the network, and at 125 nodes 34% less.

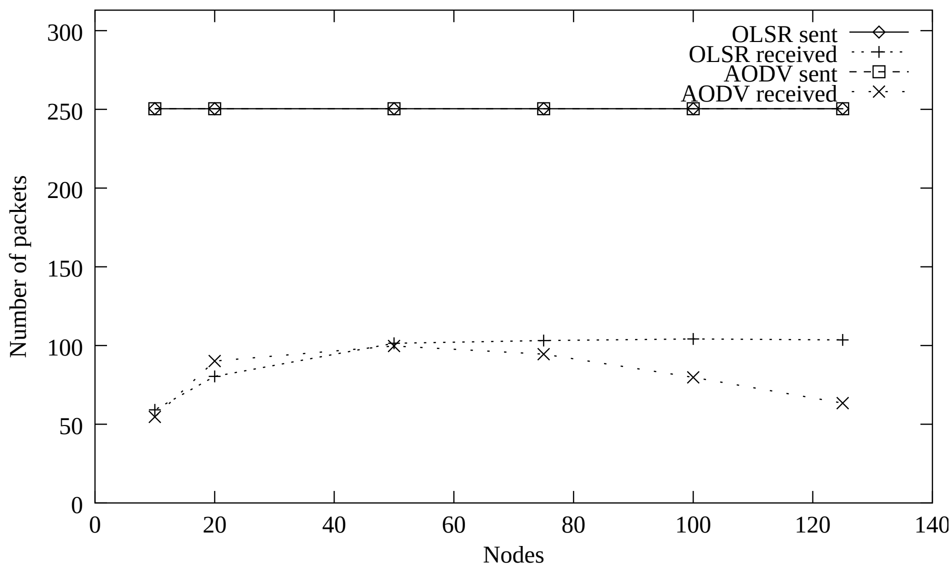
Graph 8.8 shows that the average packet delay for AODV and OLSR. The graph shows that when the protocols compare in throughput, AODV has a higher packet delay than OLSR. In high density network, where OLSR has a higher throughput, AODV has a lower average packet delay than OLSR.



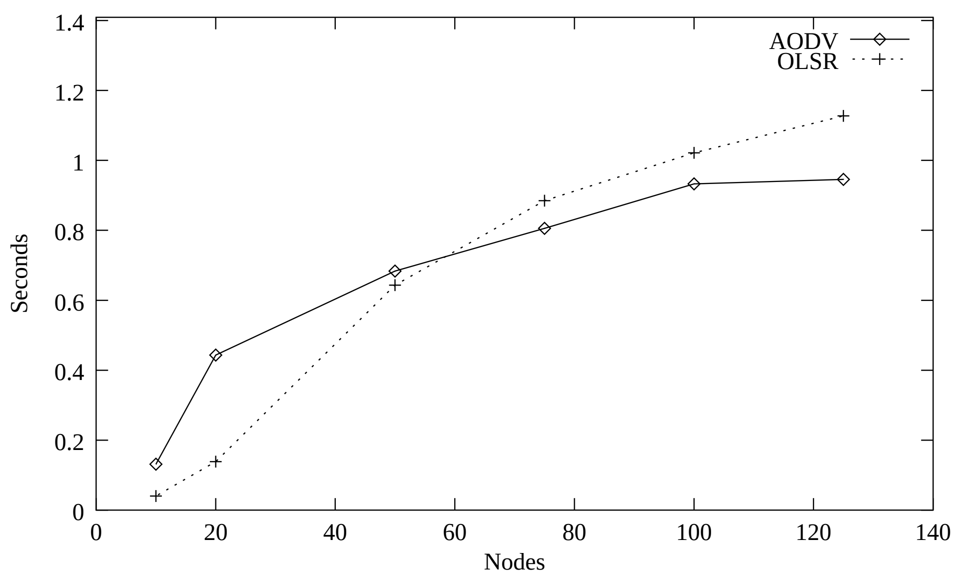
Graph 8.5: *Packet delay with variable density.*



Graph 8.6: *Amount of control traffic with a variable number of nodes*



Graph 8.7: *Number of sent and received packets with variable density and constant amount of traffic.*



Graph 8.8: *Average packet delay with variable density and constant amount of traffic.*

Graph 8.9 shows the amount of control traffic transmitted with each protocol. The control traffic in networks using AODV is approximately 100 Kb/s. That is 2.5 times higher than in networks using OLSR which is around 40 Kbps.

Analysis

The throughput of AODV drops in high density networks, even in this test where the amount of traffic is constant. This, we think, is mainly caused by the extra control overhead of AODV. When there are more nodes, the flooding of route request packets consumes much more bandwidth. This is also true for OLSR, but the amount control traffic increases at a much lower rate than that of AODV because of the use of MPRs in OLSR.

The cause of AODV's lower packet delay is, most likely, the fact that the throughput is also lower and that the packets that do get through the network are packets which have only a short route to travel. This is consistent with the results when there is variable traffic (graph 8.5), but the difference here is smaller because there is less traffic.

8.5. Performance with Various Types of Traffic

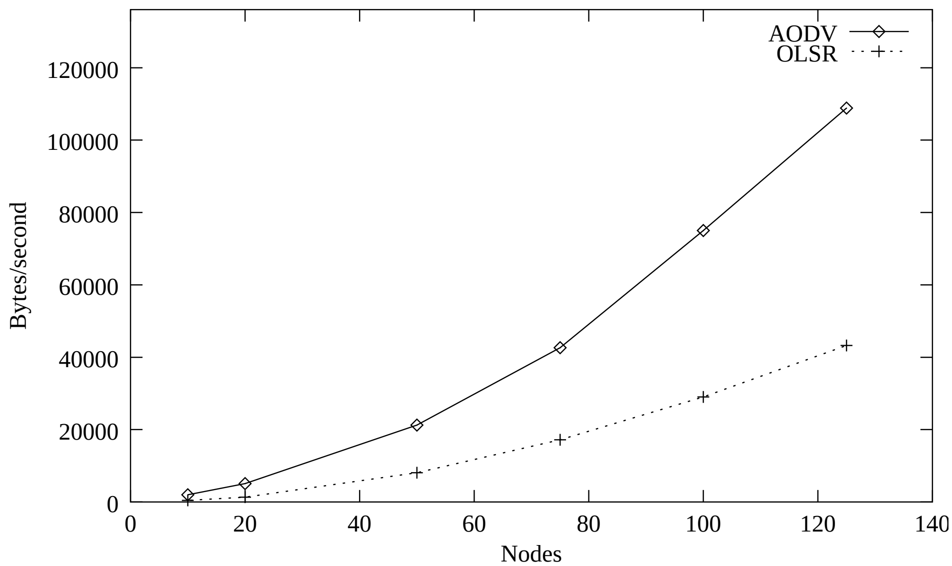
In order to test how OLSR and AODV perform under various types of traffic, we have run simulations with sporadic and static streaming traffic (section 8.5.1). We have also run traffic with TCP sessions in order to test how well the protocols handle bulk data transfers (section 8.5.2). In this test we have measured both the common performance parameters such as throughput and delay, but also the transfer time, that is the time it takes to perform an entire TCP transfer.

8.5.1. Variable Duration

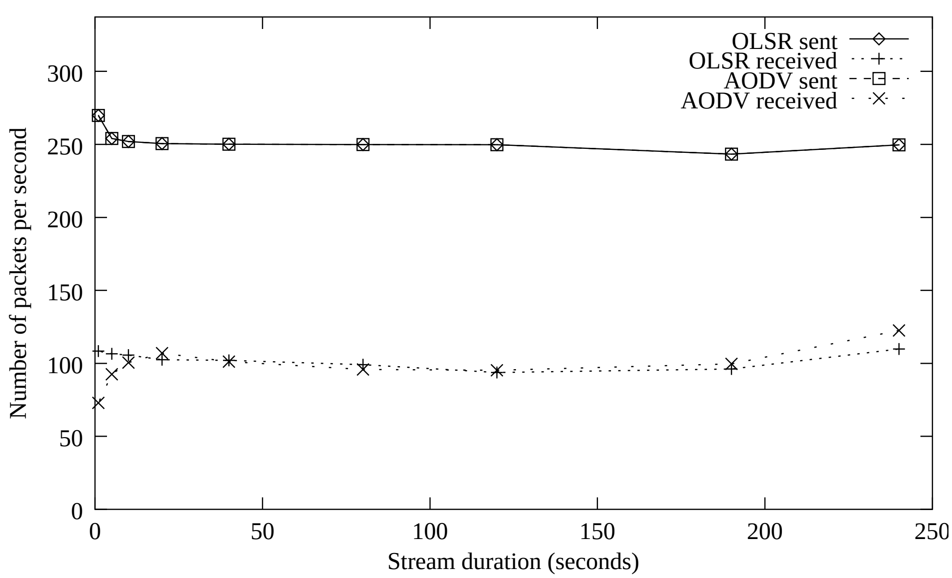
We anticipate that OLSR performs better with sporadic traffic and AODV better with static traffic. Therefore, we created scenarios with variable duration of the stream in the network. Test sets with a variable stream duration with values of 10, 20, 40, 80, 120 and 240 seconds were performed. The average number of simultaneous streams in these tests is 25. That means that with a stream duration t, there will be a total of $25\frac{t}{250}$ streaming sessions (250 seconds is simulated).

Results

Graph 8.10 shows the throughput as the number of packets received per second in the simulated networks. The throughput using OLSR and AODV is equal, except in the boundary cases. At very low duration, that is, when the number of streaming sessions is high, the AODV throughput drops while the OLSR throughput remains the same. At high duration, that is, when the is only a few streaming session in the entire simulation, the AODV throughput increases a little more than the OLSR throughput.



Graph 8.9: *Amount of control traffic with variable density and constant amount of traffic.*



Graph 8.10: *Number of sent and received packets with variable stream duration*

The average packet delay is shown in graph 8.11. The graph shows that the average packet delay of AODV is a little higher than that of OLSR (10-15%), except with very short duration where the packet delay is equal.

The amount of control traffic sent by each protocol is shown in graph 8.12. At low duration, the control traffic of AODV increases significantly while the control traffic of OLSR remain constant. Also, note that the control message overhead when using AODV is at least twice the overhead of OLSR control traffic, in all scenarios.

Analysis

At low duration, when the number of streaming sessions is high, the performance of AODV drops significantly. This may be explained by that AODV's activity depends on the traffic. When the number of sessions increases, AODV must request routes much more often and hence overloads the network with control traffic.

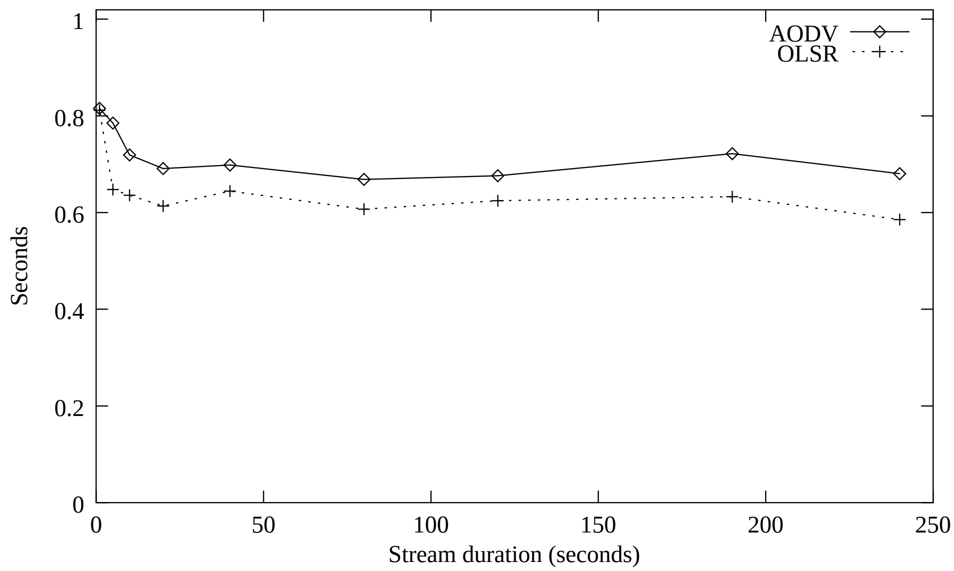
In cases where OLSR and AODV do not exhibit the same throughput it is hard to compare the average packet delay, because the lower throughput may be caused mainly by lost packets in long paths (many hops), while short paths (few hops) may give the same throughput. If the average packet distance, that is, the number of hops a packet uses to travel from its source to its destination, is lower, the average packet delay will also be lower. But, when the throughput is equal for AODV and OLSR, that is between a duration of approximately 20 and 150, we can assume that the average number of hops a packet uses is the same for both protocols. It is interesting that the average packet delay is a little higher with AODV than with OLSR. We anticipate that this is because OLSR uses optimal routes, provided that enough topology information is available, while AODV uses the route over which the *Route Request* first reaches its destination, which may be suboptimal.

8.5.2. Bulk Transfer Test

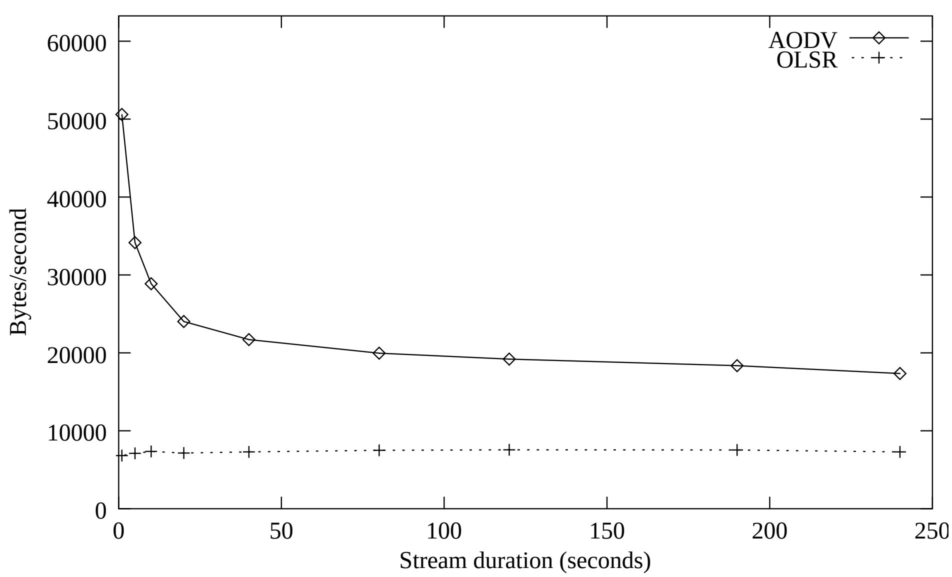
To test each of the protocols under various loads and using bulk transfers over TCP instead of streaming traffic, test sets with 6, 8, 10, 12, 14, 16, and 18 TCP transfers per second were performed, each transferring 16 Kb of data. The packet size used in each TCP session was set to 1024 bytes. Therefore, the number of queued packets per second is on average $16 \times t$, where t is the number of transfers per second. The traffic in these scenarios is very sporadic because the transfers are very short. Hence, the traffic is similar to that of streaming with low duration.

Results

The throughput as number of packets sent and received is shown in graph 8.13. The graph also shows the number of queued packets in the TCP flows (16 per transfer in this scenario). The number of sent packets is the number of packets that leave the nodes. Hence, packets that are not send due to TCP congestion handling are not included in the graph.



Graph 8.11: *Average packet delay with variable stream duration*



Graph 8.12: *Amount of control traffic with variable duration*

The control traffic overhead is shown in graph 8.14. This graph shows that the control message overhead of AODV is 4 to 6 times that of OLSR. The AODV maximum overhead reaches 50 Kb/s. The OLSR control message overhead is around 10 Kb/s.

Analysis

Graph 8.13 shows that the networks using OLSR both manage to send more packets and get more packets through than AODV. The reason that the number of sent packets does not follow the number of queued packet is retransmission and congestion handling in TCP. The number of received packets is higher that the number of queued packet, in some cases, because of retransmissions that result in duplicate reception at the destination node. The number of received packets as a fraction of the number of queued packets is shown in graph 8.15. The graph shows that OLSR manages to get significantly more packets through than the AODV protocol. The major reason for this is the amount of control traffic that AODV sends on the network.

The reason that the OLSR bandwidth drops and higher loads is that the graph only includes control messages actually transmitted over the medium, and at higher loads, the interface queues get more congested and more control traffic is dropped.

Generally, the TCP transport layer protocol performs badly over wireless network because of the relatively high drop rates due to collisions and interference. TCP assumes that the reason for drops is congestion and therefore lowers the data rate, hoping to get more data through by avoiding congestion. However, with a fixed probability for packet drop of for example 20%, 80 Kb/s will get through if the source sends 100 Kb/s, and only 40 Kb/s if the rate is lowered to 50 Kb/s.

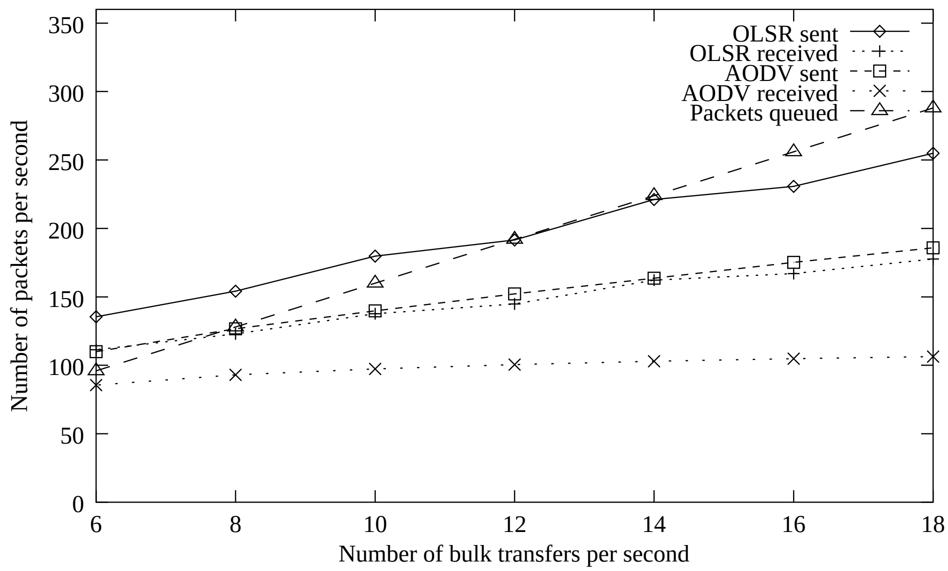
TCP's adaptive behavior make this test a bad measure of how the protocols perform under various load, while it is still a good measure of how it performs under this particular type of traffic.

8.5.3. Transfer Time

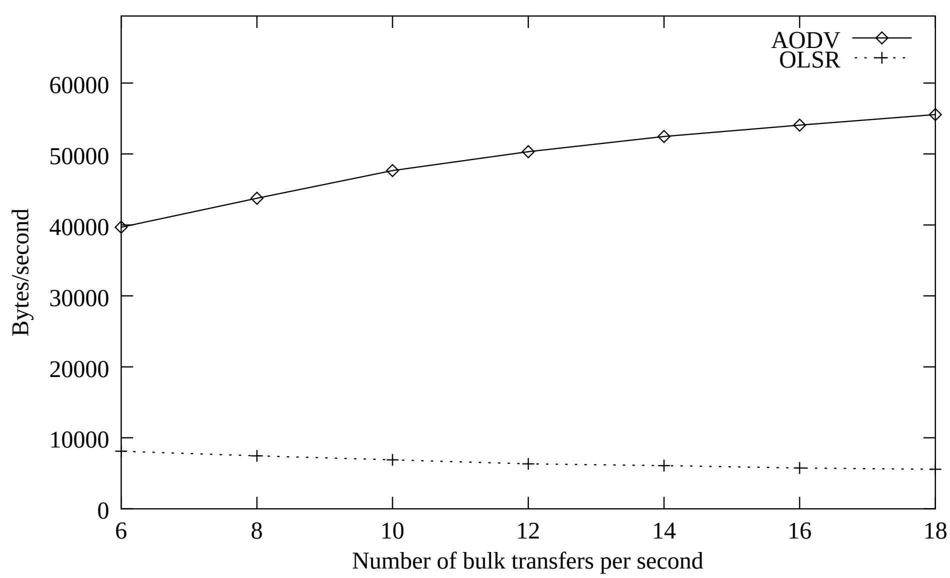
In order to test the actual transfer time used to perform a bulk TCP transfer, we have run simulations of networks with TCP transfers of 16 kilobytes of data transferred between random nodes. We measured the transfer time as the time from the initiation of the TCP transfer, when the first packet is sent from the application layer, and until the final packet is received at the destination nodes application layer. In each scenario, 100 bulk transfers were performed within the 250 simulated seconds.

Results

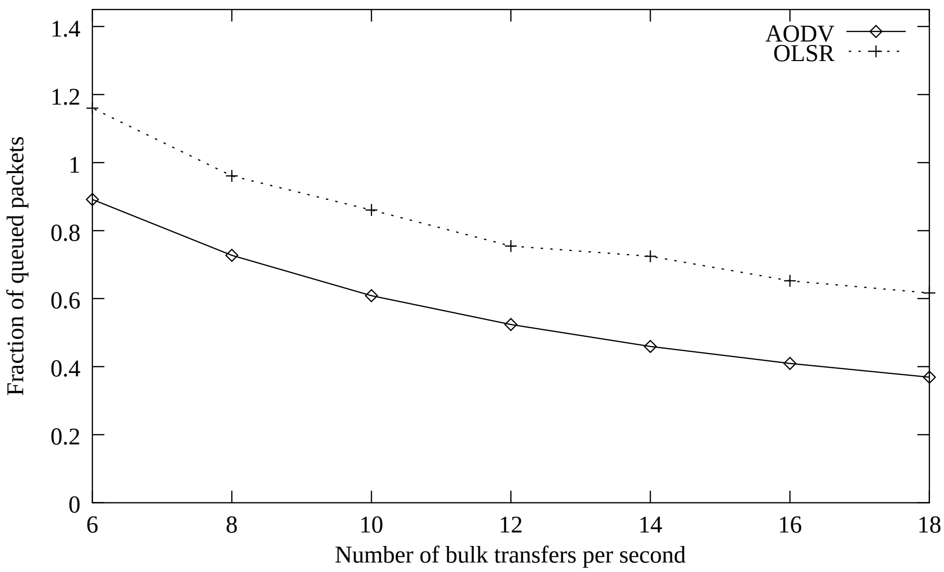
The measures of tendency and dispersion for the bulk TCP transfer time are shown in table 8.2. The average time for a TCP transfer is approximately 10% higher with AODV than with OLSR.



Graph 8.13: *Number of sent and received packets with a variable number of bulk transfers per second*



Graph 8.14: *Amount of control traffic with variable load*



Graph 8.15: *Number of received packets as a fraction of the number of queued packets with variable number of transfers per second.*

	OLSR	AODV
Average bulk transfer time	3.25 seconds	3.56 seconds
Standard deviation	0.66	1.16
Variance coefficient	20.44%	32.67%

Table 8.2.: *Descriptive measures for bulk TCP transfer times.*

Analysis

The higher bulk transfer time is most likely caused by AODV queuing packets while requesting routes and possibly because it uses suboptimal routes. The variance coefficient is a little higher for AODV than for OLSR. This means that the networks using OLSR are also more stable than AODV, that is, with AODV there is a higher risk of a bad case.

8.6. Performance with Variable Load

In order to test how each protocol performs under variable load we have run a series of simulations with variable number of streams with UDP traffic. We have simulated networks with 0, 5, 10, 15, 20, 25, 50, and 100 simultaneous streams. We have used 512 byte packets in this test. At 5 streams, the nodes try to send 25 Kb/s. When the number of stream is over 100, the load is quite extreme. At 100 streams the nodes tries to transmit 512 Kb/s, that is, twice the expected bandwidth in a local region. Because the network spans more than the radio range of a single node, a higher total throughput than the 2 Mbit total can be expected, but it should be taken into consideration that packets transmitted in this test take multiple hops to reach their destination and, hence, will use the medium multiple times.

Results

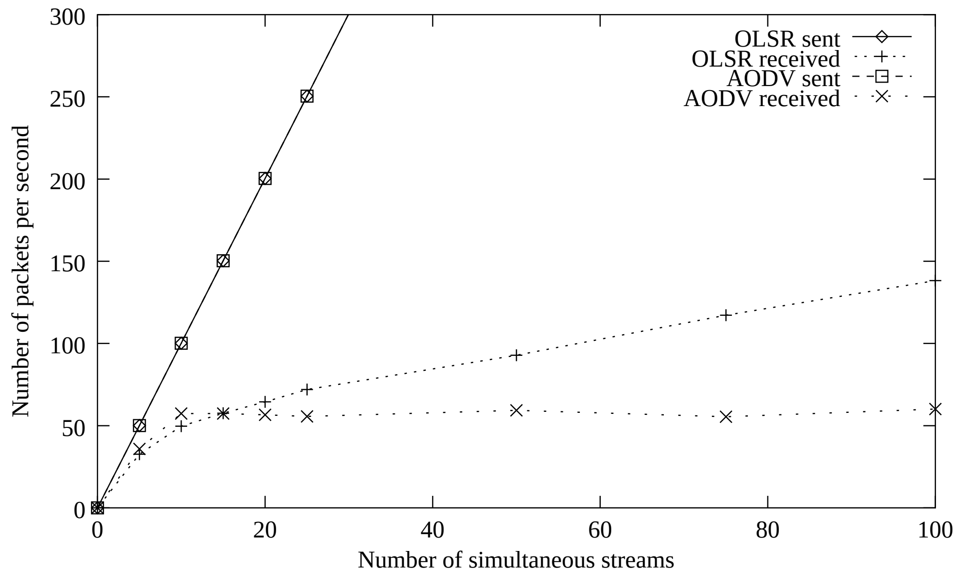
The number of packets sent and received for networks with less than 100 streams are shown in graph 8.16. The line plotting the number of sent packets has been cut to show the difference in number of received packets for the protocols. The number of sent packets increases linearly with the number of streams. The graph shows that both protocols only get a small fraction of the number of sent packets through the network. At more than 25 streams, the two protocols differ more and more in number of received packets. At 100 streams, the number of received packets with OLSR is 119% higher than with AODV.

The average packet delay is plotted in graph 8.17. The graph shows that the average packet delay when using AODV is consistently slightly higher than with OLSR. At medium load (25 streams), the delay with AODV is around 25% higher than with OLSR. At higher load (50-100 streams), the difference is smaller, namely only 10% higher than with OLSR.

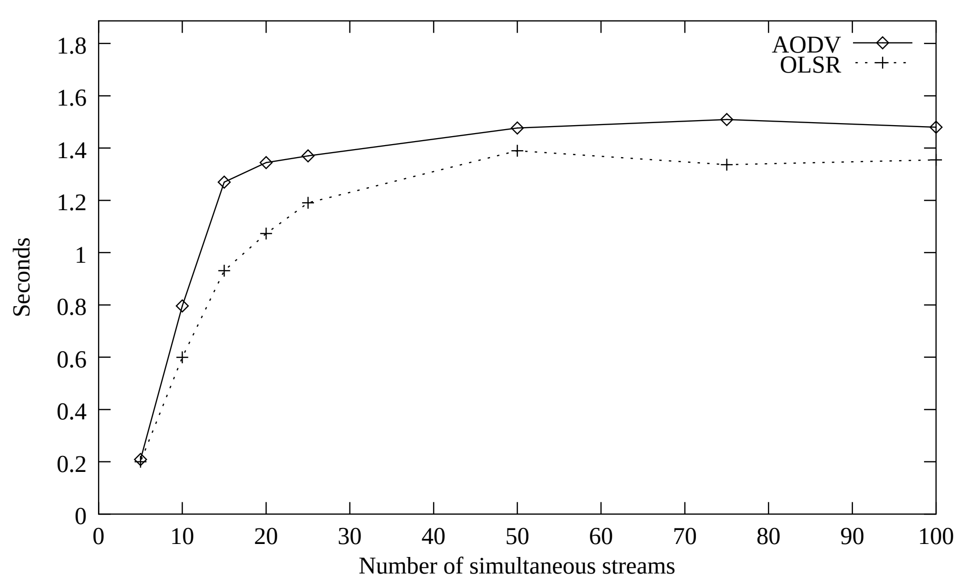
The amount of control traffic with each protocol is shown in graph 8.18. The graph shows that the control traffic with AODV is significantly higher than with OLSR in most situations. Only in networks with very little traffic (5 or less simultaneous streams), the control traffic with AODV is lower.

Analysis

The difference in throughput in this test is very significant in networks with high loads. OLSR manages to get more than twice the packets through when compared to AODV, in networks with 100 simultaneous streams.



Graph 8.16: *Number of sent and received packets with variable number of simultaneous streams.*



Graph 8.17: *Average packet delay with variable number of simultaneous streams.*

	Big group	Small group
Number of nodes	30-50 nodes	3-7 nodes
Node speed	0-1 meters/sec	0-10 meters/sec
Stream to other group	5-15 stream	15-30 stream
Packet size	64 bytes	64 bytes
Bulk transfers to other group	15-25 transfers	80-120 transfers
Bulk transfer amount	8-24 kilobytes	8-24 kilobytes
Simulated time	250 seconds	
Field size	1000×1000	

Table 8.3.: *Simulation parameters for the cluster test.*

The packet delay with AODV is higher than with OLSR. This has already been discussed in previous sections and the same explanations apply here.

The control traffic with AODV increases with the number of streams, which is expectable because there are more active routes in the network, and hence AODV nodes have to request and maintain more routes. The amount of OLSR control traffic in the networks drops with increasing number of streams. This is caused by the medium getting saturated and hence more OLSR packets are dropped in the interface queues (the control traffic is measured as the actual number of bytes transmitted on the medium, not the amount generated by the protocol implementations, described in section 5.3). At low traffic rate, the overhead with OLSR is high because it make the same effort to detect neighbors and diffuse topology information as with traffic. The overhead with AODV is low here because there is no traffic in the network and no routes are requested. The control traffic that AODV nodes transmit when there is no data traffic is the hello messages used to detect broken links.

8.7. Clusters

All of the networks we have simulated so far have been homogeneous in terms of node placement, mobility and traffic. In the real world, it is likely that the traffic in the network will be focused on a subset of the nodes, providing special services. That is, there will be a small group of nodes communicating with a larger group of nodes. This model applies to many realistic scenarios such as those simulated by [JLH+99]. An example of such a network is an office environment where people tend to mostly use the network to access file or print servers, or gateways to other networks (for example the Internet).

In order to test the two protocols' performance in such networks, we have simulated networks with one large group (30 to 50 nodes) and one small group (3 to 7 nodes). The actual parameters are shown in table 8.3.

	OLSR	AODV
Received packets per second (mean)	139.42	134.46
Received packets per second, standard deviation	24.39	17.63
Received packets per second, coefficient of variation	17.50%	13.11%
Packet delay, mean	0.39	0.77
Packet delay, Standard deviation	0.15	0.20
Packet delay, Coefficient of variation	38.78%	26.35%
Control traffic, mean	3199 bytes	10856 bytes

Table 8.4.: *Descriptive measures for each protocol in the cluster test.*

Results

The descriptive measures for the results of these simulations are shown in table 8.4. The numbers show that the protocols achieve similar throughput, but that the average packet delay with OLSR is near the half of that with AODV. The control traffic produced in AODV networks is 2.5 times that in OLSR networks.

Analysis

In the average case, the two protocols have the almost the same throughput. The packet delay with OLSR is, however, significantly lower than in networks using AODV. This is caused by the already discussed reasons of packet queuing during route requests and route suboptimality. The coefficient of variance is lower for the number of received packets and the packet delay is lower with AODV than with OLSR. This means that networks using AODV tend to be slightly more stable than networks using OLSR.

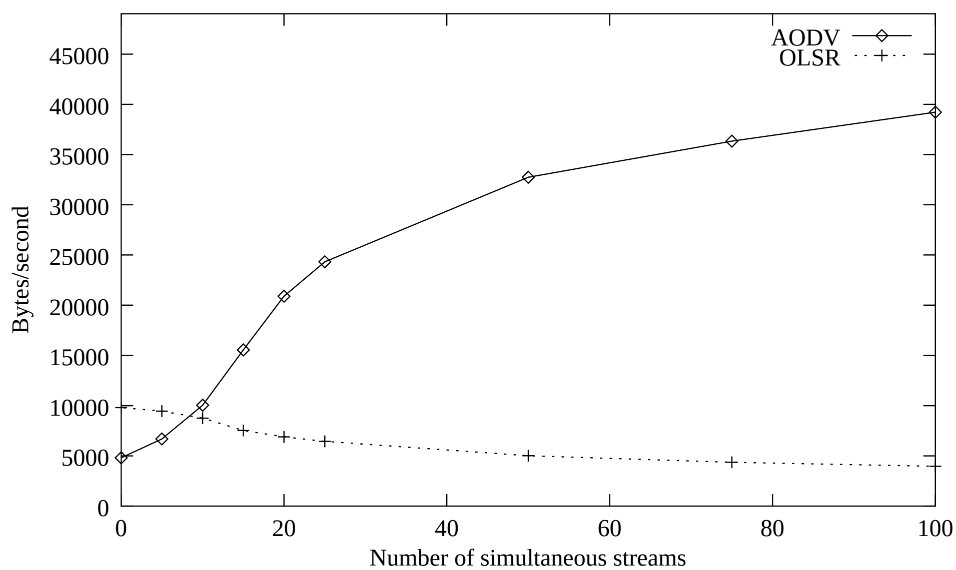
The control overhead in AODV networks is substantially higher in than in OLSR networks, but still not critically high because amount of traffic in the simulation networks is relatively low.

8.8. Summary

These test have shown that the two protocols, Optimized Link State Routing protocol and the Ad-Hoc On-Demand Distance Vector protocol perform very equal in terms of throughput, except in boundary cases. In a highly mobile network with frequent topology changes AODV has a slight advantage over OLSR. In networks with little or no mobility, that is, with a static topology, OLSR has a slight advantage over AODV. In high density networks, OLSR has a big advantage over AODV because AODV loads the network with control traffic. In low and medium density network, the protocol compare in throughput. Under very sporadic and short lived traffic sessions, streaming or bulk, OLSR has a big advantage over AODV because it has the routes available beforehand. With very static streaming traffic, AODV has a slight advantage over OLSR.

In almost all types of scenarios, OLSR gives a slightly lower packet delay than AODV. The time to transfer a 16Kb data load using TCP is slightly higher with AODV than with OLSR.

In almost all cases, the control message overhead of AODV is substantially higher than that of OLSR. Especially, the AODV overhead increases an order of magnitude faster with parameters such as number of nodes and number of traffic sessions in the network.



Graph 8.18: *Amount of control traffic with variable number of simultaneous streams.*

9. Conclusion

In this project, and our major (hovedfag), we have performed an empirical study of MANET routing protocols and simulation methods. We have performed scenario based simulations to gain results about the performance of the MANET routing protocols. In this chapter, we will summarize the products of our work, the methods we have applied and the results that we have arrived at. Finally, we will mention possible future works.

Products

We have implemented the Optimized Link State Routing protocol for NS2. This is one of at least seven implementations of the protocol (although our implementation does not have the interfaces required to work in a real network).

We have designed and implemented a scenario generator which is able to generate completely random scenarios under the constraints of a given set of scenario parameters. The use of this scenario generator allows us to simulate a wide range of scenarios with identical parameters in order to get a general picture of the MANET routing protocols' performance in particular types of scenarios.

We have developed a framework for running simulations of wireless protocols. This framework consist of the simulator, NS2, the scenario generator and a set of utilities to set up simulations, gather results from the trace files, and calculate descriptive measures such as mean, deviation, and coefficient of variation, and perform the chi square test of independence. This framework allows the user to provide the scenario parameters and ask for a certain number of simulations to be run, and then, nearly automatically, the descriptive measures will be delivered. Without this framework, the execution of all the individual simulations in this project (more than 5000) would have been a tedious work.

In addition, we are co-authors of a paper to appear in the Fourth International Symposium on Wireless Personal Multimedia Communications, namely [BCCH01]. Our contribution to the paper is a description of the OLSR protocol and an analysis of the effects of enforcing jitter and using piggybacking. Furthermore, the paper includes a documentation of practical experiments with OLSR. The paper is included with this report.

Methods

We have used the scenario generator to generate an exhaustive set of simulations on various types of scenarios. A list of all the scenarios we have simulated, and included in the results chapters, is included in appendix A. We have performed simulations of at least 30 scenarios with each set of parameters. For example, when varying parameters such as density, we have simulated at least 30 scenarios with each number of nodes that we have selected for the test. These exhaustive simulations ensure that the results are representative for the particular type of scenario. If exhaustive simulations are not performed, the result risks being based on a particular lucky or unlucky configuration of nodes, movement and traffic.

We have tested how the protocols perform with TCP bulk transfers. This has not been done in any of the related simulations of Manet routing protocols (described in section 1.4). Testing the protocols' performance with bulk transfers using TCP is important because such traffic is very common in real networks. According to [TMW97], 90% of the traffic on the Internet is TCP, and hence bulk transfers. Although TCP has performance problems in wireless networks due to congestion handling mechanisms, it is likely that it will be used in the real world, for example, to transfer files, and to access gateways to the Internet.

Even though the universe modeled by the simulator is quite comprehensive and includes a complete networking stack and a model of the physical layer, the simulation model is still a simplification of the real world. The simulations do not include any external entities that may interfere with the radio communication in the real world, such as physical obstacles and radio interference from other devices. The actual throughput may be overestimated (or even underestimated) because of a too perfect or imperfect model and we have therefore not drawn conclusion about the individual performance of a protocol in a particular scenario, but only the relative performance.

Simulation Results

OLSR Performance

We have shown that the use of enforced jitter in OLSR on the transmission of control packets is of utmost importance to the performance and stability of the routing protocol. There is no direct cost of enforcing jitter. In general, we strongly recommend that implementations of the OLSR protocol implement enforced jitter on all transmitted control packets.

We have tested the effect of piggybacking control messages in OLSR and shown that the effect is minimal. It may make the network perform better, and slightly more stable. However, the gain is small. We recommend that piggybacking is included in implementations of the OLSR protocol, because of the possibility of improvement, and because it is without cost. Under any circumstance, the overhead will be slightly reduced because fewer packets is sent on the medium.

We have tested OLSR with variable hello and TC message intervals in order to see whether the performance could be improved by adjusting them. We have shown that nothing can be gained by lowering the intervals, and that only a degradation of performance can be achieved by increasing them.

We have tested the simple, conservative link hystereses of requiring 2 out of 3 hello packets to be received in order to qualify a link as asymmetric or symmetric. The test showed little improvement. We have not had the time to further investigate in other methods of handling poor link quality, in particular other hystereses such as requiring 3 out of 4 hello messages to be received.

Comparison of OLSR and AODV

We have tested the OLSR and AODV protocols in various types of scenarios in order to determine how well they perform in comparison to each other. The main result of the simulations is that the two protocols perform very similar in many types of scenarios. However, in some particular types of scenarios they differ in performance.

In a highly mobile network with frequent topology changes AODV has a slight advantage over OLSR protocol. In networks with little or no mobility, that is, with a static topology, OLSR has a slight advantage over AODV.

In high density networks, OLSR has a substantially higher throughput than AODV because AODV loads the network with control traffic. In low and medium density networks, the protocols compare in throughput. Under very sporadic and short lived traffic sessions, streaming or bulk, OLSR has a big advantage over AODV because it has the routes available beforehand. In networks with very static streaming traffic, AODV has a slight advantage over OLSR. When the traffic in a network is mostly bulk transfers (TCP traffic), the throughput when using OLSR is substantially higher than when using AODV.

In most types of scenarios, OLSR gives a slightly lower packet delay than the AODV protocol. The time to transfer a 16 Kb data load using TCP is slightly higher with AODV than with OLSR.

In most cases, the control message overhead of AODV is substantially higher than that of OLSR. Especially, the AODV overhead increases an order of magnitude faster with parameters such as number of nodes and number of traffic sessions in the network.

In environments where sporadic bulk transfer traffic is typical such as an office environment where people surf the web, transfer files or print on network printers, OLSR has a big advantage over AODV.

Our general conclusion is that the Optimized Link State Routing protocol performs just as good as AODV in a wide range of scenarios, but has important and substantial advantages in particular scenarios such as networks with highly sporadic traffic and high density networks. This is consistent with the claims in [JMQ$^+$01], [Qay00], and [JV00]. Only in networks with very static traffic, AODV performs better than OLSR. This is contrary to the conclusions in [JLH$^+$99] that say that proactive protocols generally perform worse than reactive ones, albeit OLSR is not included in the tests.

Generally, we find that OLSR is more applicable than AODV in the widest range of scenarios. It generally generates less control traffic, gets equal or higher throughput and has lower packet delays. Only in networks with extremely static traffic, for example, two nodes far away in the network streams traffic without interruption, AODV has a higher throughput, but still gets a longer packet delay.

Future Work

It would be a logical step to perform large scale tests of Manet routing protocols, including OLSR and AODV, in real networks in order to get quantitative results about the real life performance. This may reveal new features and problems with the protocols because of real world properties that are not simulated in NS2 or other simulators.

It would be interesting to further investigate in methods for handling the potentially low and differentiating link qualities in Manets. We anticipate that it will be possible to improve the performance of the protocols by avoiding the use of low quality links, either by requiring a certain level of quality in order to accept a link into the topology, or by taking some measure of link quality into account when calculating routes.

Both protocol draft allow the use of link layer notification. We anticipate that it can improve the performance of OLSR and AODV, but it would be interesting to investigate the improvement quantitatively and relatively between the protocols.

Vocabulary

This vocabulary states the terms, definitions, and abbreviations used in this report.

AODV: Ad hoc On-Demand Distance Vector (Routing Protocol).

AODV node: A node utilizing AODV.

Broadcast: To transmit packets to all nodes within radio range.

CBR Traffic: Stream traffic with a constant bit rate.

Control Overhead: The amount of bandwidth occupied by control traffic. This may be measured in number of packets or bytes.

Control Packet: Packet with control information for use in routing protocols.

Data Packet: Packet with application data.

Flooding: Technique for transmitting packets to all parts of the network, where every incoming packet is retransmitted.

Full Flooding: Flooding of a network where all nodes retransmit packets as long as the time to live (TTL) value is larger than 0. Usually accompanied by local duplicate retransmission to avoid transmitting packet until they time out.

MANET: Mobile Ad hoc NETwork – self organizing network connected by wireless links. See section 1.3

MPR: Multi Point Relay - a node which is selected to forward control packets on behalf of other nodes.

MPR Flooding: Flooding of a network, where only MPRs retransmit packets meant for flooding.

MPR selector set: The set of neighbors which has selected a node as MPR.

MPR set: The set of neighbors which a node has chosen as MPRs.

Neighbor: A node with direct radio contact to the node in question.

Neighborhood: The total set of neighbors (of a node).

Neighbor sensing: The act of discovering which nodes are in the neighborhood.

Node: A host or router in a MANET.

NS2: Network Simulator 2.

OLSR: Optimized Link State Routing (Protocol).

OLSR node: A node utilizing OLSR.

Packet Delay: The time from a packet is transmitted by an application and until it is received.

Performance: The combined evaluation of quantitative parameter such as throughput, packet delay and control overhead.

Proactive Routing: Routing method which maintain routing tables up-to-date for every node in a network at all times.

Reactive Routing: Routing method which find routes in a network, only when needed.

Scenario: A specific setup of nodes, a specification of how they move, and what traffic they generate.

Scenario parameters: The parameters used for characterizing the settings of a scenario. In particular the parameters feed to the scenario generator.

TC: Topology Control.

Test set: The set of scenarios generated from the same scenario parameters.

Throughput: The number of data packets that reach their destination. Also named the number of received packets.

Two-hop neighbor: A node reachable through a neighbor.

Two-hop neighborhood: The set of all one- and two-hop neighbors.

Bibliography

[ASW96] David R. Anderson, Dennis J. Sweeney, and Thomas A. Williams. *Statistics for Business and Economics*. West Publishing Company, 6 edition, 1996. ISBN 0-314-06378-1.

[BCCH01] Gerd Behrman, Thomas Clausen, Lars Christensen, and Gitte Hansen. The Optimized Link State Routing Protocol - Evaluation through experiments and Simulation. In *Proceeding of Wireless Personal Multimedia Communications*, September 2001. To appear in the Fourth International Symposium on Wireless Personal Multimedia Communications.

[BHJ+00] Elena Tørnæs Beck, Gitte Hansen, Peter Jensen, Søren Enemærke Jespersen, Klaus Torst Rasmussen, and Uffe Refsgaard. Design and Implementation of the Optimized Link State Routing Protocol. Technical report, Department of Computer Science, Aalborg University, Denmark, May 2000.

[BL96] Donald A. Berry and Bernard W. Lindgren. *Statistics: Theory and Methods*. Wadsworth Publishing Company, 2 edition, 1996. ISBN 0-534-50479-5.

[Blu01] Bluetooth SIG. *Specification of the Bluetooth System*, February 2001. Version 1.1.

[BMJ+98] 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. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom '98), Dallas, Texas, USA*, October 1998.

[CEH01] Lars Christensen, Morten Ernstsen, and Gitte Hansen. Preliminary Performance Evaluation of the Optimized Link-state Routing Protocol. Technical report, Department of Computer Science, Aalborg University, Denmark, 2001.

[Dij59] E. W. Dijkstra. A note on two problems in connection with graphs. Numerische Mathematik, 1959.

[ETS95] ETSI STC-RES10 Committee. *Radio Equipment and Systems: High Performance Radio local Area Network (HIPERLAN), Type 1*, December 1995. Functional Specification.

[HJR00] Gitte Hansen, Peter Jensen, and Klaus Torst Rasmussen. Performance Test of the Optimized Link State Routing Protocol. Technical report, Department of Computer Science, Aalborg University, Denmark, May 2000.

[IET] IETF. Mobile Ad-hoc Networks (manet) charter. Available at http://www.ietf.org/html.charters/manet-charter.html.

[Jai91] Raj Jain. *The Art of Computer Systems Performance Analysis - Techniques for Experimental Design, measurement, Simulation and Modeling.* John Wiley & Sons, Inc., 1991. ISBN 0-471-50336-3.

[JLH+99] Per Johansson, Tony Larsson, Nicklas Hedman, Bartosz Mielczarek, and Mikael Degermark. Scenario-based Performance Analysis of Routing Protocols for Mobile Ad-hoc Networks. Technical report, ACM, 1999. Mobicom 99, Seattle Washington USA.

[JMQ+01] Philippe Jacquet, Paul Muhlethaler, Amir Qayyum, Anis Laoiti, Laurent Viennot, and Thomas Clausen. Optimized Link State Routing Protocol, March 2001. Internet-Draft version 04. Available at http://www.ietf.org/internet-drafts/draft-ietf-manet-olsr-04.txt, work in progress.

[JT87] John Jubin and Janet D. Tornow. The DARPA Packet Radio Network Protocols. In *Proceedings of the IEEE*, volume 75, pages 21–32, January 1987.

[JV00] Philippe Jacquet and Laurent Viennot. Overhead in Mobile Ad-hoc Network Protocols. Technical report, INRIA, Rocquencourt, June 2000. Research report RR-3965.

[LAN99a] LAN/MAN Standards Committee of the IEEE Computer Society. *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999. ANSI/IEEE std 802.11.

[LAN99b] LAN/MAN Standards Committee of the IEEE Computer Society. *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - High-speed Physical Layer in the 5GHz Band*, 1999. ANSI/IEEE std 802.11a.

[LAN99c] LAN/MAN Standards Committee of the IEEE Computer Society. *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Higher-Speed Physical Layer Extension in the 2.4 Ghz Band*, 1999. ANSI/IEEE std 802.11b.

[MBJ99] David A. Maltz, Josh Broch, and David B. Johnson. Experience Designing and Building a Multi-Hop Wireless Ad-hoc Network Testbed. Technical report, School of Computer Science, Carnegie Mellon University, USA, March 1999.

[Mit97] Tom M. Mitchell. *Machine Learning- International Edition*. McGraw-Hill, 1997. ISBN 0-07-042807-7.

[nsh] The Network Simulator - ns-2. Available at http://www.isi.edu/nsnam/ns/.

[PH76] E. S. Pearson and H. O. Hartley, editors. *Biometrika tables for statisticians*, volume 2. London: Biometrika Trust, 1976. ISBN 0-904653-11-0.

[Plu82] David C. Plummer. *An Ethernet Address Resolution Protocol*, 1982. RFC826.

[PRD00a] Charles E. Perkins, Elizabeth M. Royer, and Samir R. Das. Ad hoc On-Demand Distance Vector (AODV) Routing, July 2000. Internet-Draft version 6, obsoleted by [PRD00b].

[PRD00b] Charles E. Perkins, Elizabeth M. Royer, and Samir R. Das. Ad hoc On-Demand Distance Vector (AODV) Routing, November 2000. Internet-Draft version 7, obsoleted by [PRD01].

[PRD01] Charles E. Perkins, Elizabeth M. Royer, and Samir R. Das. Ad hoc On-Demand Distance Vector (AODV) Routing, March 2001. Internet-Draft version 8. Available at http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-08.txt. Work in progress.

[Qay00] Amir Qayyum. *Analysis and Evaluation of Channel Access Schemes and Routing Protocols in Wireless LANs*. PhD thesis, University of Paris-Sud, Orsay, France, November 2000. Host Laboratory: INRIA - Rocquencourt.

[RBP00] Elizabeth M. Royer, Santa Barbara, and Charles E. Perkins. IP Address Autoconfiguration for Ad Hoc Networks, July 2000. Internet-Draft version 00.

[Sam00] Samir R. Das and Charles E. Perkins and Elizabeth M. Royer. Performance Comparison of Two On-demand Routing Protocols for Ad Hoc Networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM), Tel Aviv, Israel*, pages 3–12, March 2000.

[SM99] S.Corson and J. Macker. Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations. Technical report, IETF, January 1999. RFC 2501, Available at http://www.ietf.org/rfc/rfc2501.txt.

[TMW97] K. Thompson, G. Miller, and R. Wilder. Wide-Area Internet Traffic Patterns and Characteristics. *IEEE Network*, 11(6), November/December 1997.

A. Simulation Overview

Jitter test	
With jitter	30 tests
Without jitter	30 tests

Piggyback test with jitter	
Holdback time in seconds	
0.0	30 tests
0.2	30 tests
0.4	30 tests
0.6	30 tests
0.8	30 tests
1.0	30 tests

Piggyback test without jitter	
Holdback time in seconds	
0.0	32 tests
0.2	32 tests
0.4	32 tests
0.6	34 tests
0.8	31 tests
1.0	32 tests

Piggyback test with jitter without mobility	
Holdback time in seconds	
0.0	33 tests
0.2	32 tests
0.4	32 tests
0.6	32 tests
0.8	32 tests
1.0	32 tests

Constant test - hello interval	
Hello interval in seconds	
0.5	30 tests
1.0	30 tests
1.5	30 tests
2.0	30 tests
2.5	30 tests
3.0	30 tests
3.5	30 tests

Constant test - TC interval	
TC interval in seconds	
1.0	32 tests
2.0	32 tests
3.0	33 tests
4.0	32 tests
5.0	35 tests
6.0	32 tests
7.0	32 tests
8.0	35 tests
9.0	32 tests
10.0	32 tests
11.0	32 tests
12.0	32 tests

Link status test	
Link rate	
1:1	32 tests
2:3	32 tests

Mobility test		
Node speed in m/s	OLSR	AODV
0.0	32 tests	32 tests
2.5	32 tests	32 tests
5.0	32 tests	32 tests
7.5	36 tests	32 tests
10.0	32 tests	32 tests
12.5	31 tests	32 tests
15.0	35 tests	32 tests

Density test - variable traffic		
Number of nodes	OLSR	AODV
10	31 tests	31 tests
20	30 tests	30 tests
50	32 tests	32 tests
75	35 tests	32 tests
100	32 tests	36 tests
125	32 tests	32 tests

Density test - constant traffic		
Number of nodes	OLSR	AODV
10	30 tests	31 tests
20	35 tests	34 tests
50	32 tests	32 tests
75	32 tests	32 tests
100	32 tests	32 tests
125	32 tests	32 tests

Variable duration test		
Stream Duration in seconds	OLSR	AODV
1.0	32 tests	32 tests
5.0	32 tests	32 tests
10.0	32 tests	32 tests
20.0	32 tests	32 tests
40.0	32 tests	32 tests
80.0	32 tests	34 tests
120.0	32 tests	36 tests
190.0	32 tests	32 tests
240.0	32 tests	32 tests

Bulk transfer test		
TCP transfer pr second	OLSR	AODV
6.0	32 tests	32 tests
8.0	35 tests	32 tests
10.0	36 tests	32 tests
12.0	32 tests	32 tests
14.0	32 tests	32 tests
16.0	33 tests	32 tests
18.0	32 tests	32 tests

Transfer time test		
	OLSR	AODV
	31 tests	30 tests

Variable load test		
Number of streams	OLSR	AODV
0	31 tests	30 tests
5	32 tests	30 tests
10	32 tests	32 tests
15	32 tests	33 tests
20	32 tests	32 tests
25	32 tests	32 tests
50	32 tests	32 tests
75	32 tests	32 tests
100	32 tests	32 tests

Cluster test		
	OLSR	AODV
	31 tests	32 tests

B. Simulation Data

This appendix states the results from the simulations – all numbers are stated with three significant digits. For each test set there is a table with mean values and a table with deviations.

The measured variables are listed in the leftmost columns. Measures of the form `count-*` are total counts for the entire simulation (250 seconds). Measures of the form `rate-*` are measures of bytes or packets per seconds. `time-avgpacketdelay` is the average packet delay of application layer data packets. `rate-bandwidth-*` is measures of bandwidth usage. `rate-{MAC,RTR,IFQ}-*` are measures of the drop reasons. The words after `rate` are, respectively, the layer that dropped the packet, the reason for the drop, and the type of packet that was dropped.

If the chi-square test of independence has been performed for a particular set of numbers it is also stated in this chapter.

Jitter Test - Standard Deviations		
	OLSR	
	0.0	1.0
Number of simulations	30.0	30.0
count-noroutedrop	14100.	2630.
count-olsr_hello	.000	12.1
count-olsr_tc	70.5	57.2
count-olsr_total	70.5	59.0
count-received	6780.	3810.
count-sent	.000	.000
rate-IFQ_—_ARP	.925	1.02
rate-IFQ_—_OLSR	.734	.531
rate-IFQ_—_cbr	14.1	7.86
rate-IFQ_ARP_cbr	2.79	1.87
rate-IFQ_END_cbr	.0158	.0131
rate-MAC_—_ARP	1.07	.819
rate-MAC_—_OLSR	12.0	4.48
rate-MAC_—_cbr	11.9	3.05
rate-MAC_BSY_MAC	.0966	.0821
rate-MAC_COL_MAC	33.0	22.6
rate-MAC_RET_MAC	9.47	2.31
rate-RTR_LOOP_cbr	.523	.522
rate-RTR_NRTE_cbr	56.6	10.5
rate-RTR_TTL_cbr	.591	.370
rate-bandwidth_byterate_ARP	228.	92.3
rate-bandwidth_byterate_MAC	27800.	5440.
rate-bandwidth_byterate_OLSR	359.	127.
rate-bandwidth_byterate_cbr	21400.	4650.
rate-bandwidth_packetrate_ARP	2.85	1.15
rate-bandwidth_packetrate_MAC	674.	132.
rate-bandwidth_packetrate_OLSR	4.98	1.76
rate-bandwidth_packetrate_cbr	158.	34.2
time-avgpacketdelay	.212	.115

Table B.1.: *Jitter Test - Standard Deviations.*

Jitter Test - Means		
	OLSR	
	0.0	1.0
Number of simulations	30.0	30.0
count-noroutedrop	41600.	9430.
count-olsr_hello	6250.	6250.
count-olsr_tc	1670.	1750.
count-olsr_total	7920.	8000.
count-received	14100.	27900.
count-sent	62300.	62300.
rate-IFQ_—_ARP	2.11	3.51
rate-IFQ_—_OLSR	1.36	3.65
rate-IFQ_—_cbr	29.2	60.2
rate-IFQ_ARP_cbr	3.94	8.15
rate-IFQ_END_cbr	.0248	.0352
rate-MAC_—_ARP	1.49	3.23
rate-MAC_—_OLSR	19.6	47.9
rate-MAC_—_cbr	11.1	37.6
rate-MAC_BSY_MAC	.179	.480
rate-MAC_COL_MAC	27.9	121.
rate-MAC_RET_MAC	9.17	28.1
rate-RTR_LOOP_cbr	.916	1.64
rate-RTR_NRTE_cbr	166.	37.6
rate-RTR_TTL_cbr	1.13	1.92
rate-bandwidth_byterate_ARP	197.	642.
rate-bandwidth_byterate_MAC	27000.	92800.
rate-bandwidth_byterate_OLSR	2560.	3850.
rate-bandwidth_byterate_cbr	22000.	72700.
rate-bandwidth_packetrate_ARP	2.46	8.02
rate-bandwidth_packetrate_MAC	659.	2250.
rate-bandwidth_packetrate_OLSR	35.5	53.5
rate-bandwidth_packetrate_cbr	162.	535.
time-avgpacketdelay	.174	.596

Table B.2.: *Jitter Test - Means.*

Received			
Interval	Without jitter	With jitter	Sum
4001-8000	6	0	6
8001-12000	8	0	8
12001-16000	5	0	5
16001-20000	4	0	4
20001-24000	4	4	8
24001-28000	2	12	14
28001-32000	1	8	9
32001-36000	0	6	6
Sum	30	30	60

Chi-square	41.5873
Degree of freedom	7
$Q(\chi^2 \vert df)$	1.0000

Table B.3.: *The calculation of the dependency probability of packets received caused by jitter with an interval of 4000.*

Noroute drop			
Interval	Without jitter	With jitter	Sum
4001-8000	0	10	10
8001-12000	0	15	15
12001-16000	2	5	7
16001-20000	1	0	1
20001-24000	1	0	1
24001-28000	2	0	2
28001-32000	3	0	3
32001-36000	1	0	1
36001-40000	2	0	2
40001-44000	1	0	1
44001-48000	1	0	1
48001-52000	7	0	7
52001-56000	7	0	7
56001-60000	2	0	2
Sum	30	30	60

Chi-square	54.2857
Degree of freedom	13
$Q(\chi^2 \mid df)$	1.0000

Table B.4.: *The calculation of the dependency probability of packets dropped due to route unavailability caused by jitter with an interval of 4000.*

Piggyback Test With Jitter and Mobility - Means						
	OLSR					
	0.0	0.2	0.4	0.6	0.8	1.0
Number of simulations	30.0	30.0	30.0	30.0	30.0	30.0
count-noroutedrop	8740.	6610.	8100.	8560.	7330.	7650.
count-olsr_hello	8370.	8360.	8380.	8370.	8370.	8370.
count-olsr_tc	1920.	1940.	1920.	1920.	1930.	1940.
count-olsr_total	10300.	10300.	10300.	10300.	10300.	10300.
count-received	26300.	28400.	26800.	25300.	27800.	27000.
count-sent	62300.	62300.	62300.	62300.	62300.	62300.
rate-IFQ_—_	.000	.00667	.000	.00667	.000	.000
rate-IFQ_—_ARP	4.18	4.29	4.27	4.33	4.42	4.53
rate-IFQ_—_OLSR	12.8	6.36	4.75	4.14	3.70	3.53
rate-IFQ_—_cbr	67.4	69.2	67.4	71.7	68.0	69.4
rate-IFQ_ARP_cbr	9.78	9.46	9.88	10.0	10.2	10.4
rate-IFQ_END_cbr	.056	.0553	.0622	.0622	.0578	.056
rate-MAC_—_ARP	3.74	3.48	3.37	3.63	3.58	3.71
rate-MAC_—_OLSR	69.9	70.2	48.7	40.8	38.4	34.7
rate-MAC_—_cbr	38.4	37.7	38.5	38.6	37.3	37.2
rate-MAC_BSY_MAC	.427	.406	.423	.420	.430	.440
rate-MAC_COL_MAC	120.	117.	106.	112.	117.	116.
rate-MAC_RET_MAC	28.8	27.9	29.8	29.2	27.8	27.7
rate-RTR_LOOP_cbr	1.43	1.14	1.35	1.33	1.25	1.49
rate-RTR_NRTE_cbr	34.9	26.3	32.3	34.1	29.2	30.5
rate-RTR_TTL_cbr	1.64	1.32	1.38	1.55	1.31	1.47
rate-bandwidth_byterate_ARP	745.	704.	731.	761.	743.	766.
rate-bandwidth_byterate_MAC	91400.	89900.	89500.	90800.	89300.	91400.
rate-bandwidth_byterate_OLSR	8720.	5440.	4410.	3950.	3730.	3560.
rate-bandwidth_byterate_cbr	69800.	68600.	67500.	68700.	68000.	70600.
rate-bandwidth_packetrate_ARP	9.31	8.79	9.13	9.51	9.29	9.57
rate-bandwidth_packetrate_MAC	2220.	2180.	2170.	2200.	2170.	2220.
rate-bandwidth_packetrate_OLSR	121.	75.5	61.2	54.8	51.8	49.4
rate-bandwidth_packetrate_cbr	513.	505.	496.	505.	500.	519.
time-avgpacketdelay	.573	.587	.618	.620	.603	.624

Table B.5.: *Piggyback Test With Jitter and Mobility - Means.*

Piggyback Test With Jitter and Mobility - Standard Deviations						
	OLSR					
	0.0	0.2	0.4	0.6	0.8	1.0
Number of simulations	30.0	30.0	30.0	30.0	30.0	30.0
count-noroutedrop	2840.	2720.	2250.	3010.	3090.	2650.
count-olsr_hello	25.2	25.4	21.1	24.3	21.3	26.2
count-olsr_tc	85.7	94.5	111.	87.7	100.	94.4
count-olsr_total	80.2	96.3	113.	91.8	104.	99.9
count-received	4480.	4380.	3870.	4000.	4660.	4850.
count-sent	23.0	17.3	15.3	21.5	21.8	26.1
rate-IFQ_—	.000	.000	.000	.000	.000	.000
rate-IFQ_—_ARP	1.54	1.82	1.54	1.25	1.54	1.28
rate-IFQ_—_OLSR	2.97	1.22	1.05	.675	.792	.599
rate-IFQ_—_cbr	12.0	12.7	11.6	9.58	12.0	11.4
rate-IFQ_ARP_cbr	2.95	2.96	2.60	2.17	2.78	2.16
rate-IFQ_END_cbr	.0233	.0201	.0179	.0222	.024	.024
rate-MAC_—_ARP	1.31	1.19	.920	1.08	.885	1.18
rate-MAC_—_OLSR	13.2	7.70	5.31	6.18	4.53	4.44
rate-MAC_—_cbr	4.77	3.70	5.62	3.91	5.45	4.55
rate-MAC_BSY_MAC	.0906	.104	.112	.0983	.130	.0954
rate-MAC_COL_MAC	24.9	30.3	26.1	24.9	28.4	28.1
rate-MAC_RET_MAC	3.18	2.60	4.06	3.62	4.03	3.38
rate-RTR_LOOP_cbr	.462	.513	.788	.693	.545	.684
rate-RTR_NRTE_cbr	11.4	10.8	8.99	12.0	12.3	10.6
rate-RTR_TTL_cbr	.478	.468	.465	.508	.325	.559
rate-bandwidth_byterate_ARP	145.	137.	150.	117.	133.	119.
rate-bandwidth_byterate_MAC	6710.	8150.	8030.	5590.	7740.	7860.
rate-bandwidth_byterate_OLSR	742.	301.	190.	121.	112.	118.
rate-bandwidth_byterate_cbr	5400.	6880.	5940.	5540.	6600.	6880.
rate-bandwidth_packetrate_ARP	1.82	1.71	1.88	1.46	1.66	1.49
rate-bandwidth_packetrate_MAC	162.	199.	193.	137.	188.	191.
rate-bandwidth_packetrate_OLSR	10.3	4.18	2.63	1.68	1.55	1.64
rate-bandwidth_packetrate_cbr	39.7	50.6	43.7	40.7	48.5	50.6
time-avgpacketdelay	.142	.172	.178	.123	.170	.134

Table B.6.: *Piggyback Test With Jitter and Mobility - Standard Deviations.*

Received							
	Holdback time						
Interval	0.0	0.2	0.4	0.6	0.8	1.0	Sum
16001-20000	3	0	0	2	1	1	7
20001-24000	5	6	6	9	4	5	35
24001-28000	10	6	13	11	12	13	65
28001-32000	8	13	8	7	6	6	48
32001-36000	4	5	3	1	7	3	23
36001-40000	0	0	0	0	0	2	2
Sum	30	30	30	30	30	30	180

Chi-square	31.3002
Degree of freedom	25
$Q(\chi^2\|df)$	0.8207

Table B.7.: *The calculation of the dependency probability of packets received caused by piggybacking and jitter with an interval of 4000.*

Noroute drop							
	Holdback time						
Interval	0.0	0.2	0.4	0.6	0.8	1.0	Sum
0-4000	0	5	1	3	1	1	1
4001-8000	13	20	15	11	21	16	96
8001-12000	14	4	13	11	5	10	57
12001-16000	3	1	1	5	2	3	15
16001-20000	0	0	0	0	1	0	1
Sum	30	30	30	30	30	30	180

Chi-square	32.5318
Degree of freedom	20
$Q(\chi^2\|df)$	0.9620

Table B.8.: *The calculation of the dependency probability of packets dropped due to route unavailability caused by piggybacking and jitter with an interval of 4000.*

Piggyback Test Without Jitter and With Mobility - Means						
	OLSR					
	0.0	0.2	0.4	0.6	0.8	1.0
Number of simulations	32.0	32.0	32.0	34.0	31.0	32.0
count-noroutedrop	40200.	45900.	44000.	43300.	45700.	40000.
count-olsr_hello	6250.	6250.	6250.	6250.	6250.	6170.
count-olsr_tc	1620.	1620.	1630.	1630.	1620.	1620.
count-olsr_total	7870.	7870.	7880.	7880.	7870.	7700.
count-received	14900.	13000.	13300.	13000.	13400.	15400.
count-sent	62300.	62300.	62300.	62300.	62300.	62300.
rate-IFQ_—_ARP	2.11	1.71	1.84	2.45	1.36	2.02
rate-IFQ_—_OLSR	4.25	1.05	1.51	1.68	.888	1.20
rate-IFQ_—_cbr	31.6	18.8	31.0	34.0	17.5	29.0
rate-IFQ_ARP_cbr	4.07	2.78	3.69	3.84	2.93	4.48
rate-IFQ_END_cbr	.0273	.0239	.0233	.0277	.0259	.0362
rate-MAC_—_ARP	1.93	1.05	1.39	2.46	1.29	2.16
rate-MAC_—_OLSR	24.9	16.5	17.4	17.9	14.5	18.3
rate-MAC_—_cbr	11.7	6.58	8.48	10.2	6.59	11.5
rate-MAC_BSY_MAC	.223	.192	.189	.238	.164	.231
rate-MAC_COL_MAC	40.6	13.9	30.0	34.5	15.9	37.4
rate-MAC_RET_MAC	9.38	5.74	6.79	8.09	5.55	9.01
rate-RTR_LOOP_cbr	.944	.932	.829	.969	.523	1.07
rate-RTR_NRTE_cbr	161.	184.	176.	173.	183.	160.
rate-RTR_TTL_cbr	1.35	.886	.976	1.35	.933	1.43
rate-bandwidth_byterate_ARP	217.	120.	145.	196.	121.	222.
rate-bandwidth_byterate_MAC	29800.	16900.	22200.	25100.	17100.	29700.
rate-bandwidth_byterate_OLSR	3470.	2440.	2480.	2470.	2370.	2440.
rate-bandwidth_byterate_cbr	24300.	14500.	18500.	20400.	14600.	24500.
rate-bandwidth_packetrate_ARP	2.71	1.50	1.82	2.45	1.51	2.78
rate-bandwidth_packetrate_MAC	727.	414.	542.	611.	417.	724.
rate-bandwidth_packetrate_OLSR	48.1	33.9	34.5	34.2	32.9	33.9
rate-bandwidth_packetrate_cbr	179.	107.	136.	150.	108.	180.
time-avgpacketdelay	.181	.0817	.140	.163	.0744	.169

Table B.9.: *Piggyback Test Without Jitter and With Mobility - Means.*

Piggyback Test Without Jitter and With Mobility - Standard Deviations						
	OLSR					
	0.0	0.2	0.4	0.6	0.8	1.0
Number of simulations	32.0	32.0	32.0	34.0	31.0	32.0
count-noroutedrop	15600.	11500.	13500.	16300.	11600.	16900.
count-olsr_hello	.000	.000	.000	.000	.000	442.
count-olsr_tc	78.8	66.9	84.6	94.0	82.1	136.
count-olsr_total	78.8	66.9	84.6	94.0	82.1	755.
count-received	7790.	6580.	6360.	7980.	6180.	8790.
count-sent	27.9	25.8	25.1	24.7	26.9	25.2
rate-IFQ_—_ARP	1.03	1.07	1.27	1.18	1.02	1.17
rate-IFQ_—_OLSR	2.60	.921	.946	.779	.811	.831
rate-IFQ_—_cbr	17.5	15.2	19.8	19.9	17.3	19.5
rate-IFQ_ARP_cbr	2.79	2.88	3.00	3.61	2.70	3.18
rate-IFQ_END_cbr	.0146	.021	.0169	.0191	.0179	.0239
rate-MAC_—_ARP	.931	1.20	1.21	1.54	1.11	1.29
rate-MAC_—_OLSR	18.6	11.1	11.6	11.5	7.43	10.2
rate-MAC_—_cbr	12.6	8.46	11.4	12.9	9.99	12.9
rate-MAC_BSY_MAC	.140	.147	.160	.162	.173	.142
rate-MAC_COL_MAC	42.6	25.5	43.8	44.7	31.2	44.4
rate-MAC_RET_MAC	9.74	6.86	8.53	9.65	7.81	9.72
rate-RTR_LOOP_cbr	.637	.714	.653	.606	.506	.611
rate-RTR_NRTE_cbr	62.3	46.2	53.9	65.1	46.4	66.7
rate-RTR_TTL_cbr	.763	.735	.628	.790	.896	.669
rate-bandwidth_byterate_ARP	243.	201.	217.	276.	197.	269.
rate-bandwidth_byterate_MAC	30100.	21300.	28300.	31300.	22900.	31900.
rate-bandwidth_byterate_OLSR	1460.	332.	374.	319.	196.	221.
rate-bandwidth_byterate_cbr	23200.	16900.	21900.	24300.	17600.	24800.
rate-bandwidth_packetrate_ARP	3.04	2.51	2.71	3.45	2.47	3.37
rate-bandwidth_packetrate_MAC	730.	519.	688.	761.	556.	774.
rate-bandwidth_packetrate_OLSR	20.3	4.61	5.19	4.42	2.72	3.07
rate-bandwidth_packetrate_cbr	170.	124.	161.	179.	130.	183.
time-avgpacketdelay	.226	.162	.236	.241	.156	.224

Table B.10.: *Piggyback Test Without Jitter and With Mobility - Standard Deviations.*

Received							
	Holdback time						
Interval	0.0	0.2	0.4	0.6	0.8	1.0	Sum
0-4000	3	1	0	0	0	1	5
4001-8000	4	7	8	12	4	5	40
8001-12000	7	8	9	10	12	9	55
12001-16000	2	8	6	3	8	4	31
16001-20000	8	3	4	2	2	2	21
20001-24000	5	2	2	2	2	2	15
24001-28000	2	1	2	3	2	3	13
28001-32000	0	2	1	1	1	4	9
32001-36000	1	0	0	1	0	0	2
Sum	32	32	32	34	31	30	190

Chi-square	45.3530
Degree of freedom	40
$Q(\chi^2 \| df)$	0.7414

Table B.11.: *The calculation of the dependency probability of packets received caused by piggybacking and without jitter with an interval of 4000.*

Noroute drop							
	Holdback time						
Interval	0.0	0.2	0.4	0.6	0.8	1.0	Sum
0-4000	1	0	0	1	0	0	2
8001-12000	1	0	1	2	0	1	5
12001-16000	1	0	1	2	2	4	10
16001-20000	1	3	2	1	1	2	10
20001-24000	2	0	0	0	0	1	3
24001-28000	1	1	1	1	0	2	6
28001-32000	3	1	0	1	1	0	6
32001-36000	3	1	2	1	1	0	8
36001-40000	2	0	3	1	1	1	8
40001-44000	2	2	0	0	1	2	7
44001-48000	0	5	4	1	4	2	16
48001-52000	4	8	6	9	14	3	44
52001-56000	7	6	11	11	4	9	48
56001-60000	4	5	1	3	2	3	18
Sum	32	32	32	34	31	30	191

Chi-square	68.9838
Degree of freedom	65
$Q(\chi^2 \mid df)$	0.6557

Table B.12.: *The calculation of the dependency probability of packets dropped due to route unavailability caused by piggybacking and without jitter with an interval of 4000.*

Piggyback Test With Jitter and Without Mobility - Means						
	OLSR					
	0.0	0.2	0.4	0.6	0.8	1.0
Number of simulations	33.0	32.0	32.0	32.0	32.0	32.0
count-noroutedrop	8530.	6480.	7340.	7800.	6800.	7430.
count-olsr_hello	5010.	5000.	5000.	5010.	5000.	5000.
count-olsr_tc	1460.	1490.	1500.	1450.	1490.	1450.
count-olsr_total	6470.	6490.	6500.	6460.	6490.	6460.
count-received	48700.	51100.	49300.	50200.	51000.	50400.
count-sent	62300.	62300.	62300.	62300.	62300.	62300.
rate-IFQ_—_ARP	.0444	.0815	.0536	.0426	.100	.0664
rate-IFQ_—_OLSR	1.96	.552	.420	.244	.298	.254
rate-IFQ_—_cbr	15.5	13.6	16.4	11.9	12.8	13.8
rate-IFQ_ARP_cbr	.352	.283	.363	.254	.316	.317
rate-IFQ_END_cbr	.004	.004	.004	.004	.004	.004
rate-MAC_—_ARP	.490	.462	.531	.442	.535	.489
rate-MAC_—_OLSR	85.7	75.5	56.2	45.7	42.9	36.7
rate-MAC_—_cbr	23.1	22.9	25.7	25.0	23.5	22.7
rate-MAC_BSY_MAC	.909	.924	.957	.947	.906	.917
rate-MAC_COL_MAC	264.	277.	290.	280.	270.	256.
rate-MAC_RET_MAC	3.99	4.17	4.27	4.03	4.07	3.85
rate-RTR_LOOP_cbr	1.16	.815	.887	.708	.806	.838
rate-RTR_NRTE_cbr	34.0	25.9	29.3	31.2	27.1	29.7
rate-RTR_TTL_cbr	.640	.574	.617	.622	.695	.573
rate-bandwidth_byterate_ARP	89.8	82.8	96.4	81.6	87.6	81.6
rate-bandwidth_byterate_MAC	109000.	111000.	114000.	113000.	109000.	107000.
rate-bandwidth_byterate_OLSR	7270.	4670.	3680.	3160.	2960.	2710.
rate-bandwidth_byterate_cbr	100000.	101000.	104000.	103000.	100000.	98300.
rate-bandwidth_packetrate_ARP	1.12	1.03	1.21	1.02	1.10	1.02
rate-bandwidth_packetrate_MAC	2680.	2730.	2800.	2760.	2690.	2620.
rate-bandwidth_packetrate_OLSR	101.	64.8	51.0	43.9	41.2	37.6
rate-bandwidth_packetrate_cbr	736.	745.	764.	757.	738.	723.
time-avgpacketdelay	.423	.401	.443	.353	.369	.333

Table B.13.: *Piggyback Test With Jitter and Without Mobility - Means.*

Piggyback Test With Jitter and Without Mobility - Standard Deviations						
	OLSR					
	0.0	0.2	0.4	0.6	0.8	1.0
Number of simulations	33.0	32.0	32.0	32.0	32.0	32.0
count-noroutedrop	4500.	4110.	4120.	5190.	5010.	6360.
count-olsr_hello	9.54	7.96	7.29	8.62	7.25	6.67
count-olsr_tc	169.	177.	108.	172.	138.	147.
count-olsr_total	169.	178.	108.	172.	137.	145.
count-received	5570.	6440.	5780.	5790.	6520.	7150.
count-sent	.000	.000	.000	.000	.000	.000
rate-IFQ_—_ARP	.0535	.0652	.045	.0484	.118	.0628
rate-IFQ_—_OLSR	1.47	.648	.340	.182	.251	.272
rate-IFQ_—_cbr	12.9	15.4	13.1	9.71	11.5	15.3
rate-IFQ_ARP_cbr	.274	.273	.260	.200	.292	.251
rate-IFQ_END_cbr	.000	.000	.000	.000	.000	.000
rate-MAC_—_ARP	.358	.257	.261	.205	.343	.325
rate-MAC_—_OLSR	30.3	14.7	9.97	8.75	9.35	8.74
rate-MAC_—_cbr	11.4	10.3	9.40	10.9	10.7	11.4
rate-MAC_BSY_MAC	.320	.344	.252	.325	.309	.314
rate-MAC_COL_MAC	115.	107.	82.2	103.	102.	111.
rate-MAC_RET_MAC	3.10	2.78	2.14	2.39	2.41	2.27
rate-RTR_LOOP_cbr	1.07	.650	.729	.601	.659	.737
rate-RTR_NRTE_cbr	18.0	16.4	16.5	20.7	20.0	25.5
rate-RTR_TTL_cbr	.493	.246	.287	.409	.364	.429
rate-bandwidth_byterate_ARP	34.3	30.4	29.8	23.5	29.5	28.3
rate-bandwidth_byterate_MAC	20600.	18400.	13200.	17100.	18100.	23200.
rate-bandwidth_byterate_OLSR	981.	634.	297.	326.	200.	203.
rate-bandwidth_byterate_cbr	14300.	12000.	9350.	11300.	12300.	17700.
rate-bandwidth_packetrate_ARP	.429	.380	.373	.293	.368	.354
rate-bandwidth_packetrate_MAC	496.	442.	316.	410.	434.	562.
rate-bandwidth_packetrate_OLSR	13.6	8.80	4.12	4.52	2.78	2.83
rate-bandwidth_packetrate_cbr	105.	87.9	68.7	83.2	90.7	130.
time-avgpacketdelay	.337	.358	.309	.246	.291	.317

Table B.14.: *Piggyback Test With Jitter and Without Mobility - Standard Deviations.*

Received							
	Holdback time						
Interval	0.0	0.2	0.4	0.6	0.8	1.0	Sum
24001-28000	0	0	0	0	0	1	1
32001-36000	1	0	1	1	0	0	3
36001-40000	0	2	0	1	3	0	6
40001-44000	5	4	8	3	4	4	28
44001-48000	10	3	3	4	0	7	27
48001-52000	7	7	6	9	8	5	42
52001-56000	7	6	12	9	9	5	48
56001-60000	3	9	2	5	7	10	36
60001-64000	0	1	0	0	1	0	2
Sum	33	32	32	32	32	32	193

Chi-square	50.9722
Degree of freedom	40
$Q(\chi^2\|df)$	0.8855

Table B.15.: *The calculation of the dependency probability of packets received caused by piggybacking and jitter with an interval of 4000 and with no mobility.*

Noroute drop							
	Holdback time						
Interval	0.0	0.2	0.4	0.6	0.8	1.0	Sum
0-4000	3	11	5	9	13	9	50
4001-8000	17	12	16	10	9	13	77
8001-12000	6	6	7	8	7	5	39
12001-16000	4	1	3	4	1	4	17
16001-20000	2	2	0	0	1	0	5
20001-24000	1	0	1	0	0	0	2
24001-28000	0	0	0	0	1	0	1
28001-32000	0	0	0	1	0	0	1
32001-36000	0	0	0	0	0	1	1
Sum	33	32	32	32	32	32	193

Chi-square	41.6762
Degree of freedom	40
$Q(\chi^2\|df)$	0.6023

Table B.16.: *The calculation of the dependency probability of packets dropped due to route unavailability caused by piggybacking and jitter with an interval of 4000 and with no mobility.*

Constant Test, Hello Interval - Means							
	OLSR						
	0.5	1.0	1.5	2.0	2.5	3.0	3.5
Number of simulations	30.0	30.0	30.0	33.0	30.0	30.0	30.0
count-noroutedrop	5470.	7590.	9290.	11400.	13200.	13300.	13500.
count-olsr_hello	24800.	12500.	8310.	6250.	5000.	4180.	3590.
count-olsr_tc	1920.	1910.	1880.	1570.	1590.	1610.	1630.
count-olsr_total	26800.	14400.	10200.	7820.	6590.	5790.	5210.
count-received	25900.	26100.	27100.	26000.	23800.	23000.	21400.
count-sent	62300.	62300.	62300.	62300.	62300.	62300.	62300.
rate-IFQ_—_ARP	5.26	4.62	4.33	3.65	3.73	3.94	4.32
rate-IFQ_—_OLSR	22.6	15.9	12.3	8.73	8.70	8.96	8.98
rate-IFQ_—_cbr	82.9	73.7	63.4	60.7	59.8	60.4	62.0
rate-IFQ_ARP_cbr	11.0	10.1	9.76	8.55	8.96	9.73	11.2
rate-IFQ_END_cbr	.0624	.0582	.0571	.0523	.0569	.0627	.0822
rate-MAC_—_ARP	3.57	3.65	3.36	3.67	3.10	3.04	3.26
rate-MAC_—_OLSR	116.	83.2	67.1	56.8	48.0	43.2	38.2
rate-MAC_—_cbr	38.0	37.7	36.9	36.1	37.6	38.7	40.4
rate-MAC_BSY_MAC	.355	.412	.418	.459	.394	.322	.274
rate-MAC_COL_MAC	109.	115.	114.	120.	102.	83.3	71.4
rate-MAC_RET_MAC	28.0	28.2	27.8	26.9	29.5	31.8	34.4
rate-RTR_LOOP_cbr	.798	1.15	1.16	1.66	1.24	.887	.616
rate-RTR_NRTE_cbr	21.9	30.3	37.0	45.3	52.6	53.0	53.9
rate-RTR_TTL_cbr	.770	1.23	1.40	1.96	1.89	1.26	1.19
rate-bandwidth_byterate_ARP	708.	718.	728.	681.	663.	703.	752.
rate-bandwidth_byterate_MAC	86500.	89200.	89300.	90400.	85400.	79700.	76700.
rate-bandwidth_byterate_OLSR	13900.	10100.	8690.	6730.	6280.	6110.	5750.
rate-bandwidth_byterate_cbr	62900.	67100.	68700.	71100.	65100.	58600.	54600.
rate-bandwidth_packetrate_ARP	8.85	8.98	9.10	8.51	8.29	8.79	9.41
rate-bandwidth_packetrate_MAC	2090.	2160.	2170.	2200.	2070.	1930.	1850.
rate-bandwidth_packetrate_OLSR	193.	141.	121.	93.4	87.2	84.8	79.8
rate-bandwidth_packetrate_cbr	463.	493.	505.	523.	479.	431.	401.
time-avgpacketdelay	.559	.560	.531	.599	.561	.528	.552

Table B.17.: *Constant Test, Hello Interval - Means.*

Table B.18.: *Constant Test, Hello Interval - Standard Deviations.*

Constant Test, Hello Interval - Standard Deviations							
	OLSR						
	0.5	1.0	1.5	2.0	2.5	3.0	3.5
Number of simulations	30.0	30.0	30.0	33.0	30.0	30.0	30.0
count-noroutedrop	2290.	2700.	3430.	3320.	3420.	3820.	3810.
count-olsr_hello	61.9	19.0	12.7	12.0	9.55	7.00	5.08
count-olsr_tc	94.6	113.	134.	56.6	66.6	78.3	60.3
count-olsr_total	122.	120.	131.	59.3	65.2	78.4	60.5
count-received	4020.	4130.	4100.	4570.	2750.	4100.	4460.
count-sent	28.8	17.4	22.4	24.1	24.8	23.7	26.1
rate-IFQ_—_ARP	1.29	1.42	1.61	1.11	1.06	1.60	1.54
rate-IFQ_—_OLSR	4.68	3.08	3.09	1.88	1.65	1.98	1.87
rate-IFQ_—_cbr	12.1	13.8	12.2	11.7	9.05	11.8	11.4
rate-IFQ_ARP_cbr	2.21	2.48	2.89	2.08	1.66	2.65	2.51
rate-IFQ_END_cbr	.0234	.0178	.0203	.0213	.0219	.019	.0263
rate-MAC_—_ARP	.932	1.37	1.26	1.24	.943	.687	.940
rate-MAC_—_OLSR	20.2	13.9	14.6	11.3	10.1	11.0	8.10
rate-MAC_—_cbr	4.14	4.38	4.46	4.65	4.14	4.36	3.93
rate-MAC_BSY_MAC	.0964	.0895	.0846	.0905	.0951	.100	.0586
rate-MAC_COL_MAC	22.7	23.6	28.9	22.7	24.2	21.3	15.3
rate-MAC_RET_MAC	3.33	3.39	3.32	3.69	3.19	4.30	3.15
rate-RTR_LOOP_cbr	.471	.512	.601	.715	.516	.463	.330
rate-RTR_NRTE_cbr	9.15	10.7	13.7	13.3	13.7	15.3	15.2
rate-RTR_TTL_cbr	.277	.435	.497	.566	.482	.523	.505
rate-bandwidth_byterate_ARP	93.4	145.	143.	112.	87.2	120.	99.4
rate-bandwidth_byterate_MAC	5460.	6160.	6510.	6130.	5780.	6110.	4650.
rate-bandwidth_byterate_OLSR	869.	976.	949.	564.	568.	726.	529.
rate-bandwidth_byterate_cbr	5100.	4950.	5140.	5140.	4630.	6100.	4240.
rate-bandwidth_packetrate_ARP	1.17	1.81	1.79	1.40	1.09	1.50	1.24
rate-bandwidth_packetrate_MAC	133.	149.	157.	149.	140.	150.	113.
rate-bandwidth_packetrate_OLSR	12.1	13.6	13.2	7.83	7.88	10.1	7.35
rate-bandwidth_packetrate_cbr	37.5	36.4	37.8	37.8	34.0	44.8	31.2
time-avgpacketdelay	.110	.120	.134	.139	.100	.130	.149

Constant Test, TC Interval - Means												
	OLSR											
	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0
Number of simulations	32.0	32.0	33.0	32.0	35.0	32.0	32.0	35.0	32.0	32.0	32.0	32.0
count-noroutedrop	7850.	8170.	9040.	10000.	11000.	11700.	11200.	13800.	12200.	12400.	13000.	13700.
count-olsr_hello	6250.	6250.	6250.	6250.	6250.	6250.	6250.	6250.	6250.	6250.	6250.	6250.
count-olsr_tc	8730.	4350.	2860.	2190.	1730.	1460.	1240.	1080.	954.	854.	769.	693.
count-olsr_total	15000.	10600.	9100.	8430.	7980.	7710.	7490.	7330.	7200.	7100.	7020.	6940.
count-received	26700.	27300.	27200.	26000.	26900.	25800.	26000.	26100.	24800.	24400.	23400.	23200.
count-sent	62300.	62300.	62300.	62300.	62300.	62300.	62300.	62300.	62300.	62300.	62300.	62300.
rate-IFQ_—_ARP	3.54	3.44	3.59	3.83	3.32	3.44	3.47	2.77	3.68	3.89	3.93	3.63
rate-IFQ_—_OLSR	51.8	25.1	15.9	12.5	9.39	8.45	7.10	5.78	5.54	5.04	4.64	4.31
rate-IFQ_—_cbr	76.0	69.5	65.7	65.2	59.0	58.6	59.5	52.6	60.2	60.2	60.7	59.7
rate-IFQ_ARP_cbr	7.74	7.82	8.26	8.44	7.82	8.28	8.32	6.87	8.91	9.13	9.30	8.80
rate-IFQ_END_cbr	.0346	.033	.0402	.0375	.039	.0421	.0409	.0387	.0376	.0436	.040	.0396
rate-MAC_—_ARP	3.45	3.24	3.43	3.33	3.02	3.42	3.43	2.81	3.71	4.02	3.93	3.91
rate-MAC_—_OLSR	260.	129.	89.4	73.9	59.2	56.0	49.7	44.5	43.2	39.1	37.8	36.3
rate-MAC_—_cbr	37.0	37.8	37.6	38.4	36.3	38.3	37.5	35.3	37.2	36.3	36.8	35.7
rate-MAC_BSY_MAC	.496	.476	.468	.474	.476	.444	.510	.426	.505	.532	.566	.547
rate-MAC_COL_MAC	151.	129.	128.	131.	120.	124.	127.	119.	136.	139.	146.	146.
rate-MAC_RET_MAC	25.3	27.5	27.6	28.0	27.2	28.7	27.9	26.3	26.8	26.2	26.3	25.4
rate-RTR_LOOP_cbr	1.30	1.25	1.51	1.67	1.75	1.65	2.24	1.77	2.68	3.23	3.58	3.87
rate-RTR_NRTE_cbr	31.3	32.6	36.1	39.9	43.9	46.5	44.7	54.9	48.4	49.1	51.5	54.2
rate-RTR_TTL_cbr	.847	1.23	1.56	1.83	2.05	2.08	2.66	2.30	3.13	3.62	4.03	4.21
rate-bandwidth_byterate_ARP	590.	615.	645.	636.	628.	660.	651.	576.	691.	674.	687.	664.
rate-bandwidth_byterate_MAC	93300.	92100.	92000.	94200.	91600.	91800.	94300.	88900.	95700.	97300.	99100.	98100.
rate-bandwidth_byterate_OLSR	26700.	14600.	10200.	8500.	7260.	6610.	5660.	5450.	4770.	4330.	4070.	3870.
rate-bandwidth_byterate_cbr	71800.	70700.	71400.	73400.	72300.	71400.	74600.	70800.	77000.	78600.	80400.	80400.
rate-bandwidth_packetrate_ARP	7.37	7.69	8.07	7.96	7.85	8.26	8.14	7.20	8.64	8.42	8.58	8.30
rate-bandwidth_packetrate_MAC	2270.	2230.	2230.	2290.	2230.	2230.	2290.	2160.	2330.	2370.	2410.	2390.
rate-bandwidth_packetrate_OLSR	370.	203.	142.	118.	101.	91.8	78.7	75.7	66.2	60.1	56.6	53.7
rate-bandwidth_packetrate_cbr	528.	520.	525.	540.	532.	525.	549.	521.	566.	578.	591.	591.
time-avgpacketdelay	.530	.537	.565	.601	.559	.587	.601	.562	.625	.643	.667	.660

Table B.19.: *Constant Test, TC Interval - Means.*

Constant Test, TC Interval - Standard Deviations												
	OLSR											
	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0
Number of simulations	32.0	32.0	33.0	32.0	35.0	32.0	32.0	35.0	32.0	32.0	32.0	32.0
count-noroutedrop	2170.	1990.	2180.	2400.	2960.	2530.	2960.	3330.	2690.	2810.	2440.	2870.
count-olsr_hello	11.6	12.2	11.5	10.6	11.8	15.1	10.2	10.6	10.2	13.2	8.65	10.4
count-olsr_tc	304.	139.	123.	70.5	57.4	46.6	40.0	44.9	31.2	33.9	27.6	26.4
count-olsr_total	307.	143.	123.	68.8	58.0	42.0	41.2	47.8	33.8	35.8	27.1	27.6
count-received	2770.	2930.	3280.	3030.	3950.	2860.	3580.	3040.	4130.	2290.	2980.	2810.
count-sent	54.2	.000	59.3	.000	54.2	.000	59.1	.000	55.6	.000	.000	.000
rate-IFQ_—_ARP	1.04	.924	1.06	.988	.774	.892	1.10	.680	.967	1.03	1.23	1.21
rate-IFQ_—_OLSR	8.39	3.88	2.60	1.90	1.60	1.50	1.26	1.11	.960	1.03	.857	.930
rate-IFQ_—_cbr	7.97	9.93	9.21	5.45	8.58	7.67	8.79	7.55	9.43	8.25	7.70	9.59
rate-IFQ_ARP_cbr	1.71	1.57	1.83	1.68	1.34	1.64	1.99	1.10	1.78	1.86	2.25	2.16
rate-IFQ_END_cbr	.0114	.0132	.0132	.0137	.0147	.0148	.0135	.0132	.0123	.0148	.0157	.013
rate-MAC_—_ARP	.852	.704	.907	1.07	.624	.858	.920	.615	.785	1.16	1.05	.981
rate-MAC_—_OLSR	50.8	20.4	18.2	12.2	9.94	7.70	7.44	7.89	6.77	7.74	4.76	5.32
rate-MAC_—_cbr	2.67	2.84	3.87	3.01	3.09	3.55	3.56	2.91	4.05	3.33	3.22	3.59
rate-MAC_BSY_MAC	.0891	.0739	.064	.0765	.0932	.084	.0931	.0733	.0909	.0824	.0948	.0775
rate-MAC_COL_MAC	17.5	19.4	22.7	19.1	19.0	18.8	23.8	23.8	24.3	23.4	21.9	21.2
rate-MAC_RET_MAC	2.15	2.40	3.02	2.41	2.74	2.53	2.41	2.31	2.55	2.28	2.32	2.45
rate-RTR_LOOP_cbr	.383	.457	.567	.457	.510	.413	.744	.630	.824	.894	1.06	1.14
rate-RTR_NRTE_cbr	8.67	7.94	8.72	9.54	11.7	10.1	11.8	13.3	10.8	11.2	9.70	11.5
rate-RTR_TTL_cbr	.271	.406	.241	.359	.495	.457	.636	.496	.769	.589	.793	.834
rate-bandwidth_byterate_ARP	75.6	74.5	93.8	100.	78.2	92.8	103.	66.1	95.1	98.7	119.	112.
rate-bandwidth_byterate_MAC	5080.	5140.	5620.	5020.	5320.	5300.	5900.	5440.	6880.	6000.	5610.	5010.
rate-bandwidth_byterate_OLSR	3060.	1400.	946.	696.	580.	420.	372.	409.	294.	269.	231.	218.
rate-bandwidth_byterate_cbr	4400.	4450.	4080.	4210.	4620.	4320.	5120.	4710.	5880.	5100.	4650.	4170.
rate-bandwidth_packetrate_ARP	.945	.931	1.17	1.26	.977	1.16	1.29	.827	1.19	1.23	1.48	1.40
rate-bandwidth_packetrate_MAC	124.	125.	135.	122.	130.	129.	144.	133.	168.	146.	136.	121.
rate-bandwidth_packetrate_OLSR	42.5	19.5	13.1	9.67	8.05	5.83	5.17	5.68	4.09	3.73	3.21	3.02
rate-bandwidth_packetrate_cbr	32.3	32.7	30.0	31.0	33.9	31.8	37.6	34.6	43.3	37.5	34.2	30.7
time-avgpacketdelay	.0857	.0999	.107	.0781	.106	.115	.125	.122	.131	.139	.146	.153

Table B.20.: *Constant Test, TC Interval - Standard Deviations.*

Link Hysteresis Test - Standard Deviations		
	OLSR	
	1.1	2.3
Number of simulations	32.0	32.0
count-noroutedrop	3050.	2820.
count-olsr_hello	10.9	9.25
count-olsr_tc	69.3	70.5
count-olsr_total	70.6	70.3
count-received	2470.	1830.
count-sent	3.28	3.98
rate-IFQ_—_ARP	1.09	.782
rate-IFQ_—_OLSR	1.72	1.52
rate-IFQ_—_cbr	8.48	7.77
rate-IFQ_ARP_cbr	1.84	1.46
rate-IFQ_END_cbr	.00979	.013
rate-MAC_—_ARP	.851	.850
rate-MAC_—_OLSR	9.49	10.1
rate-MAC_—_cbr	2.40	2.63
rate-MAC_BSY_MAC	.076	.0769
rate-MAC_COL_MAC	18.7	20.4
rate-MAC_RET_MAC	1.98	2.11
rate-RTR_LOOP_cbr	.479	.529
rate-RTR_NRTE_cbr	12.1	11.3
rate-RTR_TTL_cbr	.373	.428
rate-bandwidth_byterate_ARP	90.6	84.1
rate-bandwidth_byterate_MAC	4320.	5220.
rate-bandwidth_byterate_OLSR	570.	609.
rate-bandwidth_byterate_cbr	3820.	4480.
rate-bandwidth_packetrate_ARP	1.13	1.05
rate-bandwidth_packetrate_MAC	105.	127.
rate-bandwidth_packetrate_OLSR	7.91	8.45
rate-bandwidth_packetrate_cbr	28.1	33.0
time-avgpacketdelay	.111	.098

Table B.22.: *Link Hysteresis Test - Standard Deviations.*

Link Hysteresis Test - Means		
	OLSR	
	1.1	2.3
Number of simulations	32.0	32.0
count-noroutedrop	13500.	13200.
count-olsr_hello	6250.	6250.
count-olsr_tc	1710.	1740.
count-olsr_total	7960.	7990.
count-received	25400.	24300.
count-sent	62600.	62600.
rate-IFQ_—_ARP	3.12	3.68
rate-IFQ_—_OLSR	8.16	8.97
rate-IFQ_—_cbr	58.5	62.4
rate-IFQ_ARP_cbr	7.62	8.34
rate-IFQ_END_cbr	.0297	.0358
rate-MAC_—_ARP	3.37	3.66
rate-MAC_—_OLSR	55.3	55.3
rate-MAC_—_cbr	35.6	35.4
rate-MAC_BSY_MAC	.453	.431
rate-MAC_COL_MAC	126.	120.
rate-MAC_RET_MAC	25.9	26.3
rate-RTR_LOOP_cbr	1.62	1.76
rate-RTR_NRTE_cbr	53.6	52.6
rate-RTR_TTL_cbr	1.68	1.69
rate-bandwidth_byterate_ARP	613.	631.
rate-bandwidth_byterate_MAC	86600.	84800.
rate-bandwidth_byterate_OLSR	7360.	7360.
rate-bandwidth_byterate_cbr	68100.	65700.
rate-bandwidth_packetrate_ARP	7.67	7.89
rate-bandwidth_packetrate_MAC	2100.	2060.
rate-bandwidth_packetrate_OLSR	102.	102.
rate-bandwidth_packetrate_cbr	501.	483.
time-avgpacketdelay	.590	.634

Table B.21.: *Link Hysteresis Test - Means.*

Mobility Test - Means														
	OLSR							AODV						
	0.0	2.5	5.0	7.5	10.0	12.5	15.0	0.0	2.5	5.0	7.5	10.0	12.5	15.0
Number of simulations	32.0	32.0	32.0	36.0	32.0	31.0	35.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0
count-noroutedrop	7870.	12700.	13100.	14000.	14600.	16000.	15900.	1830.	8250.	9510.	10100.	10500.	10900.	11300.
count-olsr_hello	6250.	6250.	6250.	6250.	6250.	6250.	6250.	n/a						
count-olsr_tc	1560.	1690.	1800.	1880.	1940.	1990.	2040.	n/a						
count-olsr_total	7800.	7940.	8050.	8130.	8190.	8240.	8290.	n/a						
count-received	45500.	26200.	21200.	18000.	16000.	13900.	12600.	43100.	27000.	21500.	19400.	16800.	15300.	14700.
count-sent	62600.	62600.	62600.	62600.	62600.	62600.	62600.	62600.	62600.	62600.	62600.	62600.	62600.	62600.
rate-IFQ_—_AODV	n/a							.000	6.78	8.02	6.54	5.94	5.36	4.77
rate-IFQ_—_ARP	.176	3.04	5.10	5.64	5.67	5.26	5.37	.119	2.82	6.26	6.33	7.44	7.41	7.07
rate-IFQ_—_OLSR	3.43	8.29	9.30	8.84	8.57	7.56	7.10	n/a						
rate-IFQ_—_cbr	31.1	61.7	62.1	56.7	52.8	45.6	43.1	59.2	87.0	92.4	91.3	92.4	91.6	89.4
rate-IFQ_ARP_AODV	n/a							.133	2.49	4.45	4.61	5.49	5.49	5.57
rate-IFQ_ARP_cbr	.819	6.89	12.1	15.1	17.1	19.3	21.1	.250	2.33	4.86	5.05	5.79	6.02	5.75
rate-IFQ_END_AODV	n/a							.004	.0169	.0389	.0641	.104	.122	.129
rate-IFQ_END_cbr	.004	.0226	.0645	.108	.154	.193	.235	.000	.0045	.00661	.0076	.0117	.0132	.0137
rate-MAC_—_AODV	n/a							87.4	139.	139.	134.	135.	138.	140.
rate-MAC_—_ARP	1.07	3.26	4.63	5.05	5.44	5.90	6.28	1.11	7.04	11.5	11.0	12.3	12.6	11.9
rate-MAC_—_OLSR	87.8	58.4	48.8	43.3	39.9	37.7	35.9	n/a						
rate-MAC_—_cbr	27.4	33.9	43.2	52.6	59.1	65.7	72.1	28.8	24.3	26.2	30.9	35.5	38.7	42.4
rate-MAC_BSY_MAC	.848	.506	.340	.257	.217	.174	.145	1.01	.690	.561	.461	.408	.375	.349
rate-MAC_COL_MAC	287.	142.	89.4	63.3	49.1	39.2	32.0	356.	240.	182.	149.	130.	118.	109.
rate-MAC_RET_MAC	5.10	22.8	36.2	47.5	55.1	62.8	69.8	4.93	12.6	21.2	30.7	38.8	45.1	51.6
rate-RTR_CBK_cbr	n/a							5.13	11.5	19.5	26.4	32.8	37.1	41.2
rate-RTR_IFQ_AODV	n/a							.000	.0194	.00836	.0126	.0085	.008	.0224
rate-RTR_IFQ_cbr	n/a							6.32	5.91	6.74	6.40	6.77	7.64	6.89
rate-RTR_LOOP_cbr	1.06	1.88	1.61	1.82	1.60	1.81	1.72	.557	2.96	3.33	3.79	3.61	3.82	3.89
rate-RTR_NRTE_AODV	n/a							.0243	1.29	1.53	1.44	1.60	1.69	1.77
rate-RTR_NRTE_cbr	31.4	50.6	52.2	55.8	58.1	63.8	63.2	7.29	31.7	36.5	38.9	40.5	42.1	43.5
rate-RTR_TOUT_AODV	n/a							.000	.004	.000	.000	.00533	.010	.000
rate-RTR_TOUT_cbr	n/a							1.72	.371	.190	.131	.124	.116	.125
rate-RTR_TTL_AODV	n/a							7.68	13.2	14.1	14.4	15.5	15.9	16.2
rate-RTR_TTL_cbr	.288	1.73	1.66	1.59	1.48	1.53	1.26	.310	.990	1.23	1.01	1.29	1.07	1.05
rate-bandwidth_byterate_AODV	n/a							8800.	21000.	22400.	22500.	22900.	23300.	24000.
rate-bandwidth_byterate_ARP	135.	535.	886.	1130.	1260.	1440.	1550.	126.	661.	1150.	1280.	1430.	1500.	1480.
rate-bandwidth_byterate_MAC	101000.	89200.	81400.	76700.	73500.	71400.	69600.	114000.	113000.	108000.	104000.	103000.	102000.	101000.
rate-bandwidth_byterate_OLSR	7600.	7230.	7190.	7210.	7180.	7110.	7120.	n/a						
rate-bandwidth_byterate_cbr	89500.	71400.	58400.	49700.	44100.	40000.	36000.	96100.	80600.	71900.	64800.	60900.	56400.	53700.
rate-bandwidth_packetrate_AODV	n/a							89.5	212.	226.	227.	231.	235.	241.
rate-bandwidth_packetrate_ARP	1.68	6.69	11.1	14.1	15.7	17.9	19.3	1.57	8.27	14.4	15.9	17.9	18.7	18.5
rate-bandwidth_packetrate_MAC	2480.	2170.	1970.	1840.	1760.	1700.	1660.	2800.	2750.	2620.	2530.	2500.	2450.	2440.
rate-bandwidth_packetrate_OLSR	106.	100.	99.9	100.	99.7	98.8	99.0	n/a						
rate-bandwidth_packetrate_cbr	658.	525.	429.	366.	324.	294.	264.	707.	593.	528.	476.	448.	415.	395.
time-avgpacketdelay	.654	.640	.642	.600	.608	.566	.559	1.20	.724	.669	.714	.780	.840	.829

Table B.23.: *Mobility Test - Means.*

Mobility Test - Standard Deviations														
	OLSR							AODV						
	0.0	2.5	5.0	7.5	10.0	12.5	15.0	0.0	2.5	5.0	7.5	10.0	12.5	15.0
Number of simulations	32.0	32.0	32.0	36.0	32.0	31.0	35.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0
count-noroutedrop	6060.	2470.	2110.	1630.	1870.	1740.	1710.	760.	921.	833.	635.	597.	712.	780.
count-olsr_hello	13.5	12.4	13.7	10.5	11.1	11.3	12.0	n/a						
count-olsr_tc	159.	79.6	46.0	32.1	32.6	28.4	27.3	n/a						
count-olsr_total	158.	77.7	47.9	35.8	32.8	29.0	30.0	n/a						
count-received	4700.	2710.	1800.	1740.	1460.	1370.	1250.	4030.	2720.	2570.	2410.	1660.	1860.	1310.
count-sent	2.90	4.23	3.04	3.31	3.60	2.93	3.65	3.50	3.73	3.02	3.46	3.81	3.20	3.32
rate-IFQ_—_AODV	n/a							.000	2.61	2.48	1.57	1.65	1.86	1.52
rate-IFQ_—_ARP	.187	.716	1.14	1.10	1.29	.966	.885	.193	1.29	2.45	1.89	2.16	2.36	1.85
rate-IFQ_—_OLSR	1.63	1.65	1.16	1.08	1.16	.922	.869	n/a						
rate-IFQ_—_cbr	12.4	9.48	6.74	6.01	6.57	5.61	4.83	18.7	9.17	8.92	6.55	7.71	6.21	5.72
rate-IFQ_ARP_AODV	n/a							.066	.935	1.69	1.04	1.13	1.46	1.23
rate-IFQ_ARP_cbr	.563	1.34	1.89	1.86	2.71	2.25	2.24	.187	.914	1.60	1.25	1.50	1.41	1.10
rate-IFQ_END_AODV	n/a							.000	.00973	.0167	.0213	.0243	.0283	.0359
rate-IFQ_END_cbr	.000	.00848	.0128	.0188	.0248	.0324	.0337	.000	.00141	.00354	.00365	.00565	.0066	.00651
rate-MAC_—_AODV	n/a							16.7	24.3	18.7	14.3	12.4	12.5	11.1
rate-MAC_—_ARP	.593	.891	1.01	.801	1.06	.866	.942	.451	2.47	3.28	2.26	2.30	2.46	2.01
rate-MAC_—_OLSR	28.3	10.8	4.61	4.35	4.23	3.64	3.22	n/a						
rate-MAC_—_cbr	7.68	2.84	3.35	2.81	3.34	2.61	3.35	4.46	1.84	2.23	2.37	2.42	2.89	2.76
rate-MAC_BSY_MAC	.247	.0714	.050	.0385	.0418	.0272	.0273	.182	.0844	.0628	.0664	.0637	.0445	.0423
rate-MAC_COL_MAC	77.3	19.5	12.4	7.33	6.82	4.73	3.59	42.2	19.4	15.5	9.68	8.79	9.52	8.00
rate-MAC_RET_MAC	2.03	1.90	3.00	2.69	3.17	2.61	3.31	2.29	1.21	1.66	2.14	2.64	3.18	2.87
rate-RTR_CBK_cbr	n/a							2.30	1.29	2.07	2.40	2.75	2.83	2.87
rate-RTR_IFQ_AODV	n/a							.000	.0196	.00454	.0154	.00542	.00566	.0285
rate-RTR_IFQ_cbr	n/a							11.7	4.33	3.25	3.09	2.08	3.06	2.08
rate-RTR_LOOP_cbr	.707	.557	.462	.420	.471	.572	.365	.965	1.62	1.50	1.46	1.12	1.35	.964
rate-RTR_NRTE_AODV	n/a							.0262	.595	.644	.438	.396	.471	.540
rate-RTR_NRTE_cbr	24.2	9.83	8.39	6.52	7.44	7.01	6.81	3.03	3.40	3.37	2.53	2.46	2.83	3.12
rate-RTR_TOUT_AODV	n/a							.000	.000	.000	.000	.00231	.00849	.000
rate-RTR_TOUT_cbr	n/a							1.39	.286	.0899	.0466	.135	.067	.0402
rate-RTR_TTL_AODV	n/a							.902	2.32	1.84	1.19	1.42	1.31	1.39
rate-RTR_TTL_cbr	.287	.294	.375	.315	.379	.349	.298	.392	.808	.710	.553	.832	.574	.588
rate-bandwidth_byterate_AODV	n/a							735.	2490.	1930.	1160.	1130.	1380.	1410.
rate-bandwidth_byterate_ARP	48.9	81.2	83.4	96.2	139.	133.	124.	28.5	134.	218.	159.	179.	169.	125.
rate-bandwidth_byterate_MAC	16900.	4150.	3780.	2400.	3140.	2590.	2570.	9470.	5610.	5780.	4230.	3660.	4470.	3360.
rate-bandwidth_byterate_OLSR	1260.	572.	343.	305.	325.	292.	275.	n/a						
rate-bandwidth_byterate_cbr	13100.	3710.	3130.	2120.	2440.	2460.	1890.	7640.	6100.	5490.	4240.	4020.	3200.	2890.
rate-bandwidth_packetrate_AODV	n/a							7.17	25.3	19.6	11.8	11.4	13.9	14.3
rate-bandwidth_packetrate_ARP	.611	1.02	1.04	1.20	1.73	1.66	1.55	.356	1.67	2.72	1.99	2.23	2.12	1.57
rate-bandwidth_packetrate_MAC	410.	101.	91.3	58.2	75.5	63.2	61.5	230.	138.	141.	104.	90.2	108.	81.8
rate-bandwidth_packetrate_OLSR	17.5	7.94	4.77	4.24	4.51	4.06	3.83	n/a						
rate-bandwidth_packetrate_cbr	96.7	27.3	23.0	15.6	17.9	18.1	13.9	56.2	44.9	40.4	31.2	29.6	23.6	21.2
time-avgpacketdelay	.301	.135	.0929	.0848	.113	.0966	.0919	.297	.119	.0878	.0867	.104	.0846	.0615

Table B.24.: *Mobility Test - Standard Deviations.*

Density Test, Variable Traffic - Means												
	OLSR						AODV					
	10.0	20.0	50.0	75.0	100.0	125.0	10.0	20.0	50.0	75.0	100.0	125.0
Number of simulations	31.0	30.0	32.0	35.0	32.0	32.0	31.0	30.0	32.0	32.0	36.0	32.0
count-noroutedrop	9050.	13800.	12400.	19600.	31800.	45700.	59.2	706.	8740.	14000.	16600.	17500.
count-olsr_hello	1250.	2500.	6250.	9380.	12500.	15600.	n/a					
count-olsr_tc	117.	489.	1910.	3100.	4220.	5350.	n/a					
count-olsr_total	1370.	2990.	8160.	12500.	16700.	21000.	n/a					
count-received	3140.	8940.	25200.	29200.	31300.	32000.	3400.	10900.	25100.	20100.	14400.	11900.
count-sent	12500.	25000.	62600.	92700.	125000.	155000.	12500.	25000.	62600.	92700.	125000.	155000.
rate-IFQ_—_AODV	n/a						.000	.179	7.58	58.8	224.	453.
rate-IFQ_—_ARP	.000	.146	3.44	8.20	13.3	18.5	.100	.100	4.26	28.0	96.9	171.
rate-IFQ_—_OLSR	.004	.149	9.24	26.9	49.7	79.4	n/a					
rate-IFQ_—_cbr	.0988	2.31	62.7	117.	169.	212.	.175	2.37	88.3	195.	299.	384.
rate-IFQ_ARP_AODV	n/a						.00867	.0341	3.15	27.1	97.2	171.
rate-IFQ_ARP_cbr	.0597	.441	8.04	18.8	30.9	42.9	.102	.182	3.33	12.8	29.1	42.6
rate-IFQ_END_AODV	n/a						.00533	.00622	.0221	.128	.930	2.58
rate-IFQ_END_cbr	.004	.0068	.0353	.0882	.173	.285	.004	.004	.00545	.0096	.0296	.0527
rate-MAC_—_AODV	n/a						.0726	2.27	141.	653.	1850.	3710.
rate-MAC_—_ARP	.000	.0384	3.81	17.1	43.9	86.1	.016	.0865	8.41	60.5	177.	297.
rate-MAC_—_OLSR	.0154	.540	60.3	244.	569.	1130.	n/a					
rate-MAC_—_cbr	1.37	6.57	36.5	55.3	70.2	81.7	1.19	5.96	24.8	24.8	21.0	17.8
rate-MAC_BSY_MAC	.000	.020	.479	.789	.923	1.00	.000	.0419	.674	.727	.634	.579
rate-MAC_COL_MAC	.117	2.95	131.	250.	334.	402.	.184	8.08	220.	380.	521.	616.
rate-MAC_RET_MAC	1.37	6.36	26.2	36.3	45.7	52.8	1.19	5.62	15.0	20.7	25.5	28.6
rate-RTR_CBK_cbr	n/a						1.24	5.62	14.1	21.4	35.7	48.2
rate-RTR_IFQ_AODV	n/a						.000	.000	.00889	.0516	.185	.327
rate-RTR_IFQ_cbr	n/a						32.1	41.3	6.33	9.65	27.4	47.8
rate-RTR_LOOP_cbr	.000	.209	1.84	2.16	2.18	2.00	1.46	1.35	3.11	3.33	3.52	2.95
rate-RTR_NRTE_AODV	n/a						.000	.008	1.40	9.21	18.5	23.8
rate-RTR_NRTE_cbr	36.2	55.2	49.4	78.2	127.	182.	.408	2.82	33.6	46.8	47.8	46.1
rate-RTR_TOUT_AODV	n/a						.000	.000	.052	.00737	.0315	.0795
rate-RTR_TOUT_cbr	n/a						1.34	1.96	.246	.172	.378	.870
rate-RTR_TTL_AODV	n/a						.123	.711	13.3	44.9	99.3	168.
rate-RTR_TTL_cbr	.032	.198	1.63	1.44	1.11	.910	1.19	1.19	1.25	.856	.407	.215
rate-bandwidth_byterate_AODV	n/a						1280.	3300.	21500.	59000.	103000.	140000.
rate-bandwidth_byterate_ARP	8.38	57.6	625.	1350.	2110.	2820.	20.1	69.8	823.	2710.	4970.	6300.
rate-bandwidth_byterate_MAC	2840.	14900.	88500.	120000.	139000.	154000.	4190.	22200.	111000.	152000.	179000.	191000.
rate-bandwidth_byterate_OLSR	433.	1390.	7800.	14200.	20300.	27000.	n/a					
rate-bandwidth_byterate_cbr	2640.	13000.	69300.	88200.	97100.	105000.	4190.	21000.	78000.	75900.	64000.	54100.
rate-bandwidth_packetrate_AODV	n/a						13.1	33.4	217.	594.	1030.	1400.
rate-bandwidth_packetrate_ARP	.105	.720	7.81	16.9	26.4	35.3	.252	.873	10.3	33.8	62.1	78.8
rate-bandwidth_packetrate_MAC	69.9	365.	2150.	2910.	3360.	3710.	104.	547.	2710.	3680.	4310.	4600.
rate-bandwidth_packetrate_OLSR	6.02	19.4	108.	197.	282.	375.	n/a					
rate-bandwidth_packetrate_cbr	19.4	95.8	509.	648.	714.	769.	30.8	155.	574.	558.	471.	397.
time-avgpacketdelay	.00542	.0633	.616	1.12	1.53	1.95	.375	.529	.693	.916	1.03	1.15

Table B.25.: *Density Test, Variable Traffic - Means.*

Density Test, Variable Traffic - Standard Deviations												
	OLSR						AODV					
	10.0	20.0	50.0	75.0	100.0	125.0	10.0	20.0	50.0	75.0	100.0	125.0
Number of simulations	31.0	30.0	32.0	35.0	32.0	32.0	31.0	30.0	32.0	32.0	36.0	32.0
count-noroutedrop	1450.	2840.	2260.	2150.	2150.	2740.	143.	459.	753.	1090.	1110.	1070.
count-olsr_hello	4.91	6.47	9.95	14.5	16.1	14.2	n/a					
count-olsr_tc	37.8	64.6	72.5	60.2	75.8	88.4	n/a					
count-olsr_total	38.7	62.4	73.3	62.2	80.3	92.1	n/a					
count-received	1320.	2010.	2640.	2200.	2020.	3000.	1730.	2450.	2360.	2750.	1470.	2110.
count-sent	1.65	2.32	2.69	4.39	5.13	4.70	1.43	2.45	3.31	3.69	4.51	4.85
rate-IFQ_—_AODV	n/a						.000	.231	1.96	13.9	26.4	47.8
rate-IFQ_—_ARP	.000	.127	.911	1.32	1.21	2.22	.000	.183	1.27	6.92	16.2	23.0
rate-IFQ_—_OLSR	.000	.121	1.63	2.99	4.37	5.79	n/a					
rate-IFQ_—_cbr	.134	1.78	7.40	5.73	7.91	8.08	.225	1.43	8.20	9.70	11.0	14.5
rate-IFQ_ARP_AODV	n/a						.00817	.028	.953	7.11	16.0	23.7
rate-IFQ_ARP_cbr	.032	.355	1.80	2.52	2.02	4.23	.078	.162	.945	2.59	4.40	4.71
rate-IFQ_END_AODV	n/a						.00231	.00211	.00985	.0394	.262	.581
rate-IFQ_END_cbr	.000	.00369	.0133	.0241	.0289	.0382	.000	.000	.0027	.00447	.0137	.0176
rate-MAC_—_AODV	n/a						.060	.913	19.7	75.6	139.	268.
rate-MAC_—_ARP	.000	.0536	1.13	3.17	5.25	10.9	.0191	.0666	2.34	13.6	24.0	29.6
rate-MAC_—_OLSR	.0136	.328	11.5	28.6	57.3	84.9	n/a					
rate-MAC_—_cbr	.728	2.81	2.70	2.85	2.45	3.45	.758	1.67	2.07	2.57	1.58	1.81
rate-MAC_BSY_MAC	.000	.0169	.0705	.0918	.0938	.076	.000	.0318	.0968	.0712	.0627	.0638
rate-MAC_COL_MAC	.324	2.85	18.6	17.6	17.4	22.4	.343	5.22	21.3	17.6	17.4	14.3
rate-MAC_RET_MAC	.725	2.69	2.31	2.17	2.35	2.37	.751	1.59	1.51	1.24	1.16	1.35
rate-RTR_CBK_cbr	n/a						.735	1.56	1.70	2.87	4.60	4.91
rate-RTR_IFQ_AODV	n/a						.000	.000	.00558	.0294	.0522	.0832
rate-RTR_IFQ_cbr	n/a						7.39	11.5	3.50	3.30	5.54	7.26
rate-RTR_LOOP_cbr	.000	.214	.462	.523	.544	.434	1.86	1.05	1.72	1.10	1.37	1.03
rate-RTR_NRTE_AODV	n/a						.000	.000	.558	1.44	1.59	1.53
rate-RTR_NRTE_cbr	5.78	11.4	9.08	8.52	8.59	10.9	.708	1.84	2.98	4.89	4.09	4.21
rate-RTR_TOUT_AODV	n/a						.000	.000	.000	.00406	.0194	.0304
rate-RTR_TOUT_cbr	n/a						.617	.997	.314	.116	.289	.477
rate-RTR_TTL_AODV	n/a						.0454	.116	1.85	4.14	5.13	8.54
rate-RTR_TTL_cbr	.000	.172	.552	.351	.288	.250	1.22	1.29	.889	.684	.310	.272
rate-bandwidth_byterate_AODV	n/a						58.0	210.	1780.	3510.	2850.	2980.
rate-bandwidth_byterate_ARP	4.85	26.4	125.	144.	120.	203.	15.7	27.1	148.	358.	407.	420.
rate-bandwidth_byterate_MAC	1660.	5580.	4900.	5450.	4430.	4680.	3200.	6490.	6120.	6670.	6260.	5390.
rate-bandwidth_byterate_OLSR	44.7	210.	564.	681.	1030.	1190.	n/a					
rate-bandwidth_byterate_cbr	1580.	4640.	4290.	4840.	4040.	4580.	3340.	6170.	6290.	7200.	4530.	4490.
rate-bandwidth_packetrate_AODV	n/a						.561	2.04	18.1	35.6	29.2	29.9
rate-bandwidth_packetrate_ARP	.0606	.329	1.56	1.80	1.50	2.54	.196	.339	1.85	4.48	5.09	5.26
rate-bandwidth_packetrate_MAC	41.0	136.	120.	133.	108.	115.	79.4	160.	151.	164.	153.	132.
rate-bandwidth_packetrate_OLSR	.620	2.92	7.83	9.45	14.3	16.5	n/a					
rate-bandwidth_packetrate_cbr	11.7	34.1	31.5	35.6	29.7	33.7	24.5	45.4	46.2	53.0	33.3	33.0
time-avgpacketdelay	.0108	.0423	.101	.146	.177	.238	.358	.228	.124	.130	.128	.175

Table B.26.: *Density Test, Variable Traffic - Standard Deviations.*

Density Test, Constant Traffic - Means												
	OLSR						AODV					
	10.0	20.0	50.0	75.0	100.0	125.0	10.0	20.0	50.0	75.0	100.0	125.0
Number of simulations	30.0	35.0	32.0	32.0	32.0	32.0	31.0	34.0	32.0	32.0	32.0	32.0
count-noroutedrop	45800.	35500.	12000.	5730.	4130.	4160.	483.	2570.	8780.	10100.	11100.	11600.
count-olsr_hello	1250.	2500.	6250.	9380.	12500.	15600.	n/a					
count-olsr_tc	123.	508.	1940.	3120.	4290.	5500.	n/a					
count-olsr_total	1370.	3010.	8190.	12500.	16800.	21100.	n/a					
count-received	14800.	20100.	25300.	25800.	26000.	25900.	13700.	22500.	24900.	23600.	20000.	15900.
count-sent	62600.	62600.	62600.	62600.	62600.	62600.	62600.	62600.	62600.	62600.	62600.	62600.
rate-IFQ_—_AODV	n/a						.440	1.44	7.60	21.5	81.1	230.
rate-IFQ_—_ARP	.060	.298	3.49	5.60	6.57	6.88	.120	.776	3.84	9.79	34.4	81.6
rate-IFQ_—_OLSR	.0836	.641	9.97	24.4	43.2	62.3	n/a					
rate-IFQ_—_cbr	3.69	15.3	63.1	74.8	74.6	70.5	5.50	30.9	87.6	93.9	106.	115.
rate-IFQ_ARP_AODV	n/a						.016	.130	2.89	11.7	47.2	110.
rate-IFQ_ARP_cbr	.104	.764	8.12	13.5	16.7	19.1	.239	.767	3.10	4.90	7.74	10.8
rate-IFQ_END_AODV	n/a						.000	.00667	.0229	.064	.433	1.80
rate-IFQ_END_cbr	.004	.0058	.0335	.082	.131	.197	.000	.004	.00444	.00655	.0102	.0144
rate-MAC_—_AODV	n/a						.442	7.92	138.	463.	1230.	2620.
rate-MAC_—_ARP	.00857	.0909	3.55	11.0	21.2	36.9	.0263	.327	8.01	30.5	102.	214.
rate-MAC_—_OLSR	.0488	1.31	63.9	246.	644.	1370.	n/a					
rate-MAC_—_cbr	4.72	12.7	37.3	47.0	51.9	55.6	3.48	11.4	24.9	24.3	20.4	17.1
rate-MAC_BSY_MAC	.0275	.0745	.488	.547	.525	.469	.0337	.270	.665	.661	.593	.541
rate-MAC_COL_MAC	1.52	15.0	137.	175.	198.	218.	5.70	57.9	221.	288.	388.	484.
rate-MAC_RET_MAC	4.59	11.5	26.6	33.6	36.7	39.3	3.13	7.48	14.4	19.3	24.0	28.0
rate-RTR_CBK_cbr	n/a						3.32	7.85	13.5	15.4	16.4	17.9
rate-RTR_IFQ_AODV	n/a						.004	.008	.0145	.0143	.0324	.0919
rate-RTR_IFQ_cbr	n/a						177.	100.	7.14	3.62	6.67	13.7
rate-RTR_LOOP_cbr	.346	.472	1.74	1.22	.621	.511	4.45	4.87	3.89	1.96	1.26	1.15
rate-RTR_NRTE_AODV	n/a						.000	.0333	1.20	4.90	12.3	18.9
rate-RTR_NRTE_cbr	183.	142.	47.7	22.8	16.5	16.6	1.93	10.3	33.9	35.6	32.0	27.6
rate-RTR_TOUT_AODV	n/a						.000	.004	.004	.0048	.013	.0536
rate-RTR_TOUT_cbr	n/a						.753	2.22	.343	.064	.145	.439
rate-RTR_TTL_AODV	n/a						.279	1.30	13.2	35.9	76.0	133.
rate-RTR_TTL_cbr	.124	.314	1.70	1.32	1.05	.743	2.19	3.14	1.37	.488	.212	.111
rate-bandwidth_byterate_AODV	n/a						1990.	5100.	21300.	42600.	75000.	109000.
rate-bandwidth_byterate_ARP	15.4	85.8	633.	1070.	1400.	1700.	59.8	163.	780.	1580.	3130.	4620.
rate-bandwidth_byterate_MAC	13600.	34600.	89700.	98000.	99000.	96500.	21400.	58200.	113000.	127000.	145000.	158000.
rate-bandwidth_byterate_OLSR	433.	1320.	8090.	17200.	29000.	43300.	n/a					
rate-bandwidth_byterate_cbr	12500.	29600.	69900.	71600.	69600.	65500.	21900.	53600.	80300.	72400.	60000.	48600.
rate-bandwidth_packetrate_AODV	n/a						19.9	51.1	214.	430.	755.	1090.
rate-bandwidth_packetrate_ARP	.193	1.07	7.91	13.4	17.5	21.2	.748	2.03	9.75	19.8	39.1	57.7
rate-bandwidth_packetrate_MAC	334.	845.	2180.	2370.	2390.	2320.	530.	1430.	2740.	3070.	3510.	3820.
rate-bandwidth_packetrate_OLSR	6.01	18.3	112.	239.	403.	601.	n/a					
rate-bandwidth_packetrate_cbr	91.6	218.	514.	527.	512.	482.	161.	394.	591.	532.	441.	358.
time-avgpacketdelay	.0401	.138	.643	.885	1.02	1.13	.131	.443	.684	.806	.933	.945

Table B.27.: *Density Test, Constant Traffic - Means.*

Density Test, Constant Traffic - Standard Deviations												
	OLSR						AODV					
	10.0	20.0	50.0	75.0	100.0	125.0	10.0	20.0	50.0	75.0	100.0	125.0
Number of simulations	30.0	35.0	32.0	32.0	32.0	32.0	31.0	34.0	32.0	32.0	32.0	32.0
count-noroutedrop	5160.	5660.	2510.	1200.	791.	732.	493.	1310.	1000.	973.	904.	1080.
count-olsr_hello	3.90	7.99	11.0	11.4	13.7	18.4	n/a					
count-olsr_tc	33.9	49.7	67.0	75.4	88.4	83.9	n/a					
count-olsr_total	33.2	51.9	69.1	74.3	91.0	86.4	n/a					
count-received	4070.	3650.	2280.	1990.	1900.	1900.	4650.	3610.	2190.	3670.	5150.	4270.
count-sent	3.27	3.53	4.11	3.47	3.89	3.72	4.00	4.14	3.60	3.65	3.46	4.08
rate-IFQ_—_AODV	n/a						.264	1.15	2.99	13.5	36.3	52.9
rate-IFQ_—_ARP	.000	.389	.794	1.26	1.12	.662	.221	.738	1.83	5.77	16.1	22.1
rate-IFQ_—_OLSR	.0792	.337	1.52	2.97	4.21	4.91	n/a					
rate-IFQ_—_cbr	4.18	8.41	6.41	4.36	5.37	4.67	8.42	18.8	8.93	11.7	18.6	12.5
rate-IFQ_ARP_AODV	n/a						.022	.0928	1.24	6.17	20.8	27.7
rate-IFQ_ARP_cbr	.0947	.696	1.47	2.10	2.08	1.21	.147	.585	1.24	2.11	2.49	2.91
rate-IFQ_END_AODV	n/a						.000	.00462	.0135	.023	.252	.451
rate-IFQ_END_cbr	.000	.00204	.0131	.0192	.0292	.0305	.000	.000	.00133	.0037	.00601	.010
rate-MAC_—_AODV	n/a						.393	2.99	23.2	80.3	186.	281.
rate-MAC_—_ARP	.00877	.102	.918	1.98	3.13	5.66	.0286	.282	3.02	12.0	31.0	43.8
rate-MAC_—_OLSR	.0355	.668	10.7	37.7	66.4	174.	n/a					
rate-MAC_—_cbr	2.07	2.71	2.76	2.91	2.68	2.92	1.43	3.00	2.01	1.71	3.01	2.84
rate-MAC_BSY_MAC	.0536	.0492	.0785	.0771	.0718	.0572	.0362	.116	.0738	.0811	.0737	.0687
rate-MAC_COL_MAC	1.84	9.93	18.1	25.0	19.9	17.1	8.64	28.3	25.4	24.9	35.7	39.1
rate-MAC_RET_MAC	2.01	2.30	2.37	2.76	1.92	1.92	1.07	1.43	1.01	1.24	2.30	1.89
rate-RTR_CBK_cbr	n/a						1.08	1.53	1.52	1.74	2.55	2.28
rate-RTR_IFQ_AODV	n/a						.000	.00566	.0172	.0139	.032	.0381
rate-RTR_IFQ_cbr	n/a						27.9	32.3	5.22	1.47	2.16	3.67
rate-RTR_LOOP_cbr	.0456	.349	.471	.319	.235	.175	3.49	2.37	1.97	1.01	.592	.770
rate-RTR_NRTE_AODV	n/a						.000	.0273	.494	2.01	3.85	3.43
rate-RTR_NRTE_cbr	20.7	22.7	9.99	4.79	3.15	2.93	1.97	5.24	3.82	3.70	3.44	2.98
rate-RTR_TOUT_AODV	n/a						.000	.000	.000	.00179	.00768	.0371
rate-RTR_TOUT_cbr	n/a						.363	1.25	.180	.032	.148	.297
rate-RTR_TTL_AODV	n/a						.0764	.178	2.35	4.61	9.22	12.1
rate-RTR_TTL_cbr	.0978	.194	.316	.372	.249	.213	1.92	1.39	1.08	.333	.203	.0997
rate-bandwidth_byterate_AODV	n/a						127.	391.	2140.	5700.	11500.	11900.
rate-bandwidth_byterate_ARP	6.53	39.1	87.0	81.6	120.	95.7	26.0	73.2	176.	394.	665.	742.
rate-bandwidth_byterate_MAC	5420.	8750.	4220.	4770.	4860.	3330.	12100.	14500.	6540.	6000.	7790.	8560.
rate-bandwidth_byterate_OLSR	28.7	119.	629.	1030.	1540.	1890.	n/a					
rate-bandwidth_byterate_cbr	4850.	7620.	3640.	3990.	4080.	2590.	12700.	12500.	6540.	7650.	8070.	7220.
rate-bandwidth_packetrate_AODV	n/a						1.25	4.00	21.6	57.7	116.	120.
rate-bandwidth_packetrate_ARP	.0816	.489	1.09	1.02	1.51	1.20	.325	.915	2.20	4.92	8.31	9.27
rate-bandwidth_packetrate_MAC	133.	214.	103.	116.	118.	80.4	301.	354.	160.	146.	186.	204.
rate-bandwidth_packetrate_OLSR	.399	1.66	8.73	14.3	21.3	26.3	n/a					
rate-bandwidth_packetrate_cbr	35.6	56.0	26.7	29.4	30.0	19.1	93.7	91.7	48.1	56.3	59.4	53.1
time-avgpacketdelay	.0351	.0592	.0914	.130	.146	.125	.0966	.166	.120	.106	.169	.161

Table B.28.: *Density Test, Constant Traffic - Standard Deviations.*

Duration Test - Means																		
	OLSR									AODV								
	1.0	5.0	10.0	20.0	40.0	80.0	120.0	190.0	240.0	1.0	5.0	10.0	20.0	40.0	80.0	120.0	190.0	240.0
simulations	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	34.0	36.0	32.0	32.0
utedrop	12300.	10500.	10400.	11500.	11900.	13700.	15800.	12500.	10300.	11300.	9780.	9270.	8780.	8980.	8510.	8240.	8430.	8380.
hello	6250.	6250.	6250.	6250.	6250.	6250.	6250.	6250.	6250.	n/a								
tc	1750.	1740.	1750.	1720.	1740.	1730.	1720.	1740.	1740.	n/a								
total	8000.	7990.	8000.	7960.	7990.	7980.	7960.	8000.	7990.	n/a								
ved	27100.	26600.	26400.	25600.	25500.	24800.	23400.	24000.	27500.	18200.	23100.	25100.	26800.	25300.	24000.	23800.	24900.	30600.
	67500.	63500.	63000.	62600.	62500.	62500.	62500.	60800.	62400.	67500.	63500.	63000.	62600.	62500.	62500.	62500.	60800.	62400.
—_AODV	n/a									40.3	14.5	9.77	6.89	6.96	8.03	8.71	5.61	3.91
—_ARP	2.54	3.28	3.67	3.55	3.49	3.39	2.90	3.18	3.24	16.0	6.22	4.36	3.22	3.56	4.53	4.71	3.21	1.57
—_OLSR	10.7	9.37	9.45	8.84	8.90	8.11	7.43	8.74	9.55	n/a								
—_cbr	69.6	64.6	64.0	62.5	62.2	59.2	58.4	59.5	60.2	121.	99.5	92.7	85.9	87.5	91.3	90.5	80.5	67.8
ARP_AODV	n/a									15.5	6.19	4.21	2.94	2.90	3.14	3.11	2.20	1.41
ARP_cbr	6.44	7.81	8.56	8.20	8.27	7.87	7.05	7.63	7.83	7.79	4.07	3.17	2.63	2.83	3.46	3.67	2.81	1.53
END_AODV	n/a									.045	.0199	.0195	.0195	.0218	.0229	.024	.0213	.0195
END_cbr	.030	.030	.033	.0314	.0345	.0336	.0303	.0303	.0374	.0114	.00612	.00467	.00489	.0044	.00533	.00492	.00567	.0044
—_AODV	n/a									287.	204.	176.	152.	138.	125.	123.	119.	117.
—_ARP	3.78	3.75	3.84	3.59	3.71	3.46	3.31	3.50	3.32	21.8	11.7	8.92	7.25	7.61	8.35	8.79	7.07	4.55
—_OLSR	63.5	59.0	60.7	57.3	57.9	53.2	51.0	55.9	61.5	n/a								
—_cbr	43.5	39.7	38.9	38.2	36.5	35.0	34.2	36.3	37.0	26.4	26.7	26.5	25.6	25.2	24.1	23.8	24.5	25.9
BSY_MAC	.523	.482	.461	.490	.450	.443	.442	.435	.481	.736	.738	.723	.700	.686	.622	.582	.662	.756
COL_MAC	141.	130.	131.	130.	126.	122.	123.	121.	127.	320.	278.	253.	231.	223.	208.	199.	209.	218.
RET_MAC	32.3	29.5	28.8	27.9	26.9	25.5	24.5	26.9	27.2	20.6	17.7	17.0	15.5	15.2	14.6	14.4	14.6	14.7
CBK_cbr	n/a									18.0	14.3	13.9	13.3	13.7	14.2	14.6	13.7	13.0
IFQ_AODV	n/a									.004	.008	.00667	.0105	.00714	.009	.0167	.0164	.006
IFQ_cbr	n/a									1.42	2.19	2.23	3.99	6.07	7.31	8.70	7.86	7.07
LOOP_cbr	1.93	1.66	1.73	1.94	1.69	1.82	1.80	1.62	1.73	3.97	2.61	2.40	2.45	2.90	3.89	4.27	4.54	3.44
NRTE_AODV	n/a									4.30	2.73	1.91	1.31	1.25	1.43	1.35	.867	.462
NRTE_cbr	49.1	42.0	41.4	45.6	47.3	54.7	62.8	49.9	41.0	40.8	36.4	35.2	33.8	34.7	32.6	31.6	32.8	33.1
TOUT_AODV	n/a									.004	.000	.004	.000	.004	.004	.005	.000	.000
TOUT_cbr	n/a									.264	.444	.359	.308	.264	.215	.215	.236	.000
TTL_AODV	n/a									25.6	23.2	20.6	17.0	14.2	11.9	10.9	10.0	9.70
TTL_cbr	1.94	1.85	1.82	1.87	1.74	1.71	1.48	1.64	1.86	3.31	2.05	1.66	1.34	1.22	1.26	1.08	1.31	1.13
idth_byterate_AODV	n/a									50600.	34200.	28900.	24000.	21700.	20000.	19200.	18400.	17400.
idth_byterate_ARP	645.	669.	683.	642.	645.	604.	571.	619.	629.	1760.	1120.	928.	782.	775.	798.	806.	724.	568.
idth_byterate_MAC	97800.	93700.	92300.	91300.	88000.	83900.	81400.	85300.	91800.	148000.	131000.	125000.	120000.	114000.	107000.	103000.	109000.	117000.
idth_byterate_OLSR	6820.	7100.	7350.	7160.	7300.	7490.	7560.	7540.	7300.	n/a								
idth_byterate_cbr	73700.	72300.	71300.	71000.	68600.	65800.	63700.	66400.	72000.	80300.	82200.	82800.	83900.	80300.	76400.	73900.	79000.	88000.
idth_packetrate_AODV	n/a									510.	344.	291.	242.	219.	202.	194.	186.	175.
idth_packetrate_ARP	8.06	8.36	8.54	8.03	8.06	7.55	7.14	7.73	7.86	22.0	14.0	11.6	9.78	9.69	9.97	10.1	9.05	7.11
idth_packetrate_MAC	2370.	2270.	2240.	2220.	2140.	2040.	1980.	2070.	2230.	3590.	3180.	3040.	2920.	2770.	2610.	2510.	2650.	2850.
idth_packetrate_OLSR	94.7	98.6	102.	99.4	101.	104.	105.	105.	101.	n/a								
idth_packetrate_cbr	542.	532.	525.	522.	504.	484.	468.	488.	530.	591.	604.	609.	617.	590.	562.	543.	581.	647.
cketdelay	.812	.648	.636	.614	.644	.607	.624	.633	.585	.815	.785	.719	.691	.698	.669	.676	.722	.680

Duration Test - Standard Deviations																		
	OLSR									AODV								
	1.0	5.0	10.0	20.0	40.0	80.0	120.0	190.0	240.0	1.0	5.0	10.0	20.0	40.0	80.0	120.0	190.0	240.0
Number of simulations	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	34.0	36.0	32.0	32.0
count-noroutedrop	2060.	1940.	1960.	2120.	2130.	2610.	2560.	3660.	2780.	700.	442.	589.	822.	906.	994.	1020.	1400.	1760.
count-olsr_hello	10.1	8.96	8.76	11.5	12.2	11.2	11.1	9.58	12.4	n/a								
count-olsr_tc	73.0	53.8	58.8	65.7	50.3	50.2	47.5	44.9	84.9	n/a								
count-olsr_total	76.0	52.9	59.7	66.7	50.4	52.8	49.5	45.8	87.3	n/a								
count-received	1110.	1270.	1650.	1870.	2330.	2700.	3240.	3640.	3290.	1790.	1570.	2270.	1790.	1920.	2380.	3550.	3550.	5190.
count-sent	26.7	14.3	11.8	6.70	3.61	.000	.000	.000	.000	32.4	16.8	9.56	5.96	4.73	.000	.000	.000	.000
rate-IFQ_—_AODV	n/a									11.5	4.03	2.80	1.58	2.11	2.31	2.77	2.08	2.04
rate-IFQ_—_ARP	.605	.861	.812	1.15	.672	.857	.845	.892	.938	3.93	2.20	1.46	1.08	1.35	1.38	1.42	1.42	.907
rate-IFQ_—_OLSR	1.94	1.38	1.37	1.72	1.65	1.21	1.20	1.17	1.81	n/a								
rate-IFQ_—_cbr	6.50	6.07	5.34	7.38	6.64	7.12	6.59	6.43	9.14	6.93	5.34	6.29	5.74	6.74	7.73	11.6	9.99	14.1
rate-IFQ_ARP_AODV	n/a									3.94	1.66	.940	.645	.786	1.01	.905	.904	.625
rate-IFQ_ARP_cbr	1.26	1.59	1.32	2.03	1.29	1.54	1.49	1.53	1.62	1.56	1.26	.998	.802	.963	.960	1.01	1.09	.713
rate-IFQ_END_AODV	n/a									.0203	.0122	.0112	.00791	.0112	.00976	.0111	.0115	.00818
rate-IFQ_END_cbr	.0132	.0127	.0145	.0105	.0152	.0151	.0111	.0114	.0141	.00662	.00287	.00153	.00176	.00126	.00289	.0024	.00267	.00126
rate-MAC_—_AODV	n/a									40.3	25.1	19.6	13.9	21.9	16.7	19.2	22.6	25.9
rate-MAC_—_ARP	.959	1.02	.907	1.05	.777	.921	.986	.814	.866	5.41	3.27	2.24	1.60	2.43	2.34	2.55	2.80	1.64
rate-MAC_—_OLSR	10.9	9.42	10.2	9.65	11.1	8.81	8.88	8.26	11.8	n/a								
rate-MAC_—_cbr	1.89	2.30	1.68	2.34	2.93	2.38	2.95	3.00	3.92	1.38	1.61	1.77	1.62	1.83	2.03	2.49	2.43	3.63
rate-MAC_BSY_MAC	.0884	.0735	.0695	.0527	.0671	.0833	.0762	.0631	.0978	.0678	.0572	.0657	.0882	.0888	.0898	.0944	.0765	.103
rate-MAC_COL_MAC	15.6	14.3	19.1	16.5	22.0	18.8	19.7	18.4	26.3	14.0	17.6	15.7	19.4	16.5	22.0	20.7	24.4	40.4
rate-MAC_RET_MAC	1.68	2.44	1.67	2.37	2.21	1.95	2.46	2.21	2.51	1.28	1.51	1.27	1.20	1.35	1.19	1.30	1.28	1.69
rate-RTR_CBK_cbr	n/a									1.50	1.62	1.60	1.11	1.62	1.63	1.80	1.36	1.54
rate-RTR_IFQ_AODV	n/a									.000	.0052	.00413	.00601	.0039	.00516	.0274	.0177	.00219
rate-RTR_IFQ_cbr	n/a									1.47	2.09	1.21	2.81	3.95	4.59	5.38	4.47	4.14
rate-RTR_LOOP_cbr	.503	.406	.423	.684	.499	.496	.608	.527	.649	.765	.644	.783	.719	1.72	1.60	2.40	3.04	2.61
rate-RTR_NRTE_AODV	n/a									.943	.701	.428	.341	.379	.492	.498	.410	.335
rate-RTR_NRTE_cbr	8.21	7.77	7.88	8.44	8.52	10.4	10.2	14.6	11.1	3.23	1.79	2.18	3.20	3.60	3.93	4.02	5.51	6.82
rate-RTR_TOUT_AODV	n/a									.000	.000	.000	.000	.000	.000	.002	.000	.000
rate-RTR_TOUT_cbr	n/a									.443	.485	.380	.195	.127	.0922	.0768	.000	.000
rate-RTR_TTL_AODV	n/a									2.25	1.69	1.63	1.43	1.61	1.90	1.87	2.40	2.39
rate-RTR_TTL_cbr	.365	.323	.357	.407	.425	.452	.442	.461	.468	1.01	.684	.785	.768	.655	.897	1.02	1.03	1.18
rate-bandwidth_byterate_AODV	n/a									2540.	1910.	1560.	1310.	1270.	1580.	1740.	2080.	3030.
rate-bandwidth_byterate_ARP	85.5	105.	76.3	110.	81.5	76.4	78.8	89.5	74.8	259.	199.	149.	116.	140.	114.	125.	166.	141.
rate-bandwidth_byterate_MAC	4290.	3780.	4520.	3480.	4470.	4410.	4640.	4380.	6760.	6160.	4100.	4050.	6360.	5880.	7360.	7450.	6790.	8690.
rate-bandwidth_byterate_OLSR	540.	461.	452.	516.	479.	417.	517.	440.	641.	n/a								
rate-bandwidth_byterate_cbr	3960.	3510.	4080.	3350.	3860.	3920.	3770.	3560.	5330.	8350.	5570.	3990.	6340.	5790.	6540.	6960.	6690.	5810.
rate-bandwidth_packetrate_AODV	n/a									25.0	19.4	15.8	13.4	12.8	16.0	17.6	20.9	30.4
rate-bandwidth_packetrate_ARP	1.07	1.31	.953	1.38	1.02	.955	.985	1.12	.935	3.23	2.49	1.87	1.45	1.75	1.43	1.56	2.08	1.76
rate-bandwidth_packetrate_MAC	105.	92.5	111.	85.2	109.	108.	112.	106.	164.	153.	102.	99.4	157.	145.	180.	183.	166.	210.
rate-bandwidth_packetrate_OLSR	7.50	6.40	6.27	7.16	6.65	5.79	7.19	6.11	8.90	n/a								
rate-bandwidth_packetrate_cbr	29.1	25.8	30.0	24.6	28.4	28.9	27.7	26.2	39.2	61.4	41.0	29.4	46.6	42.6	48.1	51.2	49.2	42.8
time-avgpacketdelay	.110	.0844	.0971	.111	.132	.102	.0977	.0986	.117	.101	.117	.0783	.0899	.0842	.112	.120	.127	.156

Table B.30.: *Duration Test - Standard Deviations.*

Bulk Transfer Test - Means														
	OLSR							AODV						
	6.0	8.0	10.0	12.0	14.0	16.0	18.0	6.0	8.0	10.0	12.0	14.0	16.0	18.0
Number of simulations	32.0	35.0	36.0	32.0	32.0	33.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0
count-noroutedrop	1180.	2050.	3440.	4420.	6700.	7400.	9910.	2430.	3330.	3870.	4370.	4800.	5190.	5490.
count-olsr_hello	6250.	6250.	6250.	6170.	6240.	6250.	6250.	n/a						
count-olsr_tc	1750.	1740.	1780.	1730.	1770.	1750.	1780.	n/a						
count-olsr_total	8000.	7990.	8030.	7900.	8020.	8000.	8030.	n/a						
count-received	27800.	30700.	34400.	36200.	40600.	41800.	44400.	21400.	23300.	24300.	25100.	25700.	26200.	26600.
count-sent	33900.	38500.	44900.	47900.	55300.	57700.	63700.	27500.	31700.	34900.	38000.	40900.	43800.	46400.
rate-IFQ_—_AODV	n/a							16.3	25.0	40.1	47.4	58.4	67.2	75.8
rate-IFQ_—_ARP	.127	.277	.360	.355	.465	.625	.657	5.17	7.43	11.1	14.7	17.4	19.9	22.4
rate-IFQ_—_OLSR	3.11	4.85	5.62	4.95	6.16	6.93	7.53	n/a						
rate-IFQ_—_ack	.751	1.21	1.61	1.67	2.04	2.33	2.77	1.60	2.40	3.08	3.63	4.12	4.73	5.02
rate-IFQ_—_tcp	2.04	3.72	5.44	5.77	7.51	9.19	10.4	5.60	9.78	15.4	20.4	26.2	31.9	37.9
rate-IFQ_ARP_AODV	n/a							10.5	14.0	19.1	23.5	26.0	28.4	31.0
rate-IFQ_ARP_ack	.348	.431	.503	.501	.543	.580	.607	.115	.141	.152	.169	.177	.193	.188
rate-IFQ_ARP_tcp	.791	1.16	1.48	1.47	1.74	2.07	2.20	.615	1.01	1.62	2.27	2.93	3.65	4.33
rate-IFQ_END_AODV	n/a							.0494	.0569	.0706	.0831	.0898	.0946	.103
rate-IFQ_END_ack	.0114	.00821	.0121	.0133	.0112	.011	.0103	.00431	.00467	.004	.00618	.00444	.00444	.00533
rate-IFQ_END_tcp	.0331	.0368	.0412	.0421	.0538	.0616	.0594	.0128	.0165	.0216	.0251	.0314	.0388	.0384
rate-MAC_—_AODV	n/a							245.	269.	313.	329.	350.	364.	377.
rate-MAC_—_ARP	2.51	3.09	3.55	3.25	3.80	4.15	4.55	12.0	14.8	19.5	22.0	23.1	24.6	26.7
rate-MAC_—_OLSR	114.	113.	110.	96.2	96.5	94.7	95.0	n/a						
rate-MAC_—_ack	5.24	5.68	5.70	5.65	5.70	5.76	5.81	2.90	2.69	2.32	2.17	1.96	1.90	1.73
rate-MAC_—_tcp	21.0	24.6	26.0	27.1	27.4	28.7	29.1	18.0	20.5	21.0	22.4	23.0	24.0	24.3
rate-MAC_BSY_MAC	.525	.672	.809	.835	.913	.929	.977	.632	.747	.750	.791	.801	.878	.880
rate-MAC_COL_MAC	144.	163.	178.	184.	187.	193.	199.	219.	243.	259.	267.	281.	292.	296.
rate-MAC_RET_MAC	14.6	17.2	18.2	18.4	19.1	20.1	20.4	18.3	19.2	19.2	20.1	19.7	19.6	19.8
rate-RTR_CBK_ack	n/a							2.00	1.79	1.56	1.46	1.32	1.26	1.16
rate-RTR_CBK_tcp	n/a							6.43	7.33	8.02	9.05	9.73	10.6	11.4
rate-RTR_IFQ_tcp	n/a							.000	.000	.000	.000	.000	.068	.161
rate-RTR_LOOP_ack	.0733	.0996	.129	.129	.131	.146	.163	.0791	.0766	.066	.0639	.050	.0451	.0404
rate-RTR_LOOP_tcp	.179	.305	.426	.454	.508	.560	.611	.479	.720	.869	1.11	1.29	1.53	1.67
rate-RTR_NRTE_AODV	n/a							2.86	3.76	4.77	5.17	5.53	5.59	5.84
rate-RTR_NRTE_ack	.813	1.28	1.70	2.32	2.47	2.67	3.00	1.32	1.51	1.36	1.31	1.24	1.31	1.17
rate-RTR_NRTE_tcp	3.91	7.23	12.1	18.5	24.3	29.7	36.7	5.53	8.07	9.35	11.0	12.4	13.9	15.0
rate-RTR_TOUT_AODV	n/a							.006	.00533	.0109	.00862	.0085	.010	.00988
rate-RTR_TOUT_ack	n/a							.004	.0045	.004	.004	.005	.004	.00533
rate-RTR_TOUT_tcp	n/a							.0924	.0785	.108	.141	.143	.139	.335
rate-RTR_TTL_AODV	n/a							21.3	22.1	24.2	24.9	25.3	25.5	26.5
rate-RTR_TTL_ack	.0546	.0484	.0482	.0384	.0353	.0293	.030	.0519	.046	.0314	.0254	.0211	.0188	.0155
rate-RTR_TTL_tcp	.392	.524	.623	.625	.647	.667	.662	.345	.524	.609	.748	.793	.863	.934
rate-bandwidth_byterate_AODV	n/a							39700.	43800.	47700.	50300.	52500.	54100.	55600.
rate-bandwidth_byterate_ARP	374.	406.	433.	426.	444.	469.	483.	985.	1160.	1390.	1550.	1610.	1710.	1820.
rate-bandwidth_byterate_MAC	51600.	56300.	60000.	63900.	65400.	66700.	67900.	82100.	89600.	92700.	96600.	99200.	102000.	103000.
rate-bandwidth_byterate_OLSR	8120.	7450.	6910.	6340.	6080.	5750.	5570.	n/a						
rate-bandwidth_byterate_ack	14600.	14800.	15400.	16500.	16700.	16700.	16800.	12000.	11700.	10900.	10500.	10100.	9820.	9550.
rate-bandwidth_byterate_tcp	193000.	209000.	222000.	239000.	244000.	248000.	252000.	185000.	205000.	208000.	220000.	226000.	236000.	242000.
rate-bandwidth_packetrate_AODV	n/a							401.	441.	480.	507.	528.	544.	558.
rate-bandwidth_packetrate_ARP	4.68	5.07	5.42	5.33	5.55	5.87	6.03	12.3	14.5	17.3	19.4	20.1	21.4	22.8
rate-bandwidth_packetrate_MAC	1260.	1370.	1460.	1550.	1590.	1620.	1650.	2000.	2180.	2250.	2340.	2410.	2470.	2500.
rate-bandwidth_packetrate_OLSR	113.	104.	95.9	88.1	84.4	79.8	77.3	n/a						
rate-bandwidth_packetrate_ack	130.	133.	137.	148.	149.	149.	150.	107.	105.	97.3	93.4	90.3	87.7	85.3
rate-bandwidth_packetrate_tcp	180.	195.	207.	223.	228.	231.	235.	172.	191.	194.	205.	211.	220.	226.
time-avgpacketdelay	.493	.640	.719	.685	.751	.791	.863	.392	.418	.419	.419	.421	.438	.432

Table B.31.: *Bulk Transfer Test - Means.*

Bulk Transfer Test - Standard Deviations														
	OLSR							AODV						
	6.0	8.0	10.0	12.0	14.0	16.0	18.0	6.0	8.0	10.0	12.0	14.0	16.0	18.0
Number of simulations	32.0	35.0	36.0	32.0	32.0	33.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0
count-noroutedrop	322.	387.	451.	941.	527.	1020.	598.	165.	207.	198.	278.	259.	400.	399.
count-olsr_hello	9.41	12.7	8.80	442.	12.3	17.5	11.7	n/a						
count-olsr_tc	74.3	90.5	71.3	148.	50.7	69.6	59.0	n/a						
count-olsr_total	77.4	90.3	72.1	577.	53.3	74.0	58.9	n/a						
count-received	919.	1030.	1500.	3190.	1970.	3330.	2770.	1500.	1420.	1330.	1640.	1670.	2200.	1870.
count-sent	881.	1300.	1300.	4170.	1730.	3560.	2410.	1390.	1340.	1260.	1570.	1620.	2080.	1690.
rate-IFQ_—_AODV	n/a							6.54	6.54	8.90	12.1	12.0	14.4	15.5
rate-IFQ_—_ARP	.0642	.138	.148	.181	.155	.211	.236	1.76	2.05	2.91	3.52	4.17	4.40	4.78
rate-IFQ_—_OLSR	1.19	1.43	1.56	1.28	1.46	1.78	1.84	n/a						
rate-IFQ_—_ack	.246	.273	.384	.344	.429	.515	.537	.312	.303	.374	.325	.334	.453	.475
rate-IFQ_—_tcp	.621	.708	1.18	1.20	1.46	2.16	2.12	1.11	1.21	1.51	1.58	1.83	2.05	2.29
rate-IFQ_ARP_AODV	n/a							2.91	3.18	4.44	4.82	5.65	5.74	6.21
rate-IFQ_ARP_ack	.0961	.125	.093	.123	.0925	.0888	.0979	.0327	.028	.0353	.0262	.0303	.0537	.035
rate-IFQ_ARP_tcp	.168	.261	.251	.327	.326	.383	.442	.151	.172	.257	.406	.462	.631	.618
rate-IFQ_END_AODV	n/a							.0182	.0183	.0254	.0316	.0346	.024	.0372
rate-IFQ_END_ack	.00547	.00339	.0058	.00629	.0056	.00628	.006	.00111	.00156	9.06e-19	.00275	.00133	.00133	.00462
rate-IFQ_END_tcp	.00997	.0163	.0151	.0191	.0189	.0224	.0194	.00616	.00616	.00793	.00875	.0103	.018	.0116
rate-MAC_—_AODV	n/a							32.0	26.1	29.2	37.7	28.7	37.7	46.7
rate-MAC_—_ARP	.496	.607	.624	.705	.644	.885	.967	3.19	3.38	4.42	5.11	4.58	5.32	5.37
rate-MAC_—_OLSR	17.6	18.7	14.2	12.1	11.6	10.6	12.4	n/a						
rate-MAC_—_ack	.458	.502	.367	.344	.355	.255	.344	.294	.321	.258	.251	.208	.178	.175
rate-MAC_—_tcp	1.95	1.88	2.41	2.26	1.82	2.09	1.64	1.30	1.43	2.08	2.18	1.93	1.71	2.02
rate-MAC_BSY_MAC	.100	.130	.106	.0903	.115	.0983	.0953	.108	.101	.105	.0711	.138	.160	.127
rate-MAC_COL_MAC	14.6	10.7	14.2	10.1	15.6	9.87	13.0	12.8	11.9	13.6	12.7	12.1	14.2	14.9
rate-MAC_RET_MAC	1.07	1.11	1.27	1.04	1.01	.973	1.19	.891	1.22	1.19	1.33	.987	1.11	1.19
rate-RTR_CBK_ack	n/a							.223	.220	.158	.167	.139	.136	.111
rate-RTR_CBK_tcp	n/a							.423	.420	.552	.598	.577	.698	.781
rate-RTR_IFQ_tcp	n/a							.000	.000	.000	.000	.000	.000	.0999
rate-RTR_LOOP_ack	.0223	.0338	.0421	.0445	.0415	.0431	.0461	.0262	.0231	.0226	.023	.0174	.0209	.0167
rate-RTR_LOOP_tcp	.0487	.0712	.0843	.117	.121	.126	.112	.0889	.120	.185	.191	.218	.259	.374
rate-RTR_NRTE_AODV	n/a							.746	.755	1.03	.782	.935	.836	1.03
rate-RTR_NRTE_ack	.178	.248	.276	.285	.372	.278	.388	.257	.309	.252	.301	.277	.346	.243
rate-RTR_NRTE_tcp	1.19	1.23	1.71	1.69	2.07	2.54	2.30	.479	.643	.930	1.04	1.26	1.53	1.57
rate-RTR_TOUT_AODV	n/a							.00231	.002	.00944	.00746	.00459	.00676	.00723
rate-RTR_TOUT_ack	n/a							.000	.00141	.000	.000	.00283	.000	.00207
rate-RTR_TOUT_tcp	n/a							.132	.0887	.151	.135	.162	.195	.508
rate-RTR_TTL_AODV	n/a							1.58	1.47	1.61	1.89	1.61	1.88	2.20
rate-RTR_TTL_ack	.0221	.0185	.0206	.0174	.0189	.0176	.0188	.0254	.0166	.0139	.0102	.0128	.0118	.0117
rate-RTR_TTL_tcp	.096	.111	.125	.164	.147	.181	.133	.113	.153	.227	.264	.233	.293	.335
rate-bandwidth_byterate_AODV	n/a							1990.	1500.	1350.	1230.	1060.	1360.	1620.
rate-bandwidth_byterate_ARP	36.5	39.7	26.0	31.5	32.5	31.1	46.0	155.	155.	191.	179.	191.	213.	204.
rate-bandwidth_byterate_MAC	2990.	2370.	2850.	2200.	3780.	3650.	2910.	3660.	3190.	3260.	3500.	3310.	4130.	3620.
rate-bandwidth_byterate_OLSR	660.	561.	483.	387.	317.	213.	245.	n/a						
rate-bandwidth_byterate_ack	834.	749.	1020.	1060.	1410.	1650.	1340.	1510.	1160.	944.	1030.	777.	1060.	933.
rate-bandwidth_byterate_tcp	11400.	10700.	12000.	10200.	15800.	16300.	14300.	20100.	16900.	16800.	18300.	19100.	22200.	23800.
rate-bandwidth_packetrate_AODV	n/a							20.2	15.2	13.6	12.2	10.7	13.3	15.8
rate-bandwidth_packetrate_ARP	.456	.497	.325	.394	.406	.389	.575	1.94	1.93	2.39	2.23	2.38	2.67	2.55
rate-bandwidth_packetrate_MAC	72.3	58.0	69.6	54.2	92.8	90.6	72.4	89.7	77.6	78.8	85.0	80.9	101.	88.0
rate-bandwidth_packetrate_OLSR	9.17	7.79	6.70	5.38	4.40	2.96	3.41	n/a						
rate-bandwidth_packetrate_ack	7.45	6.69	9.11	9.43	12.6	14.7	12.0	13.5	10.4	8.43	9.23	6.93	9.49	8.33
rate-bandwidth_packetrate_tcp	10.6	9.99	11.2	9.51	14.8	15.2	13.3	18.7	15.8	15.7	17.1	17.8	20.7	22.2
time-avgpacketdelay	.0969	.101	.131	.119	.131	.148	.153	.0359	.0424	.0412	.0405	.0337	.0495	.0378

Table B.32.: *Bulk Transfer Test - Standard Deviations.*

TCP Transfer Time Test - Means		
	OLSR	AODV
Number of simulations	31.0	30.0
count-noroutedrop	17.5	13.6
count-olsr_hello	6250.	n/a
count-olsr_tc	1640.	n/a
count-olsr_total	7890.	n/a
count-received	5850.	6180.
count-sent	6120.	6370.
rate-IFQ_—_AODV	n/a	.004
rate-IFQ_—_OLSR	.0424	n/a
rate-IFQ_—_ack	.006	.012
rate-IFQ_—_tcp	.0096	.022
rate-IFQ_ARP_AODV	n/a	.135
rate-IFQ_ARP_ack	.0424	.00817
rate-IFQ_ARP_tcp	.0467	.0173
rate-IFQ_END_AODV	n/a	.0301
rate-IFQ_END_ack	.0132	.00533
rate-IFQ_END_tcp	.032	.00533
rate-MAC_—_AODV	n/a	29.9
rate-MAC_—_ARP	.204	1.07
rate-MAC_—_OLSR	28.6	n/a
rate-MAC_—_ack	.515	.491
rate-MAC_—_tcp	1.61	1.71
rate-MAC_BSY_MAC	.0178	.0352
rate-MAC_COL_MAC	14.8	19.3
rate-MAC_RET_MAC	.854	1.12
rate-RTR_CBK_ack	n/a	.278
rate-RTR_CBK_tcp	n/a	.395
rate-RTR_LOOP_ack	.00533	.004
rate-RTR_LOOP_tcp	.00646	.0128
rate-RTR_NRTE_AODV	n/a	.0123
rate-RTR_NRTE_ack	.0133	.0105
rate-RTR_NRTE_tcp	.0578	.0371
rate-RTR_TOUT_tcp	n/a	.00775
rate-RTR_TTL_AODV	n/a	7.50
rate-RTR_TTL_ack	.00655	n/a
rate-RTR_TTL_tcp	.0134	.0136
rate-bandwidth_byterate_AODV	n/a	7190.
rate-bandwidth_byterate_ARP	147.	180.
rate-bandwidth_byterate_MAC	10500.	13000.
rate-bandwidth_byterate_OLSR	9730.	n/a
rate-bandwidth_byterate_ack	4120.	4810.
rate-bandwidth_byterate_tcp	42900.	49600.
rate-bandwidth_packetrate_AODV	n/a	73.4
rate-bandwidth_packetrate_ARP	1.84	2.26
rate-bandwidth_packetrate_MAC	260.	320.
rate-bandwidth_packetrate_OLSR	135.	n/a
rate-bandwidth_packetrate_ack	36.8	42.9
rate-bandwidth_packetrate_tcp	40.0	46.3
time-avgpacketdelay	.0859	.116
time-tcptransfertime	3.38	3.56

Table B.33.: *TCP Transfer Time Test - Means.*

TCP Transfer Time Test - Standard Deviations		
	OLSR	AODV
Number of simulations	31.0	30.0
count-noroutedrop	12.6	5.30
count-olsr_hello	11.5	n/a
count-olsr_tc	62.8	n/a
count-olsr_total	64.5	n/a
count-received	191.	137.
count-sent	196.	137.
rate-IFQ_—_AODV	n/a	.000
rate-IFQ_—_OLSR	.0493	n/a
rate-IFQ_—_ack	.00231	.00566
rate-IFQ_—_tcp	.00829	.0169
rate-IFQ_ARP_AODV	n/a	.0564
rate-IFQ_ARP_ack	.0308	.00597
rate-IFQ_ARP_tcp	.0253	.00767
rate-IFQ_END_AODV	n/a	.014
rate-IFQ_END_ack	.00706	.00231
rate-IFQ_END_tcp	.0128	.002
rate-MAC_—_AODV	n/a	3.50
rate-MAC_—_ARP	.0703	.334
rate-MAC_—_OLSR	4.22	n/a
rate-MAC_—_ack	.126	.111
rate-MAC_—_tcp	.353	.312
rate-MAC_BSY_MAC	.00879	.0215
rate-MAC_COL_MAC	2.64	3.58
rate-MAC_RET_MAC	.178	.264
rate-RTR_CBK_ack	n/a	.0858
rate-RTR_CBK_tcp	n/a	.0954
rate-RTR_LOOP_ack	.00207	.000
rate-RTR_LOOP_tcp	.00555	.00912
rate-RTR_NRTE_AODV	n/a	.00697
rate-RTR_NRTE_ack	.00829	.00774
rate-RTR_NRTE_tcp	.0475	.0154
rate-RTR_TOUT_tcp	n/a	.00342
rate-RTR_TTL_AODV	n/a	.542
rate-RTR_TTL_ack	.00532	n/a
rate-RTR_TTL_tcp	.00896	.0134
rate-bandwidth_byterate_AODV	n/a	302.
rate-bandwidth_byterate_ARP	11.7	16.7
rate-bandwidth_byterate_MAC	816.	1300.
rate-bandwidth_byterate_OLSR	764.	n/a
rate-bandwidth_byterate_ack	274.	417.
rate-bandwidth_byterate_tcp	3170.	4520.
rate-bandwidth_packetrate_AODV	n/a	2.97
rate-bandwidth_packetrate_ARP	.146	.208
rate-bandwidth_packetrate_MAC	20.0	31.8
rate-bandwidth_packetrate_OLSR	10.6	n/a
rate-bandwidth_packetrate_ack	2.45	3.73
rate-bandwidth_packetrate_tcp	2.96	4.21
time-avgpacketdelay	.0163	.0117
time-tcptransfertime	1.29	1.16

Table B.34.: *TCP Transfer Time Test - Standard Deviations.*

Load Test - Means																		
	OLSR									AODV								
	0.0	5.0	10.0	15.0	20.0	25.0	50.0	75.0	100.0	0.0	5.0	10.0	15.0	20.0	25.0	50.0	75.0	100.0
simulations	31.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	30.0	30.0	32.0	33.0	32.0	32.0	32.0	32.0	32.0
utedrop	.000	652.	3410.	8770.	15800.	23500.	69600.	121000.	174000.	.000	1030.	3410.	6230.	8810.	9910.	14200.	16100.	16400.
hello	6250.	6250.	6250.	6250.	6250.	6240.	6250.	6250.	6250.	n/a								
tc	1630.	1650.	1720.	1740.	1750.	1750.	1730.	1710.	1680.	n/a								
total	7880.	7900.	7960.	7990.	7990.	7990.	7980.	7950.	7930.	n/a								
ved	.000	8170.	12400.	14400.	16100.	18000.	23200.	29300.	34600.	.000	8950.	14300.	14300.	13700.	13900.	14800.	13800.	15000.
	.000	12500.	25000.	37600.	50100.	62600.	125000.	188000.	250000.	.000	12500.	25000.	37600.	50100.	62600.	125000.	188000.	250000.
—_AODV	n/a									.000	.0777	.526	2.48	7.95	11.8	34.1	53.1	71.9
—_ARP	.000	.233	1.23	2.41	2.78	3.74	6.05	7.44	8.17	.000	.0302	.355	2.13	6.04	9.86	31.3	58.1	84.7
—_OLSR	.012	.906	4.17	6.51	7.23	7.90	9.34	9.36	9.23	n/a								
—_cbr	.000	2.69	15.2	28.7	40.1	48.7	85.2	103.	121.	.000	1.18	15.2	45.5	82.6	116.	290.	466.	657.
RP_AODV	n/a									.000	.154	.557	2.30	5.54	7.82	15.7	21.8	27.7
RP_cbr	.000	1.47	4.07	6.79	7.96	9.91	14.7	17.3	18.2	.000	.0563	.580	2.02	4.36	6.60	20.0	36.9	52.8
END_AODV	n/a									.000	.0284	.0308	.0274	.0333	.0388	.0771	.114	.131
END_cbr	.000	.0218	.032	.0369	.0448	.0438	.058	.0576	.0573	.000	.004	.0044	.00514	.00453	.00547	.00944	.016	.0218
—_AODV	n/a									.582	31.2	75.4	124.	165.	195.	278.	326.	365.
—_ARP	.000	.651	2.06	3.98	4.91	6.15	9.87	11.2	11.3	.000	1.01	2.58	6.93	13.2	17.9	35.3	49.8	58.1
—_OLSR	3.42	35.0	64.6	72.9	75.5	75.3	69.1	63.0	57.0	n/a								
—_cbr	.000	12.6	26.1	35.1	39.2	42.2	48.3	49.5	48.3	.000	10.8	20.0	26.9	29.1	31.1	34.2	34.8	34.7
BSY_MAC	.000	.0874	.382	.651	.823	.944	1.27	1.48	1.63	.000	.144	.597	.970	1.15	1.25	1.48	1.63	1.65
COL_MAC	.000	19.0	78.4	123.	148.	170.	214.	228.	233.	.000	26.6	118.	194.	235.	265.	326.	351.	364.
RET_MAC	.000	10.3	16.4	20.5	22.6	23.9	27.7	28.7	27.9	.000	8.32	10.6	13.2	14.1	14.8	15.1	15.3	15.0
CBK_cbr	n/a									.000	7.92	9.93	12.4	14.3	16.7	29.6	46.6	62.0
IFQ_AODV	n/a									.000	.000	.000	.0085	.0186	.0327	.154	.343	.562
IFQ_cbr	n/a									.000	1.06	2.39	6.11	7.96	13.8	40.9	74.9	97.5
LOOP_cbr	.000	.150	.620	1.10	1.41	1.73	2.17	2.55	2.69	.000	.169	.893	2.09	2.85	4.41	7.91	10.7	11.3
NRTE_AODV	n/a									.000	.0117	.105	.790	2.22	2.89	4.77	5.04	5.79
NRTE_cbr	.000	2.60	13.6	34.9	63.1	93.4	278.	482.	693.	.000	4.10	13.5	24.1	33.0	36.8	52.2	59.3	59.9
TOUT_AODV	n/a									.000	.000	.012	.004	.006	.00691	.0115	.00983	.0107
TOUT_cbr	n/a									.000	.291	.187	.276	.216	.245	.360	.334	.338
TTL_AODV	n/a									.000	3.07	6.36	10.1	13.2	14.7	18.3	20.1	22.8
TTL_cbr	.000	.193	.382	.366	.306	.334	.244	.199	.176	.000	.155	.317	.484	.454	.504	.629	.516	.337
idth_byterate_AODV	n/a									4810.	6690.	10000.	15600.	21300.	24300.	32700.	36300.	39200.
idth_byterate_ARP	.000	216.	416.	583.	670.	756.	975.	1090.	1080.	.000	162.	329.	655.	1040.	1300.	2300.	3060.	3600.
idth_byterate_MAC	.000	21100.	40600.	50900.	57600.	61600.	73100.	79200.	83000.	.000	24900.	48900.	66300.	79400.	86100.	107000.	115000.	120000.
idth_byterate_OLSR	9800.	9460.	8750.	7530.	6900.	6450.	5020.	4360.	3970.	n/a								
idth_byterate_cbr	.000	72900.	131000.	158000.	178000.	189000.	221000.	242000.	257000.	.000	90200.	159000.	193000.	214000.	224000.	273000.	293000.	298000.
idth_packetrate_AODV	n/a									50.1	68.6	102.	157.	215.	245.	328.	364.	392.
idth_packetrate_ARP	.000	2.70	5.20	7.29	8.37	9.45	12.2	13.6	13.5	.000	2.03	4.11	8.19	13.0	16.2	28.8	38.3	44.9
idth_packetrate_MAC	.000	514.	983.	1230.	1390.	1490.	1760.	1910.	2000.	.000	610.	1190.	1610.	1920.	2090.	2600.	2790.	2900.
idth_packetrate_OLSR	136.	131.	122.	105.	95.8	89.6	69.7	60.5	55.1	n/a								
idth_packetrate_cbr	.000	125.	224.	270.	305.	324.	378.	414.	440.	.000	154.	273.	331.	366.	384.	468.	502.	511.
cketdelay	.000	.200	.600	.931	1.07	1.19	1.39	1.34	1.35	.000	.209	.797	1.27	1.39	1.37	1.48	1.51	1.48

Load Test - Standard Deviations																		
	OLSR									AODV								
	0.0	5.0	10.0	15.0	20.0	25.0	50.0	75.0	100.0	0.0	5.0	10.0	15.0	20.0	25.0	50.0	75.0	100.0
Number of simulations	31.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	30.0	30.0	32.0	33.0	32.0	32.0	32.0	32.0	32.0
count-noroutedrop	.000	473.	1120.	1700.	1490.	1680.	2120.	2290.	2520.	.000	550.	643.	611.	938.	804.	1530.	1530.	1150.
count-olsr_hello	9.65	13.1	10.7	9.05	8.25	9.25	11.7	12.8	10.2	n/a								
count-olsr_tc	73.5	56.4	65.0	61.7	60.9	64.8	58.4	53.6	42.8	n/a								
count-olsr_total	74.6	60.6	69.7	63.3	60.7	67.2	61.6	55.5	41.5	n/a								
count-received	.000	887.	1410.	1400.	1930.	1900.	2280.	2120.	2970.	.000	784.	1850.	1730.	2040.	2310.	2240.	2300.	1900.
count-sent	.000	1.52	2.14	2.88	3.22	3.59	5.90	6.82	6.04	.000	1.74	2.86	3.08	2.26	3.26	5.39	6.41	8.55
rate-IFQ_—_AODV	n/a									.000	.0625	.326	1.04	3.95	3.73	6.62	11.0	9.89
rate-IFQ_—_ARP	.000	.223	.461	.567	.866	.911	1.44	1.62	1.81	.000	.0391	.247	.937	2.65	2.33	6.21	15.1	12.7
rate-IFQ_—_OLSR	.00566	.451	.899	1.31	1.32	1.42	1.39	1.05	.931	n/a								
rate-IFQ_—_cbr	.000	1.68	2.75	3.91	5.28	6.45	10.2	9.15	11.0	.000	1.02	4.02	5.76	8.60	10.3	19.2	19.8	30.6
rate-IFQ_ARP_AODV	n/a									.000	.0761	.217	.876	2.40	2.27	3.82	5.94	5.31
rate-IFQ_ARP_cbr	.000	.618	.994	1.05	1.77	1.91	2.71	2.77	2.92	.000	.0523	.293	.666	1.43	1.41	3.54	8.36	7.23
rate-IFQ_END_AODV	n/a									.000	.015	.0114	.0114	.0129	.0159	.0216	.0491	.042
rate-IFQ_END_cbr	.000	.00844	.00953	.0121	.0116	.0175	.0178	.0173	.0163	.000	.000	.00126	.00195	.00141	.00239	.00502	.0082	.0127
rate-MAC_—_AODV	n/a									.105	5.13	9.22	17.9	28.2	28.9	34.5	42.0	32.4
rate-MAC_—_ARP	.000	.349	.621	.974	1.38	1.84	2.18	2.10	1.97	.000	.350	.804	1.69	4.40	4.75	7.48	13.2	8.40
rate-MAC_—_OLSR	.822	4.95	11.3	10.6	10.7	11.4	7.95	6.14	3.83	n/a								
rate-MAC_—_cbr	.000	2.18	3.78	4.08	4.41	4.10	3.78	2.79	3.82	.000	1.54	3.05	3.21	2.95	3.13	3.24	2.87	3.41
rate-MAC_BSY_MAC	.000	.0436	.0936	.117	.155	.113	.136	.224	.180	.000	.105	.135	.124	.156	.203	.189	.206	.221
rate-MAC_COL_MAC	.000	6.75	15.6	15.9	15.6	21.0	16.3	15.7	19.9	.000	9.14	17.3	17.6	15.1	15.8	19.7	15.2	19.2
rate-MAC_RET_MAC	.000	1.71	1.80	1.99	1.98	2.27	2.48	1.34	2.00	.000	1.27	1.31	1.32	.896	1.35	1.09	1.17	1.29
rate-RTR_CBK_cbr	n/a									.000	1.19	1.26	1.44	1.27	1.82	3.51	8.61	7.53
rate-RTR_IFQ_AODV	n/a									.000	.000	.000	.00542	.0201	.0209	.0562	.106	.150
rate-RTR_IFQ_cbr	n/a									.000	1.06	2.24	3.33	3.44	5.16	15.7	15.3	21.5
rate-RTR_LOOP_cbr	.000	.127	.279	.452	.447	.589	.591	.644	.625	.000	.361	.873	1.67	1.42	2.32	2.23	2.73	2.60
rate-RTR_NRTE_AODV	n/a									.000	.00619	.0732	.458	.940	.913	1.14	.719	.905
rate-RTR_NRTE_cbr	.000	1.89	4.47	6.77	6.02	6.73	8.54	9.34	9.98	.000	2.20	2.55	2.45	3.81	3.44	6.48	6.29	4.59
rate-RTR_TOUT_AODV	n/a									.000	.000	.000	.000	.00231	.00404	.0117	.00624	.00842
rate-RTR_TOUT_cbr	n/a									.000	.107	.0462	.212	.123	.199	.300	.210	.230
rate-RTR_TTL_AODV	n/a									.000	.396	.851	1.75	2.25	2.07	1.84	1.90	1.78
rate-RTR_TTL_cbr	.000	.119	.120	.154	.0945	.127	.103	.104	.0685	.000	.227	.427	.446	.384	.469	.478	.340	.235
rate-bandwidth_byterate_AODV	n/a									4.27	430.	1020.	1590.	2600.	2360.	2110.	1040.	1300.
rate-bandwidth_byterate_ARP	.000	46.3	67.0	67.4	103.	111.	136.	120.	127.	.000	26.2	58.5	107.	235.	217.	340.	447.	419.
rate-bandwidth_byterate_MAC	.000	2300.	3110.	3110.	3240.	3280.	2430.	4010.	4250.	.000	2730.	3940.	3520.	3500.	4220.	4760.	4650.	3970.
rate-bandwidth_byterate_OLSR	784.	770.	718.	498.	365.	413.	275.	234.	169.	n/a								
rate-bandwidth_byterate_cbr	.000	7200.	10100.	9940.	10900.	10800.	9680.	14400.	16500.	.000	8050.	12400.	15000.	15300.	19400.	21800.	20200.	19100.
rate-bandwidth_packetrate_AODV	n/a									.0445	4.28	10.2	16.0	26.3	23.9	21.7	10.5	13.4
rate-bandwidth_packetrate_ARP	.000	.579	.838	.842	1.29	1.39	1.70	1.50	1.59	.000	.327	.731	1.33	2.94	2.71	4.25	5.59	5.24
rate-bandwidth_packetrate_MAC	.000	55.4	75.0	74.6	78.1	78.9	58.8	97.6	104.	.000	65.9	95.5	85.9	84.3	103.	116.	114.	96.8
rate-bandwidth_packetrate_OLSR	10.9	10.7	9.98	6.91	5.07	5.74	3.82	3.25	2.35	n/a								
rate-bandwidth_packetrate_cbr	.000	12.3	17.3	17.0	18.7	18.6	16.6	24.6	28.3	.000	13.8	21.2	25.8	26.1	33.2	37.4	34.5	32.6
time-avgpacketdelay	.000	.101	.171	.167	.224	.230	.268	.202	.180	.000	.107	.218	.187	.231	.180	.188	.171	.185

Cluster Test - Means		
	OLSR	AODV
Number of simulations	31.0	32.0
count-noroutedrop	38000.	4300.
count-olsr_hello	4130.	n/a
count-olsr_tc	910.	n/a
count-olsr_total	5040.	n/a
count-received	34900.	33600.
count-sent	90500.	90300.
rate-IFQ_—_AODV	n/a	2.37
rate-IFQ_—_ARP	.747	.678
rate-IFQ_—_OLSR	1.88	n/a
rate-IFQ_—_ack	.083	.195
rate-IFQ_—_cbr	56.9	135.
rate-IFQ_—_tcp	.417	.922
rate-IFQ_ARP_AODV	n/a	.293
rate-IFQ_ARP_ack	.012	.0102
rate-IFQ_ARP_cbr	1.60	.758
rate-IFQ_ARP_tcp	.0135	.0106
rate-IFQ_END_AODV	n/a	.00533
rate-IFQ_END_ack	.006	.004
rate-IFQ_END_cbr	.006	.006
rate-IFQ_END_tcp	.004	.004
rate-MAC_—_AODV	n/a	50.7
rate-MAC_—_ARP	.421	1.06
rate-MAC_—_OLSR	16.7	n/a
rate-MAC_—_ack	.218	.259
rate-MAC_—_cbr	17.7	20.4
rate-MAC_—_tcp	1.16	1.43
rate-MAC_BSY_MAC	.470	.793
rate-MAC_COL_MAC	120.	229.
rate-MAC_RET_MAC	9.63	6.60
rate-RTR_CBK_ack	n/a	.092
rate-RTR_CBK_cbr	n/a	6.61
rate-RTR_CBK_tcp	n/a	.188
rate-RTR_IFQ_AODV	n/a	.0185
rate-RTR_IFQ_ack	n/a	.0128
rate-RTR_IFQ_cbr	n/a	59.4
rate-RTR_IFQ_tcp	n/a	.337
rate-RTR_LOOP_ack	.00753	.0076
rate-RTR_LOOP_cbr	1.16	3.76
rate-RTR_LOOP_tcp	.00975	.024
rate-RTR_NRTE_AODV	n/a	.0983
rate-RTR_NRTE_ack	.115	.0268
rate-RTR_NRTE_cbr	151.	16.9
rate-RTR_NRTE_tcp	1.13	.151
rate-RTR_TOUT_AODV	n/a	.004
rate-RTR_TOUT_cbr	n/a	.794
rate-RTR_TOUT_tcp	n/a	.0274
rate-RTR_TTL_AODV	n/a	5.13
rate-RTR_TTL_ack	.0144	.0055
rate-RTR_TTL_cbr	.692	1.41
rate-RTR_TTL_tcp	.929	.0173
rate-bandwidth_byterate_AODV	n/a	10900.
rate-bandwidth_byterate_ARP	135.	180.
rate-bandwidth_byterate_MAC	75700.	104000.
rate-bandwidth_byterate_OLSR	3200.	n/a
rate-bandwidth_byterate_ack	1410.	1410.
rate-bandwidth_byterate_cbr	65600.	85200.
rate-bandwidth_byterate_tcp	16000.	18000.
rate-bandwidth_packetrate_AODV	n/a	109.
rate-bandwidth_packetrate_ARP	1.69	2.25
rate-bandwidth_packetrate_MAC	1860.	2550.
rate-bandwidth_packetrate_OLSR	44.4	n/a
rate-bandwidth_packetrate_ack	12.6	12.6
rate-bandwidth_packetrate_cbr	482.	626.
rate-bandwidth_packetrate_tcp	14.9	16.7
time-avgpacketdelay	.390	.770

Table B.37.: *Cluster Test - Means.*

Cluster Test - Standard Deviations		
	OLSR	AODV
Number of simulations	31.0	32.0
count-noroutedrop	9310.	1650.
count-olsr_hello	7.58	n/a
count-olsr_tc	112.	n/a
count-olsr_total	113.	n/a
count-received	6100.	4410.
count-sent	440.	310.
rate-IFQ_—_AODV	n/a	1.32
rate-IFQ_—_ARP	.571	.523
rate-IFQ_—_OLSR	.792	n/a
rate-IFQ_—_ack	.0457	.0681
rate-IFQ_—_cbr	23.5	40.9
rate-IFQ_—_tcp	.204	.282
rate-IFQ_ARP_AODV	n/a	.149
rate-IFQ_ARP_ack	.00828	.00669
rate-IFQ_ARP_cbr	1.01	.465
rate-IFQ_ARP_tcp	.00685	.0075
rate-IFQ_END_AODV	n/a	.00261
rate-IFQ_END_ack	.00283	.000
rate-IFQ_END_cbr	.00566	.00283
rate-IFQ_END_tcp	.000	.000
rate-MAC_—_AODV	n/a	9.48
rate-MAC_—_ARP	.217	.779
rate-MAC_—_OLSR	6.23	n/a
rate-MAC_—_ack	.080	.0853
rate-MAC_—_cbr	3.76	4.53
rate-MAC_—_tcp	.496	.430
rate-MAC_BSY_MAC	.211	.226
rate-MAC_COL_MAC	48.6	52.7
rate-MAC_RET_MAC	1.72	1.79
rate-RTR_CBK_ack	n/a	.0423
rate-RTR_CBK_cbr	n/a	1.95
rate-RTR_CBK_tcp	n/a	.0652
rate-RTR_IFQ_AODV	n/a	.0107
rate-RTR_IFQ_ack	n/a	.00833
rate-RTR_IFQ_cbr	n/a	37.1
rate-RTR_IFQ_tcp	n/a	.202
rate-RTR_LOOP_ack	.00564	.00295
rate-RTR_LOOP_cbr	.809	2.90
rate-RTR_LOOP_tcp	.0114	.0135
rate-RTR_NRTE_AODV	n/a	.0755
rate-RTR_NRTE_ack	.0514	.0123
rate-RTR_NRTE_cbr	37.1	6.53
rate-RTR_NRTE_tcp	.344	.0564
rate-RTR_TOUT_AODV	n/a	.000
rate-RTR_TOUT_cbr	n/a	.628
rate-RTR_TOUT_tcp	n/a	.0306
rate-RTR_TTL_AODV	n/a	1.06
rate-RTR_TTL_ack	.00905	.00298
rate-RTR_TTL_cbr	.328	1.09
rate-RTR_TTL_tcp	.734	.0136
rate-bandwidth_byterate_AODV	n/a	976.
rate-bandwidth_byterate_ARP	46.0	62.5
rate-bandwidth_byterate_MAC	15300.	12000.
rate-bandwidth_byterate_OLSR	545.	n/a
rate-bandwidth_byterate_ack	292.	270.
rate-bandwidth_byterate_cbr	13000.	9360.
rate-bandwidth_byterate_tcp	3160.	2850.
rate-bandwidth_packetrate_AODV	n/a	9.90
rate-bandwidth_packetrate_ARP	.575	.781
rate-bandwidth_packetrate_MAC	375.	292.
rate-bandwidth_packetrate_OLSR	7.56	n/a
rate-bandwidth_packetrate_ack	2.60	2.41
rate-bandwidth_packetrate_cbr	95.7	68.8
rate-bandwidth_packetrate_tcp	2.94	2.66
time-avgpacketdelay	.151	.203

Table B.38.: *Cluster Test - Standard Deviations.*