Aalborg University
Department of Computer Science

DAT6

Title:
Extensive Simulation Based Evaluation of MANET Broadcast Protocols using A Simulation Framework

Project period:
February 1st, 2002 –
June 14th, 2002

Project group:
DAT6 E4-113

Authors:
Tue Brems Olesen
<tue@cs.auc.dk>

Nicolai Larsen
<cbaoth@cs.auc.dk>

Supervisor:
Thomas Heide Clausen

Number of copies: 6

Number of pages: 121 + viii

Appendix: 3

Abstract:

This report documents an extensive simulation based study of MANET broadcast protocols, and the development of a framework which automates the simulation process.
Four broadcast protocols as well as two generic protocol extensions are simulated in ns2. From the simulation results it is determined that broadcasting via OLSR's MPRs yields the best overall performance. Also, it is determined that enforcing jitter on data traffic improves the delivery rate.
The framework is applied to conduct the simulations in this work, and significantly reduces the manual work required, from specification of abstract scenario parameters, to visualisation of simulation results.

Preface

This report documents the second part of the master's year (speciale) in computer science, at Aalborg University, Denmark. The work was conducted in the period from February 1st, 2002 to June 14th, 2002.

The formal purpose of this particular part of the education is to autonomously conduct empirical and/or theoretical studies of central subjects in the field of distributed systems, by applying theories and methods on a scientific level.

This report documents practical and theoretical work in the fields of Mobile Ad-hoc Networks, protocol simulation and generic problems related to broadcast in Mobile Ad-hoc Networks.

Citations are written in brackets (for example: [JMQ+02]), and a bibliography of all cited references is included, starting on page 119. A glossary is included in the appendix of the report, starting on page 113.

Acknowledgements

The authors would like to thank INRIA Rocquencourt in Paris for providing hospitality and working facilities during the project period. Furthermore, the authors would like to thank Mindpass Centre for Distributed Systems at Aalborg University for providing computing capacity to perform the simulations presented in this report.

This work is in part funded by Oticonfonden (Denmark), Siemens Fonden (Denmark), and Project HiPERCOM at Inria Rocquencourt (France).

Nicolai Larsen *Tue Brems Olesen*

Contents

1 Introduction **1**
- 1.1 Wireless Networks . . . 1
- 1.2 Mobile Ad-hoc Networks . . . 2
- 1.3 Broadcast Protocols . . . 3
  - 1.3.1 Related Work . . . 4
- 1.4 Scenario Based Protocol Evaluation . . . 6
  - 1.4.1 Formal Analysis . . . 6
  - 1.4.2 Simulation . . . 6
  - 1.4.3 Practical Experiments . . . 7
  - 1.4.4 Selection of A Method For Protocol Evaluation . . . 7
- 1.5 Large-scale Simulation Based Studies . . . 8
  - 1.5.1 A Simulation Machinery . . . 8
  - 1.5.2 Related Work . . . 9
- 1.6 Goals . . . 11
- 1.7 Report Outline . . . 12
- 1.8 Summary . . . 12

2 Duplicate Packet Elimination **13**
- 2.1 Introduction . . . 13
- 2.2 The Cause of Duplicate Packets . . . 13
- 2.3 Eliminating Duplicate Packets . . . 15
  - 2.3.1 Approach For Duplicate Elimination . . . 15
  - 2.3.2 Using The "IP Identification" Field . . . 16
  - 2.3.3 Packet history size . . . 17
- 2.4 Summary . . . 18

3 MANET Broadcast Protocols **19**
- 3.1 Introduction . . . 19
- 3.2 A Generic Broadcast Algorithm . . . 20
  - 3.2.1 Condition 1: Addressing . . . 20

3.2.2 Condition 2: Duplicate Packets 21
3.2.3 Condition 3: Protocol-specific Optimisations 22
3.3 Broadcast Protocols 22
3.3.1 Protocol Classes 22
3.3.2 Selecting Broadcast Protocols 24
3.3.3 Protocol Extensions 27
3.3.4 Summary of selected protocols 28
3.4 Evaluation Strategy 29
3.4.1 Dimensions and Granularity 29
3.4.2 Statistical Significance 30
3.4.3 Estimated Extent of Simulations 30
3.5 Summary 32

4 Simulation Framework 33
4.1 Introduction 33
4.2 Framework Overview 33
4.2.1 Simulation platform 33
4.2.2 Simulation Stages 34
4.2.3 Data Storage 36
4.3 Framework Components 39
4.3.1 Scenario Generator 39
4.3.2 Network Simulator 40
4.3.3 Job Scheduler 42
4.3.4 Trace File Analyser 44
4.3.5 Summary Generator 46
4.3.6 Graph Generator 46
4.4 Summary 48

5 Protocol Evaluation 51
5.1 Introduction 51
5.2 Metrics 52
5.2.1 Effectiveness 52
5.2.2 Delivery Rate 56
5.2.3 Delay and Path Length 56
5.2.4 Bandwidth Utilisation 57
5.2.5 Average Graphs 57
5.3 Simulation Setup 58
5.4 Test 1: Four Flooding Protocols 59
5.4.1 Expected Results 59
5.4.2 Results 60
5.4.3 Conclusions 67

5.4.4 Summary . . . 68
5.5 Test 2: Classic Flooding with OLSR . . . 68
5.5.1 Expected Results . . . 69
5.5.2 Results . . . 69
5.5.3 Conclusions . . . 79
5.5.4 Summary . . . 79
5.6 Test 3: Full History Duplicate Elimination . . . 79
5.6.1 Expected Results . . . 80
5.6.2 Results . . . 80
5.7 Test 4: Jitter on Data Packets . . . 80
5.7.1 Expected Results . . . 81
5.7.2 Results . . . 81
5.7.3 Conclusions . . . 85
5.7.4 Summary . . . 88
5.8 Test 5: Multipacket Flooding . . . 88
5.8.1 Expected Results . . . 88
5.8.2 Results . . . 89
5.8.3 Conclusions . . . 94
5.8.4 Summary . . . 95
5.9 Conclusions . . . 95
5.10 Summary . . . 96

6 Simulation Framework Evaluation 97
6.1 Introduction . . . 97
6.2 Evaluation of Individual Components . . . 97
6.2.1 Job Scheduler . . . 98
6.2.2 Trace File Analysis Tool . . . 99
6.2.3 Graph Compilation Environment . . . 101
6.2.4 Data Storage . . . 102
6.3 Portability Issues . . . 103
6.4 Overall Experiences . . . 103
6.5 Summary . . . 104

7 Conclusions 107
7.1 Products . . . 107
7.1.1 Broadcast in MANETs . . . 107
7.1.2 Simulation Framework . . . 108
7.2 Results . . . 108
7.2.1 Broadcast Protocols . . . 108
7.2.2 Simulation Framework . . . 109
7.3 Future Work . . . 110

7.3.1 Broadcast Protocols 110
7.3.2 Simulation Framework 111
7.4 Summary 112

8 Appendix 113
A Glossary 113
B Selection of Dominating Sets 116
C Simulation Platform Details 117
C.1 Hardware Platform 117
C.2 Total Simulation Time 118

Bibliography 119

Chapter 1

Introduction

This chapter introduces and motivates the work on simulation-based studies of broadcasting in Mobile Ad-hoc Networks (MANETs), documented in the following chapters. Section 1.1 describes two major paradigms for organisation of wireless networks: cellular networks and MANETs, and section 1.2 elaborates on the latter of these network types. Section 1.3 introduces the subject of broadcasting in MANETs, and presents previous work conducted to evaluate various broadcast protocols.

Sections 1.4 presents methods for scenario based protocol evaluation, and section 1.5 describes how one such method, protocol simulation, can be applied to conduct extensive protocol evaluation with minimal effort.

Goals for this work are defined in section 1.6, and section 1.7 outlines the remainder of this report.

1.1 Wireless Networks

Wireless data communication has been an active area of research during the last two decades. In the eighties, work was conducted in the area of *packet radio networks*, focusing primarily on military application of those networks [JT87]. During the nineties, several new wireless networking technologies has emerged, and two extremes of wireless network types have become apparent: Cellular Networks and MANETs.

Cellular networks, established using, e.g., GSM [GSM02] or GPRS [GPR01], consists of a grid of fixed interconnected stations. Communication between mobile devices is established via the stations in the respective cells, even when the mobile devices are within direct transmission range of each other.

MANETs, established using, e.g., IEEE 802.11 [soc99] or Bluetooth [Blu02], define the other extreme. MANETs are self-organising, autonomous systems where nodes establish and maintain a wireless communication infrastructure in an ad-hoc fashion, during operation of the network. MANETs are independent of any fixed network infrastructure, and devices may join, leave and move around in the MANET at any time.

1.2 Mobile Ad-hoc Networks

MANETs are applicable in situations where a set of wireless, communicating devices are not all within direct transmission range of each other. Such situations present themselves in several situations: when people bring along their mobile computers, e.g., to conferences or collaborative meetings, when carrying out work in remote locations and in rescue and military operations.

Furthermore, the autonomous, auto-configuring properties of MANET routing protocols can be applied in wired network routers, to reduce the work required to configure routes when the network topology is changed.

As MANETs are independent of fixed network infrastructures, the inherent hierarchical structure known from wired networks is nonexistent. Two devices in a MANET may communicate though they are not within transmission range of each other, by having intermediate devices forward their traffic. This functionality requires each physical unit in the MANET to act as both a *host* and a *router*. This encapsulation hereafter referred to as a *node*, as illustrated in figure 1.1.

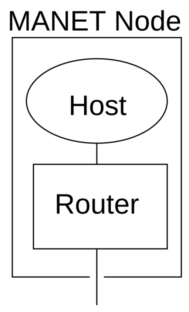

Figure 1.1: A MANET node, consisting of a host and a router contained in the same physical unit.

Three communication patterns are observed in wired networks: *unicast* communication between pairs of nodes, *multicast* where one node transmits data destined for all nodes in a multicast group, and *broadcast*[1] [Bak95], where one node transmits data to all other nodes in its local network.

Enabling these communication patterns in a MANET raise several issues, among those are changes in the physical, data link and network layers of the OSI protocol stack [LO02]. The remaining parts of this report will focus on the issues and challenges related to the network layer, in particular those related to the process of *broadcasting* in MANETs.

[1]The term "broadcast" does here loosely include both "Directed Broadcast" and "Limited Broadcast" [Bak95]

The following section presents an example of how two broadcast protocols can achieve different amounts of bandwidth overhead involved in broadcasting a message, and gives a presentation and evaluation of major works conducted to evaluate different broadcast protocols.

1.3 Broadcast Protocols

Broadcast is utilised in three of the routing protocols [JMHJ02, PBRD02, JMQ^{+}02] currently being developed in the MANET working group [man02], for the purpose of distributing control traffic. Furthermore, broadcast may be used for transporting data for user applications.

For the purpose of this work, *broadcast* is defined as follows.

> Broadcast: *"the process of delivering a packet to every node within the MANET under consideration."*

The fact that bandwidth is a constrained resource in MANETs [CM99] motivates the use of optimised broadcast protocols. The optimisations considered in this work are intended to reduce the bandwidth overhead and to improve the probability of delivering a broadcast packet to all nodes in the MANET. For the purpose of this work, *bandwidth overhead* is defined as follows.

> Bandwidth Overhead: *"the number of unnecessary forwards involved in delivery of a broadcast packet to all nodes in the MANET under consideration."*

Observing that the wireless network medium may have inherent broadcast capabilities, as those found in e.g. IEEE 802.11 networks, it is clear that in a collision-free environment, a single broadcast message will be received by all nodes within transmission range of the sender. Thus, by reducing the number of transmissions required to cover all nodes in the network, the bandwidth overhead of broadcasting a message can be reduced. One example of the ideal reduction is illustrated in figure 1.2.

For the purpose of this work, the term *"transmit"* denotes the original transmission of a packet. Nodes other than the originator is said to *"forward"* the packet.

If all nodes receive the packet, eight forwards will occur, as illustrated in figure 1.2b. This algorithm is denoted "Classic Flooding". The minimal overhead is achieved by letting only the nodes in the *minimal connected dominating set* forward the message, in this situation generating two forwards, as illustrated in figure 1.2c.

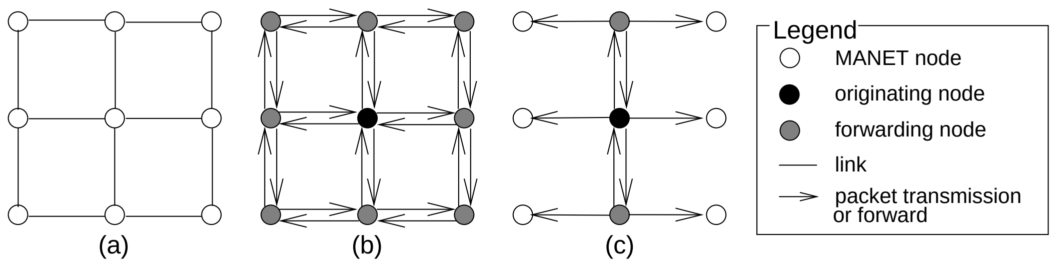

Figure 1.2: (a) An example MANET. (b) Classic Flooding generates 8 forwards. (c) Broadcast via nodes in the minimal connected dominating set generates 2 forwards.

1.3.1 Related Work

With respect to the evaluation of MANET broadcast protocols, two works stand out: [NTCS99] and [WC02]. The subjects and conclusions of these works are presented below. Further, broadcast protocols are *applied* in a number of unicast routing protocols developed in the MANET working group [PBRD02, JMHJ02, JMQ$^+$02].

- [NTCS99] considers the issues of redundant rebroadcasts, contention and collision, as a whole referred to as "the broadcast storm problem".

  Through analysis, it is shown that the maximum additional coverage which can be obtained by the first forward of a originated message is 61% of the area covered by the original transmission. By simulation, it is shown that the expected additional coverage for a node quickly drops, as the number of duplicate messages received by that node increases. When receiving four or more duplicates, a node's expected additional coverage is below 0.05%. Furthermore, it is shown by analysis that contention and collision are also likely to be present, adding further to the broadcast storm problem.

  Next, probability-based, distance-based, location-based and cluster-based broadcast schemes are evaluated using a custom-built network simulator. From the simulation results, it is concluded that the largest reduction of redundant rebroadcasts are achieved by the location-based protocols.

- [WC02] define four classes of broadcast protocols: simple flooding, probability based methods, area based methods and neighbour knowledge based methods.

> A performance comparison of protocol classes is conducted, using the network simulator “ns2”. One protocol is selected to represent each class. Furthermore, a worst-case bound is included, represented by the Classic Flooding algorithm, and a theoretical best-case is represented by flooding via the minimum connected dominating set in each simulated node configuration.
>
> The protocols are studied in static networks, mobile networks, congested networks and a combination of the three, with the purpose of pinpointing situations where the protocols perform well, and where improvements are possible.
>
> This results of this work supports the conclusion in [NTCS99], namely that the location-based scheme yields better performance than the probability and counter based schemes. Furthermore, it is concluded that the neighbour knowledge based protocols yield even better performance, and directions for future examinations of protocols in this class are given.

The work of the MANET working group is targeted towards stabilisation and standardisation of MANET routing protocols. Of the six MANET routing protocols currently having status as IETF drafts, DSR [JMHJ02] and AODV [PBRD02] use Classic Flooding as a broadcast mechanism. OLSR uses *multipoint relays* (MPRs) to optimise the broadcast of control messages [JMQ$^+$02]. This technique is in the class of neighbour knowledge based protocols.

Evaluation and comparison of the performance of these protocols have been carried out in, e.g., [CHCB01, DPR00, Qay00, BMJ$^+$98, JLH$^+$99b]. These works evaluate the overall performance of the routing protocols. Analytical evaluations of the optimisation of OLSR by broadcasting via MPRs is presented in [JL99, QVL00].

Evaluation of Related Work

The observations in [NTCS99] regarding the broadcast storm problem show that reductions in the bandwidth overhead over the Classic Flooding approach are achievable by reducing the number of redundant forwards. The simulation results presented in[NTCS99] are problematic, as the behaviour of the custom-built simulator is not specified in detail, thus not allowing for comparison with results from other simulators in widespread use.

The comparison of broadcast protocol classes in [WC02] are based on simulations of one or two protocols selected to be representative for each class. The selection of protocols is based on considerations about the theoretical functionality of the protocols, and comparisons of previous simulation results, obtained from different works which do *not* use the same simulation tools and techniques.

Work conducted so far to evaluate and compare DSR, AODV and OLSR has focused on the performance of the routing protocols as a whole. No known large simulation based studies evaluate the performance of the broadcast mechanisms applied in these routing protocols exist.

1.4 Scenario Based Protocol Evaluation

The development of routing protocols tailored for use in MANETs is an active area of research, embracing routing protocol specification and formal proofs of correctness, as well as evaluation and comparison of the performance of various routing protocols. *Scenario based evaluation* is widely used to evaluate and compare protocols, and three approaches to scenario based evaluation exist: formal analysis, simulation and practical experiments.

1.4.1 Formal Analysis

Formal analysis of protocol performance as in [JL99, JV00], is a feasible method to examine the communication and computation complexities of routing protocols, isolated from the effects imposed by actual implementations, network media characteristics and changing signal propagation conditions. The protocol performance results obtained by formal analysis is thus independent of network conditions.

Though not impossible, formal analysis becomes intractably complex and time-consuming in complex scenarios. Hence, formal analysis has its limits both with respect to the degree of realism and the complexity of the analysed scenarios.

1.4.2 Simulation

Evaluation through simulation is performed using a software network simulator providing a model of a network environment, in which routing protocols can be implemented. Evaluation of a protocol is done by simulating the protocol operation in various scenarios, which defines node configurations and movements, communication patterns and physical surroundings. The performance of the protocol can be observed as its ability to route the traffic generated in a scenario, and is measured as the actual *network performance.* E.g., packet delivery rates, path lengths and bandwidth consumption achieved by the protocol.

Current network simulation tools are restricted to evaluation of two-dimensional scenarios, to keep the computations required by the network environment model at a reasonable scale. The realism of the simulation results are limited by the accuracy of the simulators model of the real world.

1.4.3 Practical Experiments

Practical experiments is the third approach to evaluating MANET routing protocols. Typically, such experiments are conducted by creating a set-up of a number of nodes, e.g., notebooks equipped with IEEE 802.11 network interfaces.

The performance of a routing protocol can be measured by observing properties of the communication between nodes. The benefit of this method is that it shows the performance of a real, specific implementation of a routing protocol, and includes any effects imposed by network technology, physical mobility and physical surroundings. The size of the scenarios that can be constructed is constrained by the available equipment.

Naturally, conducting practical experiments implies great costs in merely obtaining the necessary hardware. The definition of the scenarios in use can be more or less specific: describing "three nodes in an office environment" as opposed to "three nodes placed at certain coordinates in an otherwise empty Faraday cage". The latter allows for unknown factors to be present in the environment, which leads to a reduced, but not necessarily unsatisfying, degree of reproducibility.

1.4.4 Selection of A Method For Protocol Evaluation

The three approaches to scenario-based evaluation illustrate the tradeoff between the desire to obtain reproducible evaluation results with foundation in logical reasoning, and evaluating protocols in realistic scenarios, considering as many properties of a real-life environment as possible.

Formal analysis is constrained to simple scenarios, in environments that do not describe the detailed characteristics of the wireless network. Practical experiments, on the other hand, evaluate protocols in realistic environments, at the expense of large experimental costs, and to some extent, reproducibility of the results.

Simulators provide a tradeoff between formal analysis and practical experiments, allowing for evaluation of larger and more complex scenarios than formal analysis allows, at less than the cost involved practical experiments. Hence, for the present work, network simulators are selected as the method for performing scenario based protocol evaluations.

1.5 Large-scale Simulation Based Studies

A comprehensive evaluation of a MANET routing protocol, with the goal of demonstrating the overall protocol performance, should contain statistically significant results, showing the protocol performance in a variety of scenarios. One method to ease the work required to conduct such evaluations is described in section 1.5.1. Related work and existing tools to manage large-scale simulation based protocol evaluations is presented and evaluated in section 1.5.2.

1.5.1 A Simulation Machinery

With the desire for large-scale simulation based studies in mind, this section describes a method to efficiently conduct such studies. Figure 1.3 illustrates a "simulation machinery", intended to automate the process of conducting large amounts of simulations.

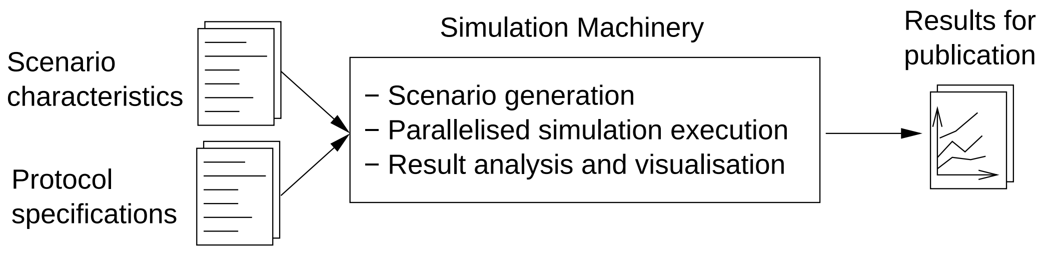

Figure 1.3: The simulation machinery.

The simulation machinery takes as input abstract scenario descriptions and protocol implementations and, with little or no interaction, arrives at a visual presentation of the simulation results. This is an efficient method for conducting large amounts of simulations, potentially speeding up the simulation process and minimising the amount of manual work involved.

Conducting a large amount of simulations based on the input of scenario characteristics and protocol descriptions is a task that lends itself to automation. The large computational requirements involved with conducting the simulations motivates parallelisation of the computationally intensive tasks present in the machinery.

It is noticed that such a machinery must be able to support a wide range of different simulation studies to be successful. Furthermore, a flexible, generic simulation machinery is required to fulfil requirements set by different types of simulations, while maintaining the role of an efficient aid for conducting the simulations.

Structuring such a simulation machinery as a *simulation framework*, provides the requested flexibility: existing and new tools can be integrated, and individual tools in such a framework can be upgraded and replaced, without influencing the other parts of the framework.

1.5.2 Related Work

This section will give an overview of the main works in the field of protocol evaluation by simulation, emphasising on the simulations conducted in these works, and leaving out the conclusions drawn from these simulations. First, three major works in the field of simulation based studies are presented, to identify the quantity of simulations conducted. Next, software tools for conducting simulations are presented.

Simulation based studies

- [CHCB01] presents a scenario based evaluation of the OLSR protocol, conducted using ns2. Scenarios of 50 nodes moving within an area of 1000×1000 m were simulated for a duration of 250 seconds, using one fixed traffic load imposed on the network. By including and excluding jitter on the control traffic of OLSR, and by using various hold-back times for piggybacking control traffic, a total of 420 simulations are defined to evaluate the two optimisations of OLSR.

- [DPR00] compares the performance of DSR and AODV through tree sets of simulations, one set using 1500×300m / 50 node scenarios, the other two using scenarios of 2200×600m / 100 nodes. Five random scenarios are generated for each simulation set, and by varying the number of traffic sources, the amount of mobility and the offered network load, a total of 600 simulations (300 per protocol) are defined to compare the two protocols.

- [JLH+99a] compares the performance of DSR, AODV and DSDV [PB94] using five sets of simulations, of 1000×1000 and 1500×300 m. Only one scenario is generated for each set of simulations. By varying the offered network load and the amount of mobility, a total of 123 simulations are defined to compare the three protocols.

Tools for conducting simulations

The conduction of simulations encompasses both the network simulator and auxiliary tools: scenario generators are used for automatic construction of scenarios, and generic job scheduling systems are available for distribution of calculations. Existing tools for those purposes are shortly presented in the following.

Simulators: Four network simulators are used to conduct the majority of studies of routing protocols:

OpNet [opn02] is a commercial simulator, integrating the implementation of protocols, specification of network scenarios, and conduction of simulations in a graphical environment. No graphical user interfaces are present for the other two simulators considered here, making OpNet unique in this respect.

ns2 [pro02] has become the de-facto simulator applied in research work. ns2 is open-source software, which gives free access to modify the complete simulator source code, and to publish the modifications. A a wide range of features are therefore present in or available for ns2, many of them contributed by the simulator's user community.

GloMoSim [glo02] is freely available for educational purposes, including access to the simulator source code. Its use is observed in several research works, but this simulator has a smaller user community, and lacks the wide range of features present in ns2.

QualNet [qua02]is a commercial simulator, derived from GloMoSim. QualNet is characterised by a dedicated effort towards achieving a detailed model of the physical and data link layers.

Scenario Generators: Along with the ns2 distribution comes separate utilities (`setdest` and `cbrgen`) for generating random node movement patterns and random communication patterns. Together, they enable generation of simple scenarios. Another scenario generator, *wsg* [CHCB01], combines both features in the same tool.

Job Schedulers: The "Maui Scheduler" [SRDG02] is an advanced, generic batch scheduling system intended for use on large clusters of machines. Due to its large selection of advanced features, achieving a comprehensive understanding of the scheduler and the required resource manager software requires a large effort.

Evaluation of Related Work

Among the major works in simulation-based protocol evaluation, [CHCB01] presents by far the highest number of different scenarios for each set of scenario parameters. This work, as well as [DPR00], uses automatic scenario generation. The largest number of simulations from which results are presented is 600, observed in [DPR00], but only in [CHCB01] is the statistical significance of the results considered.

The need for conducting larger simulation-based studies is indeed present, as statistical significant results are obviously preferable to single samples. Nevertheless, the existing studies are all limited to presenting results from a couple of hundreds simulations. One reason may be that little software is presently available to aid the process of conducting simulations, and the lack of integration between this software and the network simulator makes the task of conducting simulations time-consuming and tedious.

Though several tools usable for conducting large quantities of simulations, no known work exists that investigates the integration between these tools, with the goal of aiding the conduction of simulations.

As it is more advanced than the two other tools presented, *wsg* is selected for scenario generation. Of the four network simulators described, only ns2 is open source software. Due to the free availability, and the possibility of reading, modifying and extending the source code, ns2 is selected as the simulator for this work.

1.6 Goals

It has become clear that large scale simulation-based studies of MANET broadcast protocols is yet to be published, and that a simulation framework automating the simulation process can aid such a study. Addressing these issues, the goals of this work are as follows:

- select a set of broadcast protocols and generic extensions that can be implemented on standard MANET nodes with minimal intrusive changes and requirements, and define a strategy for evaluating those protocols,

- develop a simulation framework that automates execution of individual simulations as well as parallel execution of large batches of simulations,

- evaluate the selected broadcast protocols, through an extensive simulation-based study, using the developed simulation machinery to conduct the simulations, and

- evaluate the simulation machinery as a method for automatic simulation processing, based on the experiences gathered from using the machinery to conduct simulations.

1.7 Report Outline

The remains of this report are structured as follows: chapter 2 describes a solution for the generic problem of eliminating duplicate packets that occurs when using multicast and broadcast in MANETs.

Chapter 3 describes different classes of MANET broadcast protocols. From one of these classes, a set of protocols is selected, and generic extensions intended to improve their performance are proposed.

Chapter 4 describes the simulation framework which has been developed to help conduct a simulation based study of the selected broadcast protocols and extensions.

The selected broadcast protocols and extensions are evaluated in chapter 5, and the simulation framework is evaluated in chapter 6.

Conclusions are drawn in chapter 7, which also presents directions for future work.

1.8 Summary

Two major subjects are introduced in this chapter: MANETs and simulation based protocol evaluation. With the advances in wireless networking technology, the construction of MANETs is possible. One of the challenges that must be addressed to achieve good communication performance in a MANET is the routing of traffic. An overview of the existing evaluations and comparisons of routing protocols reveals that a comprehensive study of neighbour-knowledge based broadcast protocols is yet to be conducted.

Scenario based protocol evaluation based on simulations present itself as the most feasible way to conduct such a study, and a simulation machinery to automate the simulation process is presented as a tool to aid such studies.

The next chapter describes a solution for the generic problem of eliminating duplicate packets that occurs when using multicast and broadcast in MANETs, being a prerequisite for the specification of broadcast protocols in chapter 3.

Chapter 2

Duplicate Packet Elimination

This chapter addresses the issue of eliminating duplicate multicast and broadcast packets, that appear in MANETs. Section 2.1 introduces the overall problem and approach to a solution. Section 2.2 elaborates on the cause of nodes receiving duplicate packets. Section 2.3 proposes a solution to the remedy the problem, using only the information present in the IPv4 header, and estimates the storage requirements put forth by this solution.

2.1 Introduction

The version 4 Internet Protocol [Joh81] (IPv4) was designed for use in wired networks. The combination of the inherent broadcast capabilities present in the wireless network medium, and the addressing scheme applied for multicast and broadcast, causes duplicate packets to be received during the forwarding process.

Preventing the duplicates from appearing is an infeasible approach, as it requires changes in the basic properties MANETs, i.e., removing the broadcast capabilities. This chapter investiagtes a feasible approach to addressing the issue, namely to *eliminate* the duplicates.

2.2 The Cause of Duplicate Packets

Broadcast can be observed as an instance of multicast, using as destination a group in which every node is member. This group may be addressed using some reserved multicast IPv4 address, referred to as *All-Manet-Nodes*.

MANET nodes typically use the same wireless interface for receiving and sending packets. When a node forwards a packet, the transmission is overheard by both the “next-hop” and “previous-hop” nodes, causing a duplicate of the packet at the previous-hop node, as illustrated in figure 2.1a. Forwarding of packets on wired networks does not cause this problem, as packets are forwarded through another interface than the one from which they were received, as illustrated in figure 2.1b.

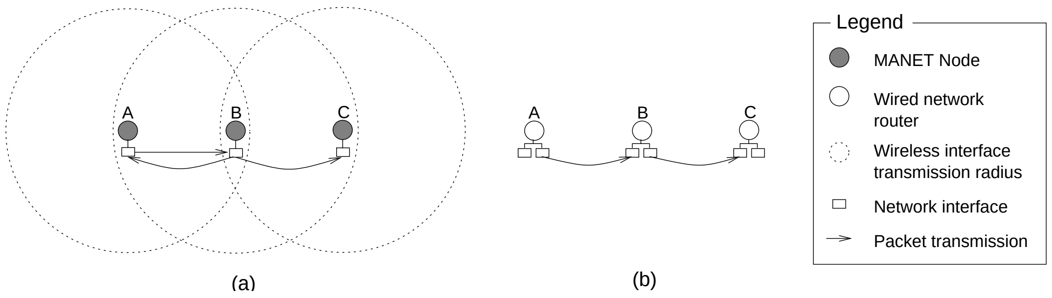

Figure 2.1: (a) Duplicate packets appear when nodes share a broadcast media, and use only one interface. (b) In wired networks using multiple interfaces the problem is avoided.

When a node transmits a packet on a broadcast media (regardless of whether it is unicast, broadcast, or multicast) local filtering is required at the receivers to determine whether or not the packet is intended for reception on this node. For unicast the filtering is straightforward: the receiving interface compares its own MAC address with that present in the MAC frame header of a packet, and discards the packet silently if they mismatch.

When multicasting and broadcasting a packet, the destination in the IPv4 header and, correspondingly the MAC frames, is set to the multicast or broadcast address. For multicast, the group members may accept the packet, and non-member nodes can silently discard it. In case of broadcast, all receiving nodes may accept the packet.

The trouble arise due to the combination of the broadcast nature of the media, and the properties of the MAC addressing scheme: nodes may receive both the original transmission of a packet and subsequent forwards. When forwarding a unicast packet, a new destination is used for the MAC frames generated on forwarding, allowing duplicate packets to be discarded, as illustrated in figure 2.2a.

When forwarding a multicast or broadcast packet, however, the destination address for the forwarded packet remains the same as in the original transmission. As illustrated in figure 2.2b, the originator can discard the duplicate packets by observing that its own address equals that in the originator field of the IPv4 header[1], but other receivers must consider additional information to identify possible duplicate packets. The worst case appears when duplicate packets are not dropped due to, e.g., collisions, and therefore loops between two or more nodes, until its TTL reaches zero.

[1]Normally, the MAC layer filters the incoming packets before delivering them to the IP layer [Ste94], but further filtering in the IP layer, based on the originator and destination address of a packet, is possible.

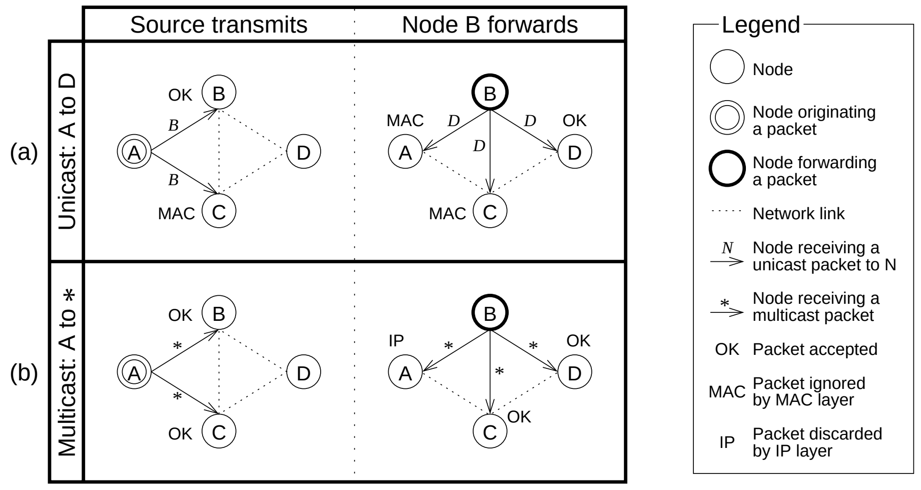
Source transmits
Node B forwards
(a)
Unicast: A to D
(b)
Multicast: A to *
OK
MAC
IP
Legend
Node
Node originating a packet
Node forwarding a packet
Network link
Node receiving a unicast packet to N
Node receiving a multicast packet
OK Packet accepted
MAC Packet ignored by MAC layer
IP Packet discarded by IP layer

Figure 2.2: (a) Forwarding unicast packets using a shared broadcast media enable nodes to eliminate duplicates at packets at the MAC layer. (b) For multicast packets, duplicates cannot be eliminated at the MAC layer. All nodes in this example are members of the multicast group used.

Notice that the problem of duplicates occurs independently of multicast and broadcast. As an example, the OLSR, AODV and DSR routing protocols specify forwarding of packets sent to the Limited Broadcast IPv4 address (255.255.255.255) to distribute control messages. These protocols solve the problem individually, by duplicate elimination of control messages as part of the routing protocol. The mechanism for eliminating duplicate *messages* is independent of that for duplicate IPv4 *packets*, as duplicate messages are forwarded by the routing protocols, resulting in generation of new IPv4 packets. In this work, only elimination of duplicate IPv4 packets is considered.

2.3 Eliminating Duplicate Packets

Systems and applications developed for use in wired networks are not required to deal with duplicate packets. Maintaining the functionality of these systems when migrating to wireless network environments requires a general scheme to be devised, which operates at a level below the application layer.

2.3.1 Approach For Duplicate Elimination

As described in section 2.2, it is necessary to use other information than just the destination address of packets to determine if a given packet is a duplicate. [LO02] describes four solutions in two different categories, briefly summarised here:

Duplicate elimination based on node addresses

These solutions depend on the ability to identify the node which sent the packet, and consider this information during the decision of whether to forward a packet.

- Inclusion of the address of the *sender* of a packet, in addition to the already present *originator* address.
- Tunnelling of packets through unicast IP-in-IP tunnels that allows nodes to identify the remote endpoint (and hence the sender) of received packets.

Duplicate elimination based on packet identification

These solutions depend on the ability to uniquely identify packets or MAC frames, and enable duplicate elimination by matching received packets or frames, with a history of earlier receptions.

- Sequence numbering of IPv4 headers, by having the source node include a sequence number in the IPv4 header, and letting each node maintain a history of sequence numbers of received packets.
- Sequence numbering of MAC frames: the sender of a MAC frame includes a sequence number, and each node maintains a history of sequence numbers.

In the following section, the solution of sequence numbering of IPv4 headers is revised to operate solely on information already present in the IPv4 header.

2.3.2 Using The "IP Identification" Field

The technique, applied locally by each node to eliminate duplicate packets, is based on principles of duplicate packet elimination in [LO02]. In the present solution, the existing *Identification* and *Fragment Offset* fields of the IPv4 header are used instead of the *IP sequence number* introduced in [LO02], thus avoiding additional information to be included in the IPv4 header.

The value of the 16-bit *IP Identification* field in the IPv4 header is incremented each time a node generates a new IPv4 packet. In case fragmentation is required, all fragments of a packet have an identical value copied into the IP Identification field. Each fragment of a packet has the "more fragments" flag set, and the "Fragment Offset" field contains the offset of the present fragment from the beginning of the original IPv4 packet [Ste94].

Each node holds a history of tuples of the form $\langle$*Originator, Destination, IP-Identification, Fragment-Offset* $\rangle$, one tuple for each multicast packet the node has received.

When a node receives a multicast IPv4 packet, a lookup in the node's packet history is performed to determine whether the packet has been received before. If this is the case, the packet is silently dropped. Otherwise, an entry for the packet is added to the history, and the packet is further processed for routing and/or delivery to applications.

2.3.3 Packet history size

As the history of received packets introduces storage requirements on each node, it is relevant to estimate the amount of storage required. The following scenario lets a node receive the maximum possible rate of unique packets per channel in an IEEE 802.11 network:

- the receiver is equipped with a number of 802.11 wireless interfaces, each operating at different of 11 Mbps channels,
- one traffic source for each different channel exists, and
- the bandwidth of each channel is consumed entirely by multicast IPv4 packets with zero-length payload, destined for a single multicast group.

The packet rate for a single channel is now calculated. To simplify the calculation, packet processing delay is not considered. This does, indeed, not decrease the rate with which packets can be delivered, and hence handled by the duplicate elimination process.

An empty IPv4 packet is mapped onto a single IEEE 802.11 MAC frame, resulting in a total of 38 bytes to be transmitted (20 bytes IPv4 header, and 18 bytes MAC frame overhead). With an inter-frame gap of 92 μsec, this results in packets being transmitted at a rate of 8359 packets per second [soc99].

As no fragmentation is expected to be present due to the packet size of 38 bytes, the 16-bit IP Identification value will wrap after 7.84 seconds. Thus, it does not make sense to store more than 7.84 seconds, or 65535 packets, of history.

Each tuple of the format proposed in section 2.3.2, can be stored using 12 bytes, requiring a total of 767 KB for sequence number history per available channel. In addition to this, an overhead may be introduced by the data structure used for the packet history.

2.4 Summary

Due to the procedure for mapping IPv4 multicast and broadcast addresses to MAC addresses, the destination MAC address of these packets is not changed during forwarding. Hence, duplicate elimination is required in situations where a node receives duplicates of the same IPv4 packet.

A solution based on packet history enable duplicate elimination using the information in the IPv4 header. Each node records tuples uniquely identifying all received multicast and broadcast packets, and performs lookup in this time-constrained history, discarding any packets received more than once within the history timeout period. A worst-case calculation shows that the proposed solution results in a maximum storage requirement for a single node of 767 KB raw packet history information per available channel.

In the following chapter, broadcast protocols and generic protocol extensions are selected for evaluation. The protocols all rely on duplicate elimination to be performed, hence being obvious targets for application of the solution just presented.

Chapter 3

MANET Broadcast Protocols

This chapter describes algorithms to perform broadcast in MANETs, and presents the selection of broadcast protocols to be evaluated through the simulation-based study described in chapter 5.

Section 3.1 introduces the overall problem, and section 3.2 presents a generic broadcast algorithm which can be extended to accommodate the behaviour of any MANET broadcast algorithm. Section 3.3 present a selection of broadcast protocols and generic extensions to be evaluated in this work. The evaluation strategy described in section 3.4 estimates the number of simulations necessary for the evaluation, and section 3.5 summarises the chapter.

3.1 Introduction

Regarding a MANET as a directed graph (where nodes are vertices, and links are edges), allows the problem of broadcasting a message to be restated as "propagating a message from one vertex to the remaining vertices via the available edges".

Reductions in the bandwidth overhead of broadcasting can be achieved by pruning the set of vertices through which a packet is forwarded. The pruning can be done by a *broadcast algorithm* that determines a set of nodes for broadcasting a packet. As illustrated in figure 1.2, an optimal reduction of the bandwidth overhead is achieved by using the nodes in the minimal connected dominating set for broadcasting [SK96, CL02, AWF02].

To be applicable in the context of MANETs, such an algorithm should operate in a completely distributed fashion, using information available locally at each node.

With the goal of increasing the probability of delivering a packet to all nodes in the MANET, the broadcast protocols can be combined with several generic extensions.

3.2 A Generic Broadcast Algorithm

Packets intended for processing by a MANET broadcast algorithm must be recognisable by their destination address. To avoid conflicts between the semantics defined for the Limited and Directed Broadcast addresses for wired networks, a specific address for broadcast in MANETs is introduced. This address is denoted *All-Manet-Nodes*.

A generic algorithm for broadcast of packets can be specified as follows:

A node forwards a packet at most once within a limited amount of time, if and only if the following conditions are fulfilled:

1. the packet is destined for the *All-Manet-Nodes* address, *and*

2. the packet has not been received (or alternatively, forwarded) earlier, *and*

3. no other constraints specific to the forwarding algorithm in use prevent the packet from being forwarded.

Notice that the time constraint present in the algorithm above, originates from use if the duplicate elimination mechanism described in chapter 2. Each of the above conditions will be described in detail in the following sections 3.2.1– 3.2.3.

3.2.1 Condition 1: Addressing

Condition 1 states that a message which is to be broadcast must be destined for the *All-Manet-Nodes* address. Introducing *All-Manet-Nodes* as an address different from the Limited Broadcast address, requires the selection of the correct address, to reach the intended set of receivers. Two solutions are possible:

- applications are aware of the semantic difference between the Limited Broadcast and *All-Manet-Nodes* addresses, and selects the correct one according to their needs, *or*

- a mapping between the Limited Broadcast address and the *All-Manet-Nodes* address is performed automatically, making every Limited Broadcast message reach all MANET nodes.

The first solution maintains the existing neighbourcast semantics of the Limited Broadcast address, while the second solution effectively removes the possibility for neighbourcast via the Limited Broadcast address. For the purpose of this work, the first solution is selected.

3.2.2 Condition 2: Duplicate Packets

Condition 2 of the generic algorithm allows two solutions to be applied for eliminating duplicate packets, differing on whether duplicate packets are eliminated if they have been *received* or *forwarded* before.

The first solution is denoted *simple duplicate elimination*, involving only information about whether a packet has been received before. The second solution, denoted *extended duplicate elimination*, eliminates duplicate packets using information about the forwarding status of the packets. By handling one case differently, this solution allows more packets to be forwarded.

Consider the situation illustrated in figure 3.1: a packet which can traverse two paths, a and b, to reach the same node, N. Path a delivers the packet first, but due to a routing constraint in condition 3, N forwards the packet only if it is received via path b. Here, the "previously forwarded" information can be utilised: Though N has already received the packet, it may still forward it, possibly reaching uncovered nodes.

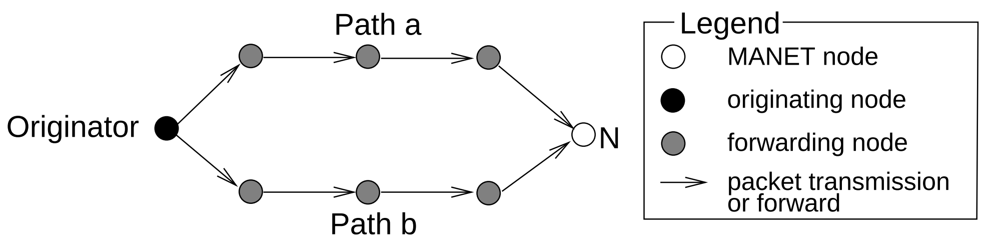

Figure 3.1: Paths a and b both delivers a packet to node N. The packet via path a arrives first, but only the packet travelling via path b may be forwarded by N.

Information about whether the packet was previously forwarded can be applied to determine whether the packet should be forwarded, resulting in the decision process illustrated in figure 3.2 when a packet is received.

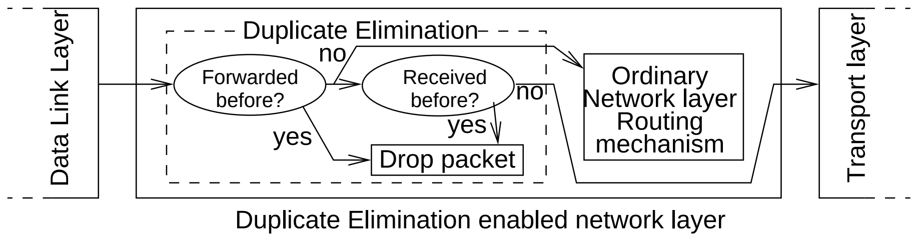

Figure 3.2: Extended duplicate elimination, taking "previously forwarded" information into account.

In general, it is desirable to keep the amount of stateful information in routers at a minimum, as maintenance and use of such information for the purpose of forwarding decisions implies processing and storage overhead. The introduction of duplicate elimination require MANET nodes to apply stateful information when deciding the fate of packets, i.e., the history of packet identifiers.

Furthermore, the situation handled by the extended duplicate elimination scheme does *not* appear in all broadcast protocols. One property must be present in the third condition in the generic broadcast algorithm is required, namely that decision of forwarding a packet depends on the path traversed by the packet so far. Hence, the simple duplicate elimination scheme will be used for all the broadcast protocols evaluated in this work.

3.2.3 Condition 3: Protocol-specific Optimisations

Recalling figure 1.2, a broadcast protocol may reduce the number of forwards required to broadcast a packet. Condition 3 of the generic algorithm leaves room for exactly such protocol-specific optimisations, as any additional rules may be added here.

3.3 Broadcast Protocols

As stated in section 1.6, one of the main goals of this work is to extensively evaluate a set of broadcast protocols. The protocols should lend themselves to implementation with a minimum of intrusion, and minimal system requirements, apart from the capability to forward packets.

In the following sections, the protocols to be evaluated are selected. First four different classes of broadcast protocols are described, to determine if their system requirements can be fulfilled by the constraint of minimal intrusion and system requirements. With this background, the specific broadcast protocols, and a set of generic extensions which will likewise be subject to evaluation, are selected.

3.3.1 Protocol Classes

Four classes of MANET broadcast protocols are defined by [NTCS99, WC02]. A short description of the implementation requirements for each of the protocol classes is given. Following, one class of protocols is selected for further examination.

Probability based protocols apply a very simple scheme for achieving broadcast: upon receiving a packet, the receiver forwards the packet with probability p ($0 \leq p \leq 1$).

Picking a random value between 0 and 1 for each packet being forwarded appears feasible at first sight. However the basic features, provided by traditional router implementations – including those of MANET nodes – negates this: a set of forwarding rules are applied to all received packets, providing no "per-packet-configurable" rules.

Counter based protocols buffer received packets. During the buffering period, the receiver counts the number of duplicates received for each buffered packet. If the number of received duplicates is below some threshold, the buffered packet is forwarded, otherwise it is silently discarded.

Queues that store packets for immediate routing may be present, intended to buffer bursts of packets that arrive at a faster rate than they can be processed – not for general-purpose use by routing protocols. Also, buffering packets in transit exposes the sending and receiving applications to possible effects of the behaviour of the buffer, and increases the end-to-end delay for each packet. Both effects may be harmful to the performance experienced by the communicating applications.

Location and distance based protocols require information about the locations and physical distance between nodes respectively, to calculate the additional coverage achieved if a broadcast packet is forwarded. Only if the coverage is above some threshold, the packet is forwarded. The location or distance information can be retrieved by GPS, but the requirement that all MANET nodes are equipped with GPS receivers is undesired, both due to the cost, and because the GPS radio signals cannot propagate in certain conditions, e.g., inside buildings.

An alternative method for calculating the distance between nodes is proposed in [DRWT97]: a simple model for radio signal propagation is applied to calculate the distance based on the received signal power. This simple approach has limited fidelity, as radio signal propagation depends highly on the physical surroundings, which are not described expressed in the signal propagation model.

Neighbour knowledge based protocols require a method to retrieve topology information about the network in which they operate. These protocols rely on the actual connectivity in the MANET when deciding to forward packets. Each node computes forwarding rules periodically

> from its local topology information, obtained through exchange of control messages between nodes.
>
> Having computed forwarding rules, these are installed in the routing table, where they remain until updated at a later point in time.
>
> Standard MANET nodes do provide the necessary functionality for neighbour knowledge based protocols to exchange control messages and manipulate the node's routing table.

The class of neighbour knowledge based protocols is the only class not requiring additional hardware or router functionality than what is already present on basic MANET nodes. Thus, protocols in this class of protocols is considered for further examination.

3.3.2 Selecting Broadcast Protocols

This section concerns the selection of broadcast protocols. Through the examination of different protocol classes, it has become clear that the protocols selected for evaluation in this work should be from the class of neighbour knowledge protocols.

Several MANET routing protocols are based on neighbour knowledge. Nodes using these protocols maintain a local base of network topology information, from which routes are calculated. Of these, only OLSR uses broadcast to disseminate topology information. For this reason its broadcast mechanism is subject for evaluation.

In addition to neighbour knowledge based protocols, other existing broadcast protocols, which do not suffer from implementation issues, are included, to achieve a comprehensive comparison.

MPR Flooding

OLSR is the only proactive protocol utilising broadcast to keep the topology information in the network up-to-date. It does so in an optimised fashion, by selecting a subset of all nodes, the *Multipoint Relay* nodes, to perform forwarding of a given broadcast packet. Using the MPR nodes to perform data broadcast is certainly possible. Doing this, the broadcast protocol denoted "MPR Flooding" is obtained. The MPR Flooding protocol is derived from the generic broadcast algorithm described in section 3.2 by adding as condition 3:

> *"the packet is forwarded iff the receiving node has been selected as MPR by the sender of the packet."*

Determining the fate of a packet using this condition, requires information about the IP address of the *sender* of each packet. This is not problematic for OLSR messages, as the messages are broadcast via MPRs using Limited Broadcast, originating new packets at each hop that the broadcast packets traverse.

For arbitrary data packets, the IP header only provides information about the *originator*. The sender address can be included as an *IP Option* field in the IP header, and updated upon each forward of the packet.

Though possible, this is not a practical solution, since it requires changes in the standard behaviour of the IP layer, and incurs additional processing overhead when packets are forwarded. The Dominating Set Flooding protocol, described next, remedies this problem.

Dominating Set Flooding

Dominating Set Flooding provides a mechanism, equivalent to MPR Flooding in terms of the set of forwarding nodes, while avoiding the requirement for sender information in the IP header.

A dominating set of a MANET topology a set of nodes selected such that every other node is adjacent to at least one node in this set. Observing that OLSR performs periodic broadcast of TC messages using the MPR optimisation is the key to achieving Dominating Set Flooding, as the set of MPR nodes broadcasting a TC message sent by node N constitute a dominating set.

Broadcasting data packets via this dominating set can be accomplished by adding the following requirement as condition 3 of the generic broadcast algorithm:

> *"the packet is forwarded iff the receiving node has forwarded the last TC message originated by a certain node N."*

Various selections of N can be applied; what is crucial is:

- that the selected node is an MPR (otherwise it does not send TC messages), *and*
- that all nodes agree upon the selection.

Two different ways of making the selection have been considered for the purpose of this work:

1. Use the MPR with the lowest ID present in the MANET.
2. If the originator is an MPR itself, use its own ID. Otherwise, use the lowest ID among the originator's MPRs.

The first solution makes all broadcast packets traverse a common shortest path tree (SPT), concentrating the traffic load to that SPT. If the selected tree is temporarily "broken", this affects all broadcast operations.

Further, the SPT for the node with the lowest ID may not be the SPT for the node which originates the data (see Appendix B for an example of this situation).

In the second solution, packets originated at different nodes may traverse different SPTs, so that outdated knowledge about one ID selection does not influence broadcasts using other ID selections. Furthermore, the tree defined by this ID selection will be a SPT rooted at the originator node, or that of it's MPRs which has the lowest ID.

Reverse Path Flooding

Reverse Path Forwarding [DM78] has been applied in wired networks, e.g., for the PIM multicast protocols [pim02]. This protocol uses information about shortest paths, provided by an underlying unicast routing protocol, to make its forwarding decisions. Using Reverse Path Forwarding as a broadcast protocol, for the purpose of this work named "Reverse Path Flooding", is possible by adding the following as condition 3 in the generic broadcast algorithm:

> *"the packet is forwarded iff it is received from the node selected as next hop on the shortest path to the originator of the packet."*

The information about next hop nodes is provided by the unicast routes computed by a unicast shortest-path routing protocol. For the purpose of this work, OLSR is selected to provide unicast routing information.

Using Reverse Path Flooding in a MANET where shortest-path routes have been established, and where no packet drops occur (an ideal situation), all nodes will eventually receive a broadcast packet from the originator node itself, or the node selected as next hop towards the originator. Hence, all nodes will forward the packet, a situation which most likely includes unnecessary forwards [NTCS99].

Classic Flooding

The reactive MANET routing protocols DSR and AODV use broadcast to disseminate control messages when discovery of new routes is required. Both protocols use Classic Flooding to perform this task (see the example of Classic Flooding in figure 1.2b).

Classic Flooding is not in the class of neighbour knowledge based broadcast protocols. However, it is included in this study, to cover the of broadcast protocols used in MANET routing protocols. For Classic Flooding, *no* additional constraints are added in condition 3 of the generic broadcast algorithm. Only conditions 1 and 2 are applied for this protocol.

The previously selected broadcast protocols base their operation on the topology information provided by OLSR. Being a proactive routing protocol, OLSR specifies periodical transmission of messages, in order for each node in the MANET to maintain its knowledge about the network topology.

This introduces a communication overhead which may influence the performance of Classic Flooding, as the unicast and broadcast protocols must share the available bandwidth. Therefore, the Classic Flooding protocol is evaluated both as a stand-alone protocol, and in situations where also the OLSR unicast protocol is used.

3.3.3 Protocol Extensions

In addition to the four protocols selected for evaluation, two extensions are included. Each of these extensions can be used in conjunction with each of the selected protocols.

Data Packet Jitter

[CHCB01] shows through simulations that adding a small, random amount of jitter when forwarding OLSR control messages increases the delivery rate. The jitter reduce the chance of collisions when packets are forwarded, hence causing fewer packet drops. Assuming that the jitter may improve the delivery rate for data packets as well, the effect of using jitter of various scales when forwarding data packets will be evaluated.

It is noticed that adding jitter when forwarding OLSR control messages lends itself to straightforward implementation: processing of OLSR control messages is independent of the functionality of the network layer in the IP stack. Adding jitter to data packets is less trivial: after the forwarding rules has been configured, the network takes care of packet forwarding. Changes must be made in the network layer to add generic support for jitter when forwarding broadcast packets.

Multipacket Flooding

The goal of broadcast may be hard to fulfil under the presence of packet loss. When broadcasting in a MANET subjected to small traffic loads, the idea of using the additional available bandwidth to achieve a better delivery rate can be realised.

Early versions of TBRPF specified the transmission of multiple copies of the same control message to increase the chance of reliable delivery [BOT01]. This idea can be transferred to broadcast of data as well: to achieve better delivery rate, at the cost of extra bandwidth consumption, multiple copies of each originated packet may be sent by the originating node. To describe this behaviour, the notion of *"Multipacket Flooding"* is introduced.

The Multipacket Flooding algorithm has one tunable parameter, the *multipacket multiplier* denoted m, and proceeds as follows when a node *originates* a packet:

1. transmit the original packet, and immediately hereafter,
2. transmit $m - 1$ copies of the original packet,

This solution may result in excessive numbers of collisions when forwarding packets, as "bursts" of packets are sent across the network, and two nodes sending such bursts at the same time is likely to cause a collision for each sent packet.

3.3.4 Summary of selected protocols

Based on the criteria of straightforward integration in MANET nodes, four broadcast protocols have been selected, and considerations regarding their operation has been described. The protocols are as follows:

- Multipoint Relay Flooding,
- Dominating Set Flooding,
- Reverse Path Flooding, *and*
- Classic Flooding (with and without OLSR)

Furthermore, two extensions have been specified, to be evaluated in conjunction with a selection of the broadcast protocols.

- Data Packet Jitter, *and*
- Multipacket Flooding

The following section describes the strategy for evaluating the protocols, and presents an estimate of the number of simulations required to perform the desired evaluation.

3.4 Evaluation Strategy

In order to conduct an extensive simulation based study of the four selected protocols, a strategy for achieving the desired results is presented. Two parameters are, in turn, considered:

1. the conditions under which the protocols will be evaluated, defined by a set of dimensions of a given granularity, and
2. the number of simulations required to present statistically significant results for the protocols.

Based on the these parameters, the number of simulations to be conducted can be estimated.

3.4.1 Dimensions and Granularity

The goal of simulation based study in this work is to evaluate the selected broadcast protocols in a wide variety of conditions. Several descriptive scenario parameters, e.g., traffic load, number of nodes, scenario size and mobility can be varied, to achieve scenarios that expose the broadcast protocols to different conditions.

The simulations will be focused on evaluating the broadcast protocols in a wide range of load conditions. One fixed, large scenario scenario size is used, in which a fixed number of mobile nodes move around with random mobility rate.

Variable load conditions

The traffic load imposed on each scenario will be generated using constant-bit-rate (CBR) traffic sources. Varying load conditions are achieved by using different numbers of streams and by varying the byte rate of each stream.

To achieve results from which tendencies of the protocol performance can be observed, a variety of configurations for both the number of streams and the stream byte rate will be evaluated. Several configurations of the tunable parameter (the jitter interval or the multipacket multiplier) are simulated. This allows the results to be compared with the corresponding results for the non-extended broadcast protocols, to observe the effect of the extension.

Tuning of extensions

Each of the two extensions presented in section 3.3.3 has a tunable parameter. An evaluation of the extensions will be performed as follows: the extension is combined with a selection of the broadcast protocols, and simulated in a subset of the load conditions in which the original broadcast protocol has been simulated.

3.4.2 Statistical Significance

Due to different node mobility and traffic patterns, each of the scenarios generated by *wsg* may yield different performance results. Therefore, a number of random scenarios generated from identical parameter configurations are simulated to achieve the average value of each sample point.

According to [Wag92], a minimum sample size of 30 is necessary to achieve representative averages for values of a population. For the purpose of this work, 30 scenarios are simulated for each configuration of a broadcast protocol (possibly combined with one of the proposed extensions), number of CBR streams, and stream byte rate.

3.4.3 Estimated Extent of Simulations

The number of simulations involved in the desired study is estimated by deciding on the granularity of the variable parameters. For this purpose, the following terms are defined:

Traffic source configuration: *a selected number of nodes having active CBR streams at any instant during the simulation.*

Byte rate configuration: *a selected rate with which the active CBR streams transmit data.*

Jitter interval configuration: *a selected range limiting the values that can be attained when selecting a random amount of jitter.*

Multiplier configuration: *a selected value of the tunable parameter "m" in the multipacket flooding extension.*

Table 3.1 illustrates the granularity of two variable parameters for the simulations of the basic protocols. The numbers in this table results in a total of 350 sample points, and with a sample size of 30, this yields a total of 10.500 simulations.

Additionally, it is desired to examine whether the time-constrained duplicate elimination history size has any effect. It may be the case that some duplicates go undetected if received after their information has timed out. For this purpose, a total of 600 simulations are defined, as illustrated in table 3.2.

Protocol	**Traffic source configurations**	**Byte rate configurations**
MPR Flooding	10	7
Dominating Set Flooding	10	7
Reverse Path Flooding	10	7
Classic Flooding	10	7
Classic Flooding with OLSR	10	7

Table 3.1: Estimated number of simulations for evaluation of the broadcast protocols.

Protocol	**Traffic source configurations**	**Byte rate configurations**
MPR Flooding	10	1
Dominating Set Flooding	10	1

Table 3.2: Estimated number of simulations for evaluation of duplicate elimination.

Table 3.3 illustrates the granularity of the variable parameters for the simulations of the jitter extension. 160 sample points are defined, for each of which 30 simulations are conducted, yielding a total of 4.800 simulations.

Protocol	**Traffic source configurations**	**Byte rate configurations**	**Jitter interval configurations**
DS Flooding & Jitter Extension	10	1	8
Classic Flooding & Jitter Extension	10	1	8

Table 3.3: Estimated number of simulations for evaluation of the jitter extension.

Table 3.4 illustrates the granularity of the scenario parameters for the Multipacket Flooding simulations. Defining a total of 180 sample points, each being simulated in 30 scenarios, this test yields a total of 5400 simulations.

The four categories of simulations add up to 21.300 simulations in all. This exceeds by far the numbers of simulations conducted in previous work presenting simulation-based studies, and motivates an automated approach to configuring and executing the simulations, and for visualising the results.

Protocol	**Traffic source configurations**	**Byte rate configurations**	**Multiplier configurations**
DS Flooding & Multipacket Ext.	10	3	3
Classic Flooding & Multipacket Ext.	10	3	3

Table 3.4: Estimated number of simulations for evaluation of the multipacket flooding extension.

3.5 Summary

The bandwidth overhead of broadcasting a packet to all nodes in a MANET can be reduced by minimising the number of forwards required to deliver the packet to all nodes in the MANET. This reduction is one of task of the broadcast protocols considered in this chapter.

A generic broadcast algorithm provides the basis for specification of the broadcast protocols to be evaluated in this work. Four classes of broadcast protocols are described with focus on investigating their implementability, and the class of neighbour knowledge based broadcast protocols is selected for further examination.

Four broadcast protocols are selected for evaluation. The three are based on neighbour knowledge, and the fourth being a traditional approach to broadcasting, present in two of the existing unicast MANET routing protocols.

Finally, an estimate of the number of simulations required to conduct the study performed in this work is estimated. Conducting the large quantity of simulations (21,300 in all) motivate minimising the work involved in conducting the simulations. The development of a simulation framework for this purpose, as suggested in section 1.5.1, is the subject of the next chapter.

Chapter 4

Simulation Framework

This chapter describes the simulation framework developed in this work, intended to aid the conduction of extensive amounts of simulations. Section 4.1 introduces the concept of a *framework* used to structure the desired simulation. Section 4.2 gives an overview of the framework, and section 4.3 describes its individual components. The chapter is summarised in section 4.4.

4.1 Introduction

The framework developed in this work divides the simulation process into a set of well-defined stages, arranges the data storage required by each stage, and integrates a set of applications specific to each stage, into a coherent structure that allows each application to be exchanged individually if desired.

The following section gives an overview of the platform available for this work, to clarify the requirements is put forth for portability.

4.2 Framework Overview

The simulation framework is divided into a set of discrete stages, and different applications are applied to complete the tasks for each stage. Section 4.2.1 gives an overview of the platform on which the simulations in this work are to be conducted. Section 4.2.2 describes the stages and their relation, and section 4.2.3 describes the organisation of the data storage for these stages.

4.2.1 Simulation platform

Three classes of machines constiture the platform onto which the simulation framework is applied. The classes are characterised by different architectures, operating systems and storage organisation, as well as different usage characteristics:

Cluster machines: a cluster of seven Linux/Intel machines with limited, shared disk storage in an isolated environment. The cluster is used by a small number of other researchers and students. This category delivers approximately half of the total computation resources.

Networked, shared machines: 15 Application servers and workstations (Solaris/Sparc and Intel) with limited, shared disk storage. These machines are accessible to a large body of researchers and students. The application servers are powerful, but occasionally very loaded. This category delivers approximately one quarter of the total computation resources

Independent machines: four Independent workstations (Linux/Intel) with non-shared, larger disk storage, typically very lightly loaded. This category delivers the remaining computation resources.

Notice that the environment is heterogeneous, with respect to both platforms and usage characteristics, and provides limited disk and computational resources. The requirement for portability influences the selection of languages for implementing the framework applications. Due to limited storage and computation resources, considerations regarding data storage and saving of possible recomputations are made during the framework design.

4.2.2 Simulation Stages

As illustrated in figure 4.1, the simulation framework consists of five main classes of utilities, sharing a common data storage for reading and writing their data. The five classes define five stages of the process leading from abstract scenario descriptions to a set of results visualised as graphs.

Scenario Generation: The first stage converts abstract scenario descriptions to a set of concrete scenarios which span the (multidimensional) space of characteristics specified at the abstract level. This stage is accomplished using a slightly modified version of *wsg* [CHCB01].

Simulation: The second stage contains both the execution of individual simulations, and the task of performing job control and scheduling on the possibly large quantity of simulations. The network simulator ns2 is applied to conduct the simulations, and a custom job scheduling system is developed to distribute the execution of simulations among multiple machines.

Trace File Analysis: The third stage operates on the raw simulation output (trace files). At this stage, operations are performed varying from

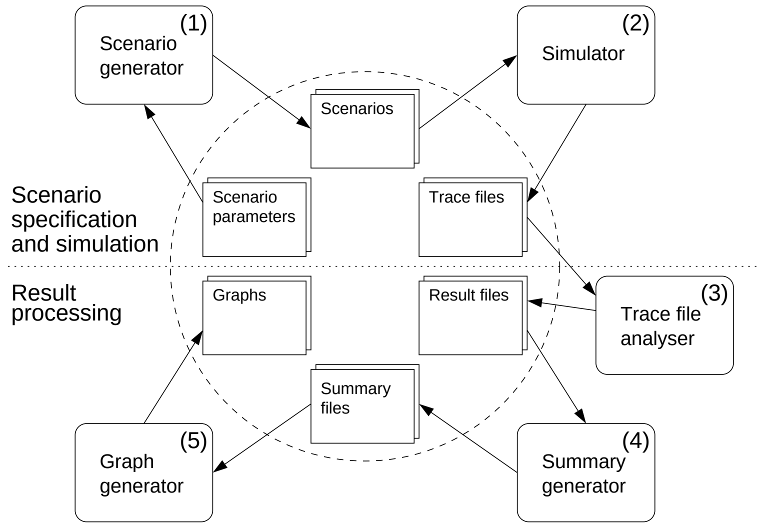

Figure 4.1: The five stages of the simulation framework, and their inputs and outputs. Above the dotted line are the stages generating the most detailed level of results. Below the dotted line are the stages that aggregate the results.

simple quantitative measures to applying advanced statistical tools, all on the output of a single simulation. A tool maned *Tafat* (short for TrAce File Analysis Tool) is developed to accomplish this stage.

Summary Generation: The fourth stage operates on the results of the trace file analysis to generate summaries of the results of several simulations having common characteristics. A tool named *Sump* (short for SUMmary Processor) is developed to accomplish this stage.

Graph Generation: The fifth and final stage makes the summarised results more accessible to users, by visualising all the results in graphs. A comprehensive set of graphs opens possibilities for discovering interesting results not in the initial scope of interest. A tool named *Grace* (short for GRAph Compilation Environment) is developed to accomplish this task.

The scenario parameters are the input to the framework, and the graphs are the output. However, the outputs of stages three, four, and five are a gradual refinement of the simulator output in which each step adds overview and accessibility at the expense of detail. Therefore, the combined outputs of the last three stages are considered the final output of the framework, though the graphs are the most human readable representation of the results.

4.2.3 Data Storage

The organisation and selection of the data stored between the framework stages illustrated in figure 4.1 must be structured to ensure flexibility, and stored under consideration of the available processing and storage resources. These issues are the subject of the present section.

Common file format

The output of trace file analysis shares many properties with the output of summary generation. For this reason, these two stages share a common output file format. This is convenient as it enables use of a common parser front end for summary generation and graph generation stages.

It is desired that the files are easy to parse, and still human readable (i.e., not binary) to help debugging. To achieve this, a format is used where a file consists of a number of lines, where each line consists of a number of fields separated by colons (“`:`”). Also, a line may not depend upon the existence or contents of other lines, i.e., all the context information necessary must be present in the line itself. A line has the following structure:

```
<module>:<variable>:<meta-data>:<unit>:<data>
```

- `<module>` is the name of the statistics module that generated the data, and `<variable>` is the name of a particular measure within the module.
- `<unit>` is a text string describing the unit of the data in this measure (e.g. packets, bytes, seconds, etc.).
- `<meta-data>` and `<data>` are lists of tuples separated by bars (“`|`”), each such tuple consists of a comma-separated list of fields, as follows:

`<descriptor`$_1$`,` ... `,descriptor`$_n$`,value>|` ...

The `<data>` field in the output of the trace file analysis usually consists of a single “`<descriptor,value>`” tuple. The summary generator, on the other hand, will need to compute aggregated values. Thus, the `<data>` field tuples of the summary generator output lines are qualified with additional descriptors, for example:

```
<descriptor,"avg",value>|<descriptor,"max",value>| ...
```

Selecting data for permanent storage

After completion of all tasks in the framework, the user is supplied with graphs displaying the results obtained by simulations of the initial abstract scenario description.

It is not unlikely, that some of the results requires further inspection. In such situations it would be advantageous to have as many intermediate data available as possible, allowing the results to be inspected at more fine-grained levels, or even allowing only part of the stages to be re-run. Finding a reasonable set of intermediate files suitable for permanent storage involves three considerations which will be described in turn.

The cost of generating and storing the results: Table 4.1 shows typical processing times and data volumes for each phase in the simulation framework. From this table, it is observed that *simulation* is the most heavy phase, both with respect to processing time and output data volume. The remaining phases have requirements of relatively smaller scales both with respect to data volumes and processing times.

Task	**Typical output data size**	**Typical processing time**
Scenario generation	Hundreds of KB	Seconds
Simulation	Hundreds of MB – Several GB	Minutes – several hours
Trace file analysis	Less than 100 KB	Minutes
Summary generation	Less than 100 KB	Seconds
Graph generation	Hundreds of KB	Seconds

Table 4.1: Data volumes and processing times (estimated for a 500 MHz Pentium II PC) for conducting the phases of a single simulation.

Reproducibility of the results: It is observed that if a stage involves nondeterministic behaviour (e.g., random without seeds), the output of this stage cannot be recreated from its input. Hence, whenever nondeterminism occurs, it is necessary to store the intermediate files from the stage involved or some subsequent stage. Among the applications used for the framework in this work, only the scenario generator, *wsg*, involves nondeterminism.

Information loss through aggregation: It may be desired to store data before and after aggregating them. Figure 4.2 shows the data associated with a single simulation, and the ones aggregated from several simulations. Results are aggregated in both the trace file analysis, summary generation and graph generation phases.

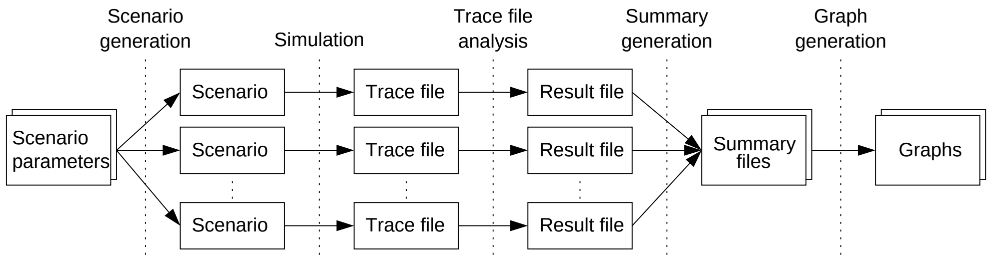

Figure 4.2: Some data are associated with just one simulation, whereas others are aggregated from multiple simulations.

With these considerations in mind, each file type present in the simulation framework is now considered for permanent storage.

Scenario parameters: Though the scenario generator involves nondeterminism, and thus the scenario parameters are insufficient to recreate the simulation results, the scenario parameters are stored, as they are the only non-simulation-specific data that exist before the simulator stage. Furthermore, the scenario parameters provide an abstract description of the conditions expressed in all scenarios of a set of simulations.

Scenarios: The scenarios are the origin of the results of a specific simulation: simulating the exact same scenarios twice results in identical trace files. As the scenario generator is nondeterministic, the scenarios are necessary if the simulations need to be re-run. Hence, they are stored permanently.

Trace files: Since all later stages are deterministic and aggregating, their outputs can all be recreated from the trace files, and at a relatively low cost (see table 4.1). However, with sizes ranging from hundreds of MB to several GB, it is not practical to store the trace files.

Result files: The result files differs from the trace files in that instead of logging events, events of each type are simply counted. The main loss of information in the transition from a trace file to a result file is the temporal dimension. The result files are selected for permanent storage, as they are the closest representation of the trace file contents available.

Summary files: When multiple simulations with identical scenario parameters are conducted, summary contain collect aggregated values, such as average, maximum, minimum, etc. of the values in the result files. It is not time consuming to generate summaries from the permanently stored result files, but their small data volume allows them to be permanently stored without problems. For the purpose of this work, it is selected to store the summary files.

Graphs: Graphs maintain a very low level of detail in order to present as much information as condensed as possible to the user. The summary data can not be generated from the graphs, speaking against storing the graphs permanently storage. However, the graphs do not require much disk space, and they are likely to be the most frequently accessed presentation of the simulation results. Hence, the graphs are included in the permanent storage.

Having considered the file types present in the simulation storage, only the trace files are excluded from permanent storage, due to their large size: each simulation generates trace files of up to several GB.

4.3 Framework Components

This section describes in further detail each application used in the simulation framework which will is to be applied in this work. As described in section 4.2.2, the scenario generator and simulator are modified versions of existing applications [CHCB01, LO02]. This section presents more elaborate descriptions of the remaining tools, as they have been developed during this work.

Notice that a separate description of the *simulation scheduler* is present in section 4.3.3 for readability, although this tool rightly belongs to the simulation phase in the framework.

4.3.1 Scenario Generator

A slightly modified version of the scenario generator used in [LO02] is used to generate the wireless broadcast scenarios. As the scenario generator is divided into a front end and a back end, interfaced via a generic file format, it is necessary to change only the back end to achieve scenarios that has the syntax required to utilise the new wireless broadcast functionality required for the simulations conducted in this work.

A set of shell scripts have been written to manage the procedure of quickly setting up large sets of scenarios. A template for the scenario files (see figure 4.3a) is read by the scripts, and fields in the template are replaced with values for, e.g., number of streams, stream byte rate, and so forth. After all the fields have been substituted, the parameter file (see figure 4.3b) is stored to disk. *wsg* generates the desired number of scenarios, typically 30, of the form exemplified in figure 4.3c. These scenarios are stored to disk, ready to be simulated by ns2.

(a)
```
# Node mobility (m/s)
node_speed <node_speed>
# Multicast stream byte rate (bytes/sec)
A.mcast_stream_byterate <mcast_stream_byterate>
# Average number of active multicast streams
A.mcast_stream_sends <mcast_stream_sends>
```

(b)
```
# Node mobility (m/s)
node_speed 1-2
# Multicast stream byte rate (bytes/sec)
A.mcast_stream_byterate 2304
# Average number of active multicast streams
A.mcast_stream_sends 25
```

(c)
```
set node_(42) [$ns_ node]
$node_(42) set X_ 1093.049294
$node_(42) set Y_ 1353.005553
set wbc_(432) [new Agent/WBC/CBR]
$node_(42) attach-wbc-agent $wbc_(432) 0
$ns_ at 245.8052008 "$wbc_(432) start
$ns_ at 255.8052008 "$wbc_(432) stop"
```

Figure 4.3: Examples of the contents of (a) scenario template files, (b) scenario parameter files and (c) the scenario files generated by *wsg*.

4.3.2 Network Simulator

In order to carry out the intended simulations the simulator selected for this work, ns2, must support network-wide broadcast (referred to just as “broadcast”) in wireless networks. The current version of, ns2 [pro02] supports only limited broadcast, also denoted “neighbourcast”.

Successful broadcast of a packet in a wireless network results in the packet being delivered to every node in the network. As described in section 2.2, this process can be viewed as an instance of multicasting, using the *All-Manet-*

Nodes multicast address as destination for broadcast packets independently of the broadcast algorithm doing the packet forwarding.

Regarding broadcast as an instance of multicast is not only an abstract convenience. In a multicast-enabled MANET, it is possible to actually carry out broadcast as if it were multicast, at all layers below the network layer. The network layer must be modified to recognise the *All-Manet-Nodes* address, and subject packets destined for this address to broadcast routing rather than multicast routing.

Network-wide broadcast in ns2

Multicast support in ns2 is a prerequisite for broadcast through multicast to work. Multicast in wireless networks is, however, not supported by the current version of ns2. The broadcast-capable version of ns2 used for this work is based on the multicast-enabled version of ns2 developed in [LO02]. Here, support for wireless multicast was created by bridging the gap between the two already supported features of wired multicast, and wireless unicast.

To fulfil the requirements for the simulations to be performed in this work, further extensions to ns2 are created. The network layer is modified to intercept packets addressed to the *All-Manet-Nodes*, and deliver these packets to broadcast algorithms. These algorithms decide the fate of the packet instead of subjecting it to multicast routing.

Furthermore, the generic packet routing infrastructure of ns2 is updated contain special handling of packets destined for *All-Manet-Nodes*. Last, a set of traffic agents that send traffic to the *All-Manet-Nodes* address is created and integrated into the simulator.

Output from ns2

Figure 4.4 illustrates the trace file output generated by ns2 during a simulation. All lines are timestamped according to the internal simulation clock of ns2, and consist of a set of well-defined fields, from which information about events in the simulation can be extracted.

```
r 0.701335726 _5_ MAC -- 2 udp 1024 [0 80000004 10 800] ---- [73:-1 -2147483644:0 249 0]
D 0.701335726 _5_ RTR dup 2 udp 1024 [0 80000004 10 800] ---- [73:-1 -2147483644:0 249 0]
r 0.701335731 _53_ MAC -- 2 udp 1024 [0 80000004 10 800] ---- [73:-1 -2147483644:0 249 0]
```

Figure 4.4: Example of trace file output from ns2.

4.3.3 Job Scheduler

As described in section 4.2.3, the heaviest task with respect to computation is the simulation stage. The extensive amount of simulations to be conducted in this work, described in section 3.4.3, has motivated the development and use of a simulation scheduler, to perform parallel execution of the large set of simulations.

With the need for executing large batches of simulations on the platform described in section 4.2.1, one central point of management that allows automated job execution across the large number of machines is desired. A simple scheduling system is implemented using Ruby [Mat02], to provide exactly this functionality.

Scheduler structure

Figure 4.5 illustrates the structure of the scheduler, which consists of the central point of management, the job manager, and a set of job servers running on remote machines.

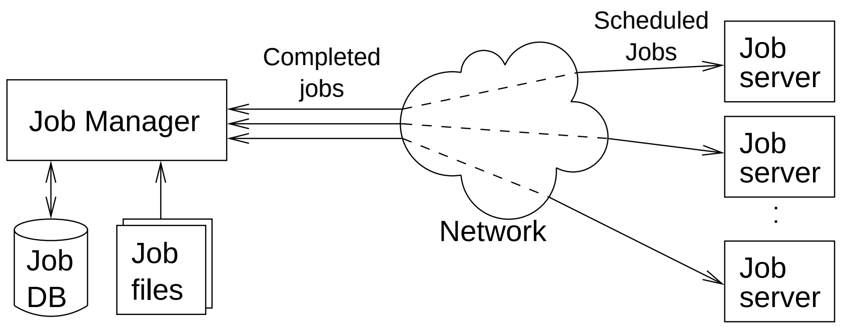

Figure 4.5: Structure of the simulation scheduling system.

Via the interface of the job manager, the scheduler provides functionality to establish and terminate connections to job servers, to manage the content of the job manager's job database, and to start and stop the scheduling of pending jobs.

For each connected job server, the job manager possesses information abut the effective user ID of the job server process, and schedules pending jobs according to the number of available CPUs, job priority and user IDs.

Defining and scheduling jobs

The job scheduler performs scheduling on a common pool of jobs, where each job consists of the information listed in table

Field	**Contents**
Command line	The command line which is executed on a job server when the job is scheduled.
CPU requirement	The number of CPUs which should be allocated when the job is scheduled
User	A regular expression matching the user names under which the job is allowed to be run
Host	A regular expression matching host names of the job servers allowed to run the job
Priority	Job priorities are integer valuesf in the range $[-19 \dots 19]$, inspired by UNIX process priorities (though completely independent hereof): jobs with low priority values are scheduled for execution before jobs with high values.

Table 4.2: Contents of a scheduler job.

Jobs are added to the job manager's queue of pending jobs from text files, containing one job definition per line. A job definition consists of a comma-separated list of fields following the order listed in table 4.2, like these example commands:

```
"ns scenario1.tcl", 1, tue, *, 0
"ns scenario2.tcl", 1, tue, *, 1
"tafat -s test.tfa -t test-trace.tr", 1, cbaoth, *, 0
```

Pending jobs are scheduled for execution according to the following simple procedure, which is executed when a job server with at least one CPU not allocated to execute a scheduled job is available.

1. From the set of all pending jobs, the set of jobs with potential to be scheduled are selected as those having host and username expressions matching those of the job server, *and* requiring at most the total number of idle CPUs on the server.

2. The set of potential jobs for scheduling is sorted according to the job priorities, and the highest prioritised job is executed.

The scheduler provides a generic platform for executing shell commands. This functionality lends itself to more than just executing ns2 simulations: For example, file management tasks can be carried out in a convenient manner across a number of computers, from one central location.

4.3.4 Trace File Analyser

The product of running a simulation in ns2 is a *trace file*, which contains timestamped information about a selection of events that occurred during the simulation, including packet transmissions, receptions, collections, etc. A dedicated application, *Tafat*, is developed to carry out the analysis of a trace file by gathering information through a single pass over the trace file, and writes it to a *result file*, of the form illustrated in figure 4.6.

```
forwards:pkt_fwd_rtr::packets:val,520934
forwards:pkt_send_agt::packets:val,23250
ifq_drops:ifq_drops::packets:val,161335
ifq_drops:ifq_drop_rate::pkt/s:val,537.783333
delay:pkt_delay_avg::seconds:val,22.8200
delay:pkt_hop_avg::hops:val,6.0357
```

Figure 4.6: Example of result file contents, generated by *Tafat*.

Before *Tafat*, `awk` [Rob02] and Ruby scripts were used to perform the trace file analysis, but both were found to incur an unacceptable performance overhead. Furthermore, lack of modularity in the `awk` scripts made it cumbersome to add new analysis scripts. *Tafat* is implemented in *C*, which adds possibilities for more modular structures than `awk`, while still allowing for performance optimisations over Ruby.

The primary objective of *Tafat* is achieving good performance, mainly by using existing knowledge of the structure of a trace file to avoid needless input parsing. The secondary objective is flexibility with respect to extending and configuring the tool. This is achieved through modularity and simplicity in design and implementation.

Structure of TAFAT

To address the requirement for modularity, the concept of *executables* is introduced. An executable defines a private scope, an initialisation function, a finalisation function, and a function that is executed once for each line in the trace file (denoted a “trace line”). Executables can express dependencies upon other executables, and may set functions or data at disposal for other executables to use. Thereby, tasks that are common for several executables may be extracted and placed in *utility* executables, thereby being run only once.

The executables have been divided into three groups: “pre-utils”, “analysis”, and “post-utils”. When an executable is initialised, it must specify in which group it belongs. The purpose of the grouping is to manage the order in which a trace line is passed to executables, as certain utility executables need to be run either before or after the analysis executables.

An example is the duplicate elimination utility. This utility, when it encounters a trace line regarding the reception of packet P by node N, registers that information. If an analysis executable subsequently asks whether (N, P) is a duplicate, the duplicate elimination utility answers “yes”. Clearly, for the first occurrence of (N, P) the answer must be “no”. Hence, it is important that the duplicate elimination utility belongs in the “post-utils” group, being run *after* the analysis executables.

Trace File Processing Flow

Figure 4.7 illustrates the flow of execution between the various components of *Tafat*, during the analysis of trace file data. Notice that before the analysis is started, all configuration of the trace file parsing module and the executables have been carried out. After reading and processing the last line of a trace file, all executables print their results to the result file.

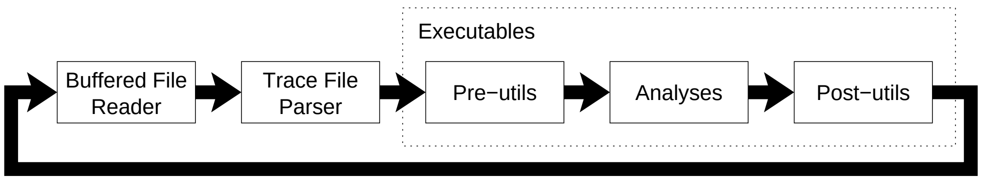

Figure 4.7: Execution flow of *Tafat* during trace file analysis, illustrated by the bold arrows.

Performance Considerations

To achieve good performance in terms of computation requirements, several optimisations are applied. Some error tolerance has been sacrificed in the trace file parser: the parser, operating at a line-by-line basis, assumes that an input line has the correct format, and thus avoids many sanity checks.

This is a sound assumption, since errors in the trace file format would be due to errors in the trace file generation code in ns2, hence requiring corrections under all circumstances.

The parser is configured by the executables at run-time, so that only the necessary fields in a trace line are extracted. Last, *Tafat* starts by reading a script file identifying what executables should be activated, to avoid running unnecessary executables, and parsing the trace file fields required only by unnecessary executables.

Furthermore, buffered reading of the input trace file is used. It has been determined through experiments that a buffer size of 32 KB yields good I/O performance.

4.3.5 Summary Generator

From the result files, summary files are generated using *Sump*, the Summary Processor. The generation of summary files distills the information from all simulations of a sample point one step further, thereby achieving data that are suitable for plotting in graphs. With the amount of results produced by the simulations conducted in this work, summary generation is the key to achieving useful graphical representation of the results.

The single most important goal for *Sump* is to conserve as much information as possible, to ensure that the summary files can be used for a variety of future examinations of the data. The task at hand is to generate one single summary file that summarises the results for a group of common measures present in all simulations in a single sample point. Referring to the specification of the file format for result files, measures from the result files are classified in groups having common values of the fields:

```
<module>:<variable>:<meta-data>:<unit>
```

All of these fields are used to specify information that allows the data associated with them to be interpreted in a particular way.

Summary Generation Output

Having collected all measures from the simulations performed in a single sample point, the summary processor calculates the average, standard deviation, minimum and maximum, and cardinality (number of measures) for each of the groups. This result is written to a summary file, using the same data format as the result files. The only difference is that an extra descriptor field is appended to the descriptor of the name of each data tuple, to indicate the type of summary statistics. An example summary generated from the result file from figure 4.6 is shown in figure 4.8.

4.3.6 Graph Generator

For a single measure, e.g., number of packet drops, one value for each sample point is present in the summary files. These values are to be plotted in

```
forwards:pkt_fwd_rtr::packets:val,avg,539165.9000|val,min,502593.0000|val,max,564779.0
                                        [...]
forwards:pkt_send_agt::packets:val,avg,23250.0000|val,min,23250.0000|val,max,23250.000
                                        [...]
ifq_drops:ifq_drops::packets:val,avg,175867.4667|val,min,144223.0000|val,max,205429.00
                                        [...]
ifq_drops:ifq_drop_rate::pkt/s:val,avg,586.2249|val,min,480.7433|val,max,684.7633|val,
                                        [...]
delay:pkt_delay_avg::seconds:val,avg,22.4239|val,min,17.6511|val,max,26.1643|val,stdde
                                        [...]
delay:pkt_hop_avg::hops:val,avg,6.3336|val,min,5.3914|val,max,7.0102|val,stddev,0.4554
                                        [...]
```

Figure 4.8: Example of output from the summary generator, *Sump* (The lines have been truncated to fit the page).

graphs, in order to present the simulation results in a form that is convenient for comparison of large sets of values.

An graph generator, *Grace*, is developed for this purpose, to provide a highly configurable environment for reading summary data and plotting graphs.

The philosophy of *Grace* is that while data may be read from several different sources, and it may be necessary to write them to different output formats, it should be possible to organise all data from a batch of simulations in one common storage structure (denoted a "sample space"), independent of the organisation of data in, e.g., summary files. The data is then extracted from the sample space for graph plotting. A set of simple queries enables data extraction without bothering about the previous organisation of data in summary files.

Phases of operation

Figure 4.9 illustrates the structure of *Grace*. Three separate phases, described in turn, make up the graph generation as a whole: Configuration parsing, input file reading and graph generation.

Configuration: The configuration phase sets up the sample space data structure, and initialises the summary file parser and output generators.

Summary reading: The summary reading phase parses all data from the summary files and stores it in the sample space. After parsing all files,

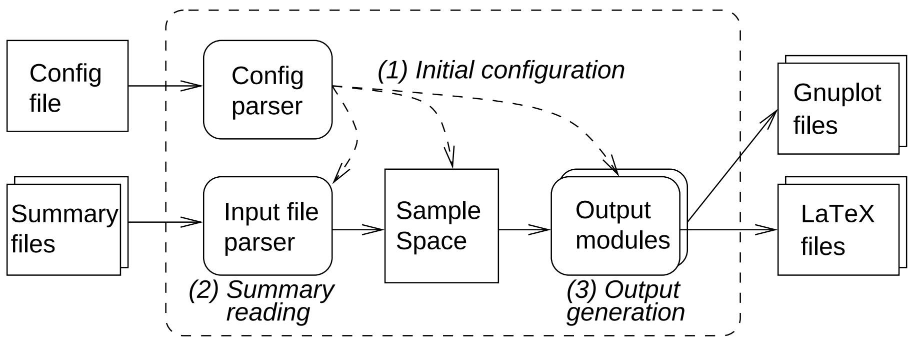

Figure 4.9: *Grace* consists of the components in the dashed frame, and operation is divided in three phases: configuration, summary reading and output generation.

a dump of the sample space may be written to disk, to avoid re-parsing the summary files for later graph generation.

Output generation: The output generation phase activates the present output modules, which extracts the required data from the sample space and writes their output, e.g., Gnuplot graphs and LaTeX summary files to provide an overview of the graphs.

4.4 Summary

The structure, phases of operation and data storage for the simulation framework has been defined. The separate tasks are solved by one or more applications, and allows individual applications to be replaced without influencing the the remaining framework functionality.

An existing scenario generator (*wsg*) and network simulator (ns2) is applied in two of the five phases. Development of applications for the remaining phases have been necessary to arrive at a complete framework. The structure and operation of the applications developed to solve the remaining tasks has been described.

Figure 4.10 illustrates the interaction between applications in the framework, during the process that takes scenario parameters as input and produces result graphs as output.

The established simulation framework is applied to conduct the simulations necessary for the protocol evaluation presented in the following chapter, and chapter 6 presents an evaluation of the framework as a way of automating the simulation process.

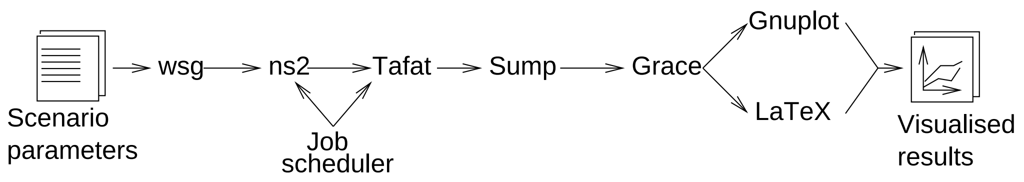

Figure 4.10: Overview of the interaction between simulation framework applications.

Chapter 5

Protocol Evaluation

In this chapter the performance of the four broadcast protocols and the suggested extensions are evaluated through simulations. Section 5.1 states the objectives for this chapter. Section 5.2 describes the set of metrics used in the evaluations, and how they are computed. Section 5.3 lists the tests that are conducted, and describes the general setup for the simulations. Sections 5.4 through 5.8 describe each of the tests in detail, and present the results of the simulations. Section 5.9 evaluates and reflects upon the results, and finally section 5.10 summarises the essential aspects of the chapter.

5.1 Introduction

The purpose of the evaluation is to identify effective techniques for broadcasting packets in a MANET. This is achieved by simulating four broadcast protocols and two protocol extensions in a large amount of randomly generated scenarios.

The protocols and the extensions are those described in chapter 3. For convenience, the protocol names are abbreviated as follows: Classic Flooding (CF), MPR Flooding (MPRF), Dominating Set Flooding (DSF), and Reverse Path Flooding (RPF).

The term *test* is used to refer to a group of simulations and results regarding a specific overall goal. The results of five tests are documented in this chapter:

Test 1: "Four Flooding Protocols" evaluates the four basic protocols

Test 2: "Classic Flooding With OLSR" examines the effect of the presence of OLSR control traffic on the Classic Flooding protocol.

Test 3: "Full-history Duplicate Elimination" identifies deficiencies in the duplicate elimination scheme.

Test 4: "Data Packet Jitter" evaluates jitter as a scheme for avoiding collisions.

Test 5: "Multipacket Flooding" evaluates multipacket flooding as a scheme for increasing reliability.

From the results of test 1 the protocol that performs "best" is identified, and this protocol will be used when conducting tests 3, 4 and 5. CF will also be used as a reference for evaluating the results.

The reason why only one protocol (in addition to CF) is selected is primarily to reduce the amount of simulations. For comparison purposes, the same protocol is used in all tests.

5.2 Metrics

This section specifies the selection of overall metrics used in the protocols evaluations.

The specified metrics are: effectiveness, delivery rate, end-to-end delay, path length, and bandwidth consumption. After describing these metrics, some general observations on *average graphs* are noted.

All simulations in this work use the same packet size. Hence, measuring bytes or packets is interchangeable, as they are proportional. This fact is used in some of the metrics.

5.2.1 Effectiveness

Effectiveness has been selected as the primary metric. For the evaluation of the effectiveness of the protocols simulated in this work, effectiveness has been defined to consist of two sub-measures: *reliability* and *efficiency*. The following sections describe these two measures in turn, as well as how they are computed. Then, effectiveness, as a function of reliability and efficiency, is defined.

Reliability

In abstract terms reliability means:

"How much traffic is successfully delivered?"

To measure reliability, the number of uniquely delivered bytes in a simulation is counted. Each sent byte is "delivered" at most once at each node, and duplicates are not counted (see the definition of "delivery" in Appendix A). To calculate the average percentage of nodes on which each byte was successfully delivered, this number is divided with the maximum possible number of delivered bytes, which is equal to the number of nodes in the network (n) times the number of bytes originated. This yields the following formula:

$$\mathbf{r} \equiv \frac{delivered}{n \cdot originated}$$

Both terms are expressed in bytes. n is the number of nodes in the network. Notice that the formula is undefined when the number of originated bytes is zero.

Note that this way reliability is not concerned with overhead of any kind, but simply expresses that if a node sends a byte, then how many nodes can in average be expected to receive the byte.

The reliability metric expresses how many percent of the nodes receive each byte (or packet) that is sent.

Efficiency

In abstract terms efficiency means:

"How many deliveries are achieved per 'bandwidth consumption'?"

The desired measure should issue a behaviour where more delivered bytes per "bandwidth consumption" yields the better result. The tricky part is to define a usable unit of bandwidth consumption.

Regarding bandwidth utilisation as being equivalent to "blocking" a node from the network (i.e., it is unable to transmit or receive a packet), bandwidth consumption can be measured as the sum of time where each node has been blocked. A node is blocked when the media (the "ether") is in use, which occurs exactly whenever the node itself is either transmitting or receiving a packet, or during a collision.

At this point, notice that when a node transmits a byte, it blocks itself and all receivers exactly for the time required to send the byte (assuming a 802.11 broadcast/multicast where no RTS/CTS/ACK occurs). Since all such time durations are equal, the bandwidth consumption may be measured in bytes instead of time. Hence, one transmitted byte or one received byte is equal to one unit of bandwidth consumption. This means that a transmission with several receivers will, as desired, cause several simultaneous bandwidth consumption units to be counted, and a transmission blocking many nodes is measured as being more expensive than a transmission blocking few nodes. So far, this yields the following formula:

$$\mathbf{e} = \frac{delivered}{transmitted + received + collisions}$$

All four terms are expressed in bytes. Each collision is counted on all the nodes blocked by the transmission. Notice that the formula is undefined when the number of transmissions, receptions, and collisions are zero (i.e., the network is never accessed). Also notice that

The problem with this formula is that it exhibits an undesired ranking of scenarios. More precisely, it does not differentiate scenarios with the desired granularity. Ideally, the following should be satisfied:

Assume two scenarios, A and B. In A, a node transmits a packet resulting in two deliveries. Scenario B is similar to A except in B the transmission results in *three* extra deliveries. It is desired that a protocol yielding scenario B should achieve a higher efficiency rating than a protocol yielding scenario A.

This, however, is not the case with the efficiency formula just described, as illustrated in table 5.1.

	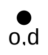	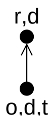	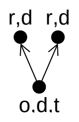	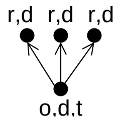	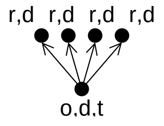
Originate (o)	1	1	1	1	1
Delivery (d)	1	2	3	4	5
Transmit (t)	0	1	1	1	1
Receive (r)	0	1	1	3	4
Collision (c)	0	0	0	0	0
$\frac{d}{t+r+c}$	n/a	1	1	1	1
$\frac{d-o}{t+r+c}$	n/a	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{3}{4}$	$\frac{4}{5}$

Table 5.1: Favorising scenarios with many local deliveries results in a wrong ranking of scenarios.

The source of the problem is that the formula really just checks whether there is a one-to-one mapping between bandwidth consumption and deliveries. This is commonly the case: On the originator there is one delivery, and one transmission. On each receiver there is one delivery, and one reception. Thus, each node involved has one unit of bandwidth consumption, and one delivery.

To overcome this problem, a disequilibrium can be enforced by not counting the delivery on the originator node. Thereby, to reach a high rating, it is necessary for the protocol to cause as many receptions as possible (non-duplicate ones, of course, to ensure a successive delivery).

Notice that subtracting these "originator deliveries" does not alter the result, because these deliveries *always* occur: There is no way (in ns2) to prevent a byte from being delivered at the node where it was originated. And since all the protocols are evaluated in identical scenarios, the same number of bytes is originated, and thus the same number of deliveries will be discarded.

This yields the following efficiency formula (see also table 5.1):

$$\mathbf{e} \equiv \frac{delivered - originated}{transmitted + received + collisions}$$

All five terms are expressed in bytes. Again, the formula is undefined when the denominator is zero (i.e., no network access occurs). Notice also that the range of the formula is $[0;1]$: It is not possible for the numerator to become negative, since an originated byte will always cause a delivered byte (at the originating node). And it is not possible for the numerator to become larger than the denominator, as shown below:

For the numerator to be larger than the denominator it is necessary to have a larger number of delivered bytes than originated bytes. But in order to achieve these deliveries, at least one reception is required per delivery. Thereby, the denominator will necessarily scale to at least the value of the numerator.

The efficiency metric expresses how many percent of the "bandwidth consumption units" result in a delivery

Weighing The Measures

The effectiveness measure, being composed of the reliability and the efficiency of a protocol, still leaves undisclosed precisely how effectiveness is calculated as a function of these two scores. Both the reliability and the efficiency measure have values in the range $[0;1]$, and it is desired that effectiveness likewise has a score in the range $[0;1]$. It is also desired that the effectiveness formula allows for context-dependent scaling of the two component factors. This is achieved with the following very general formula:

$$\mathbf{E} \equiv \frac{\mathbf{r} + x\mathbf{e}}{x + 1}$$

Where x is the scaling factor introducing the difference in the weighings of $\mathbf{r}$ and $\mathbf{e}$.

The value of x could in some instances be calculated, and in other instances be the result of subjective opinions on the importance of each of the two factors.

For the work in this report neither is the case. It is not possible to calculate a value for x, and it is not desired to introduce subjective opinions.

The selected solution is to pick a value for x that yields an effectiveness formula that "scales the two factors against each other", that is, multiplies them. Such a formula can be achieved from the general effectiveness formula by using a proper selection of x:

$$x = \frac{\mathbf{r} - \mathbf{re}}{\mathbf{re} - \mathbf{e}} \qquad \text{yields} \qquad \mathbf{E} = \mathbf{r} \cdot \mathbf{e}$$

This formula has the desired properties, and the scaling factor is removed.

5.2.2 Delivery Rate

The delivery rate expresses in average how many deliveries are achieved per flooded packet, i.e., *when a packet is sent, how many nodes will receive it?*

The delivery rate will thus always be a value in the range $[0; n]$, where n is the number of nodes in the network. This means that the delivery rate measure is directly proportional (by a factor of n) to the reliability measure of the effectiveness metric.

Since our simulations involve exactly 100 nodes, the delivery rate will always be a value in the range $[0; 100]$, and is thus equal to the reliability measure times 100. Thus, both measures are identical, only reliability is given as a decimal (e.g., 0.35), whereas delivery rate is in percent (e.g., 35%).

5.2.3 Delay and Path Length

Two questions regarding deliveries arise after examining the delivery rate. Delivery rate expresses *how many* deliveries are achieved, but it is also useful to know *when* and *where* the deliveries occur.

To answer the *when*, the end-to-end delay in (seconds) is measured.

To answer the *where*, two measurements are performed regarding path lengths: The average number of hops traversed by a packet when it is delivered, and the average number of deliveries of each packet that occur n hops away from the source.

From these results it is possible to determine three things:

1. Which protocol provides the shortest delays?
2. How closely does a protocol approximate shortest paths?
3. How far away from the source does a protocol succeed in delivering packets?

5.2.4 Bandwidth Utilisation

The bandwidth utilisation is simply measured by counting how many bytes per second are transmitted in the entire network. The amount of received packets and collisions are also counted, but the term "bandwidth utilisation" refers to the number of transmissions.

In addition to the raw transmission count, it is also counted how many packets (again an average count) are forwarded in the entire network during a whole simulation.

Notice that for these two metrics to be comparable with the same metrics from other simulations, it is required that all the simulations involve the same number of nodes, and use the same simulation time.

5.2.5 Average Graphs

In some instances it makes sense to present graphs plotting the average values of a number of other graphs, in order to present results in a more compact manner. These graphs are referred to as *average graphs*.

In this chapter, the majority of the graphs have "number of CBR streams" as the x-axis, and show a tested property on the y-axis. Furthermore, this property is tested in a number of load conditions, where the byte rate of the streams is varied. This information requires a z-axis. It is selected not to plot these results in 3-D graphs, as graphs tend to become increasingly difficult to read when more dimensions are added.

Instead, the graph is broken up into multiple 2-D graphs; one for each byte rate. These graphs are more easy to read than 3-D graphs when examining the results from a particular byte rate. The price is that 2-D graphs do not lend themselves well to overview – it is necessary to look at a different graph for each byte rate.

In this case, one way of presenting such information is to calculate the average for all byte rates at each stream-count, and plot them in a 2-D graph. In such graphs the byte rate axis is said to have been *marginalised out* of the graph. Marginalising the byte rate out of a graph has several consequences:

1. When the average of the results from a set of loads is calculated, it is implicitly assumed that all the loads in the set are weighted equally (i.e., a weight of 1).

2. An average graphs can only be compared with other average graphs calculated based upon the same set of loads.

The reason for the second consequence is that even when two average graphs are based on two sets of loads with the same load average, e.g., $A =$

$\{1,2,5\}$ and $B = \{1,3,4\}$, the average of the results might differ. If, for example, the simulation results for the loads are those shown in table 5.2, the result average for load set A is 4.3, while the result average for load set B is 7.

Load	1	2	3	4	5
Result	2	5	8	8	6

Table 5.2: The simulation results for loads 1, 2, 3, 4, and 5.

Average graphs are used frequently in this chapter in situations that satisfy the two conditions stated by the listed consequences: all byte rates are weighted equally (this is always the case in this work), and comparisons must be based on the same set of byte rates.

A particular frequent use of average graphs is for the effectiveness metric (and reliability and efficiency). The reason for applying average graphs for the effectiveness metric is that the purpose of this metric is to provide one graph that gives a quick overview of the performance of a protocol (bearing in mind that not all factors are included in the effectiveness formula).

5.3 Simulation Setup

The simulations conducted consist of a number of nodes moving around freely in a field of a given size with no obstacles. A number of these nodes are selected as data sources, emitting constant bit-rate (CBR) traffic. The selection of sources is dynamic in a simulation, but the number of simultaneous data sources remains fixed.

The movement pattern used is an instance of the random waypoint model, in which a node selects a random direction, moves a random distance in that direction at a randomly selected speed, waits for a randomly selected time interval, and selects a new direction.

Node speed is not varied in the simulations in this work to limit the number of simulations required. Testing the protocols under varying mobility is a direction of future work.

The scenarios are generated using the random scenario generator *wsg* described in section 4.3.1.

Table 5.3 lists the static parameters – the parameters that remain fixed for all tests.

All tests are conducted with 10 different numbers of simultaneously active CBR streams, which are: 5, 10, 15, 20, 25, 30, 40, 50, 75, and 100 streams.

Sections 5.4-5.8 describe these tests, and present the simulation results.

Parameter	Value
Field size	1400×1400 m
Number of nodes	100
Simulation time	300 seconds
Network capacity	2 Mb
Node movement	1–2 m/second
OLSR Hello message jitter	0–0.5 seconds
OLSR TC message jitter	0–1.5 seconds
CBR stream duration	10 seconds

Table 5.3: Static parameters for all simulations.

5.4 Test 1: Four Flooding Protocols

The purpose of this test is to provide a thorough survey of the four flooding protocols described in section 3. Based on this survey, the "best" protocol will be selected for further studies with the described protocol extensions.

In this test, each protocol is evaluated in a range of different traffic conditions. The static parameters of table 5.3 apply, and in addition the byte rates of the CBR streams is varied over the values 192, 384, 768, 2304, 3805, 5760, and 7680 bytes per second (B/s).

The combinations of byte rates and stream-counts result in a total of 70 scenarios. A test consists of 30 samples, meaning 30 simulations of each scenario per protocol, yielding 2100 simulations per protocol. With four protocols, a total of 8400 simulations are required for the completion of this test.

5.4.1 Expected Results

CF is expected to achieve a significantly higher delivery rate at low stream-counts than at high stream-counts, but the delivery rate should drop quickly as the stream-count increases. This is because CF is expected to saturate the network quicker than the other protocols since all nodes forward all packets immediately, causing a "broadcast storm" every time a packet is originated.

CF is still expected to be less reliable than the other protocols, particularly at high stream-counts.

CF is also expected to be the least efficient protocol, as it can be expected to cause many collisions.

The MPRF and DSF protocols are very similar, and hence they are expected to exhibit similar behaviours. The fact that DSF uses potentially older information than MPRF could prove a slight disadvantage.

It is expected that MPRF and DSF will achieve higher effectiveness than the other protocols, because they are expected to achieve better results regarding both reliability and efficiency.

The RPF protocol is expected to perform better than CF because a node does not forward the first reception of a flooded packet (like CF), but waits for the packet from the shortest path. This causes RPF to lessen the “broadcast storm” effect that is anticipated for CF. For the same reason, RPF is expected to show longer packet delays. RPF should, however, obtain shorter-or-equal paths when compared to CF.

Compared to DSF and MPRF, RPF is also expected to have shorter average paths, but only because RPF is expected not to distribute packets as far into the network as MPRF and DSF. There are two reasons for this: First, MPRF and DSF limits the number of nodes that forward broadcasted packets, whereas in RPF all nodes forward. Second, RPF does not provide path redundancy.

5.4.2 Results

The primary metric used for this test is the effectiveness. In the following sections the effectiveness (and the sub-metrics) are examined, followed by a more detailed look at the delivery rate, path length, and bandwidth consumption observed at selected loads.

Effectiveness

Figure 5.1 shows the average reliability. MPRF achieves the highest reliability at all stream-counts. The reliability of DSF is 7-9% lower than that of MPRF, and CF is 16-30% lower than MPRF. RPF achieves the lowest reliability, 42-62% lower than MPRF.

These results were not as expected. RPF is less reliable than expected, whereas CF is more reliable than expected. CF was expected to achieve the lowest reliability by a large margin at high stream-counts, but CF is only 10-24% lower than DSF at all stream-counts.

Figure 5.2 illustrates the efficiency of each protocol. Clearly, there is a big difference in the behaviours of the graphs. CF results in a nearly linear graph, whereas the other protocols show increasing efficiency with increasing number of streams. At 15 streams or less, CF is the most efficient protocol, resulting in a 11-90% increase compared to MPRF, which in this section of the graph has the second highest efficiency, and a 77-302% increase compared to RPF (the lowest).

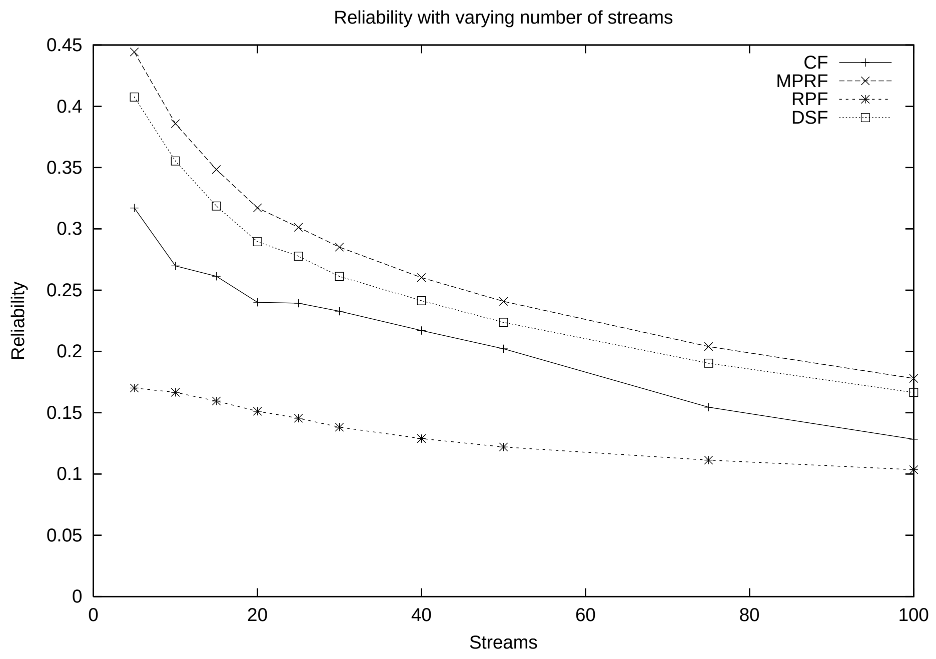

Figure 5.1: Average reliability.

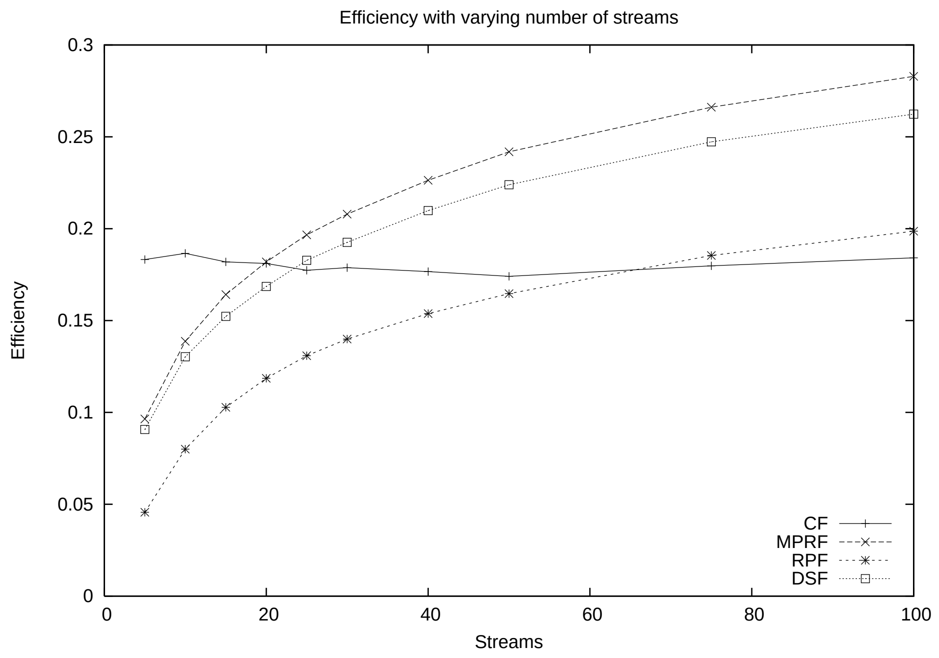

Figure 5.2: Average efficiency.

As in the reliability graph, MPRF and DSF achieve similar results, and MPRF achieves a 6-8% higher efficiency than DSF at all stream-counts.

RPF has the lowest efficiency at all stream-counts lower than 75, where it surpasses CF. The RPF graphs is shaped similar to those of MPRF and DSF, but RPF remains 24-50% lower than DSF at all stream-counts.

It is unexpected that CF achieves the best efficiency at low stream-counts, as CF was believed to cause more traffic than any other protocol – also at low stream-counts – and it was shown to provide fewer deliveries than MPRF and DSF at all stream-counts.

Figure 5.3 shows the effectiveness of the protocols, calculated from the reliability and efficiency results.

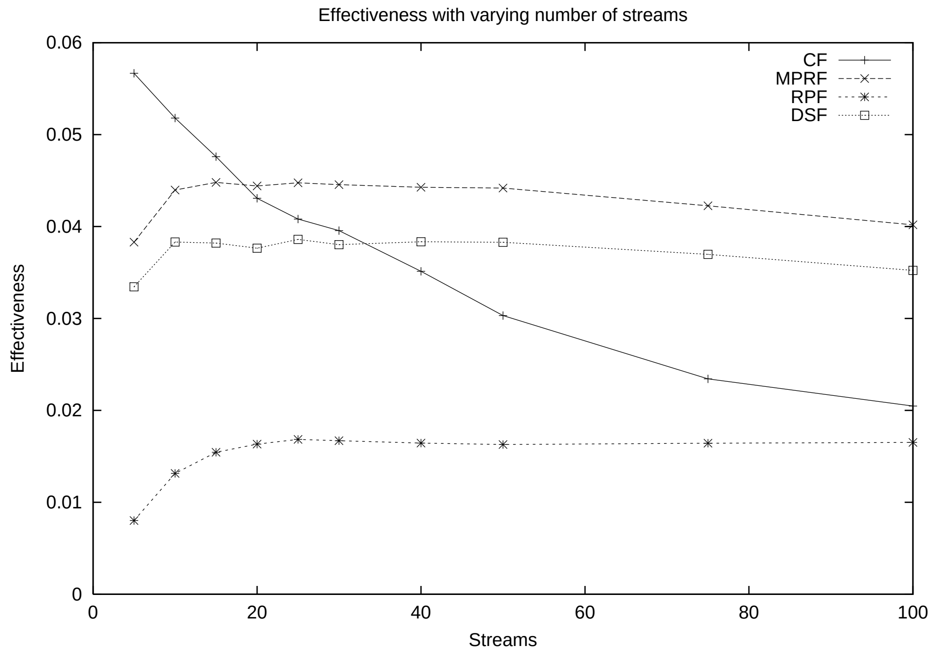

Figure 5.3: Average effectiveness.

The behaviour of the CF graph stands out in effectiveness as well as in efficiency. CF shows a continual decrease, while the other protocols show an increase from 5 to 10 streams (RPF until 15 streams), and remains stable until 50 streams, where MPRF and DSF begin to decrease.

At 15 streams or less, CF is the most effective protocol, with an advantage of up to 48% over MPRF, which has the second highest value at these stream-counts. Between 30 and 40 streams, CF drops below DSF, and at 100 streams has dropped to 58% of DSF.

An interesting observation is that CF's advantage at low stream-counts results from efficiency, and not reliability.

RPF has the lowest effectiveness at all stream-counts. At 5 streams RPF is 76% lower than DSF, and at 100 streams, 53% lower.

The fact that MPRF and DSF show similar results for both reliability and efficiency is reflected in effectiveness. As a result of having the highest value in both reliability and efficiency, MPRF also has the highest effectiveness value. At all stream-counts, MPRF is 14-18% higher than DSF.

Delivery Rate and Path Length

Figure 5.4 shows four things regarding the lowest load simulated: First, the maximum delivery rate obtained by any protocol is 51% (by both MPRF and CF). Second, RPF remains 53-64% lower than DSF at all stream-counts. Third, MPRF has a 5-12% higher delivery rate than DSF. And fourth, CF displays a peculiar peak to which no explanation has been found.

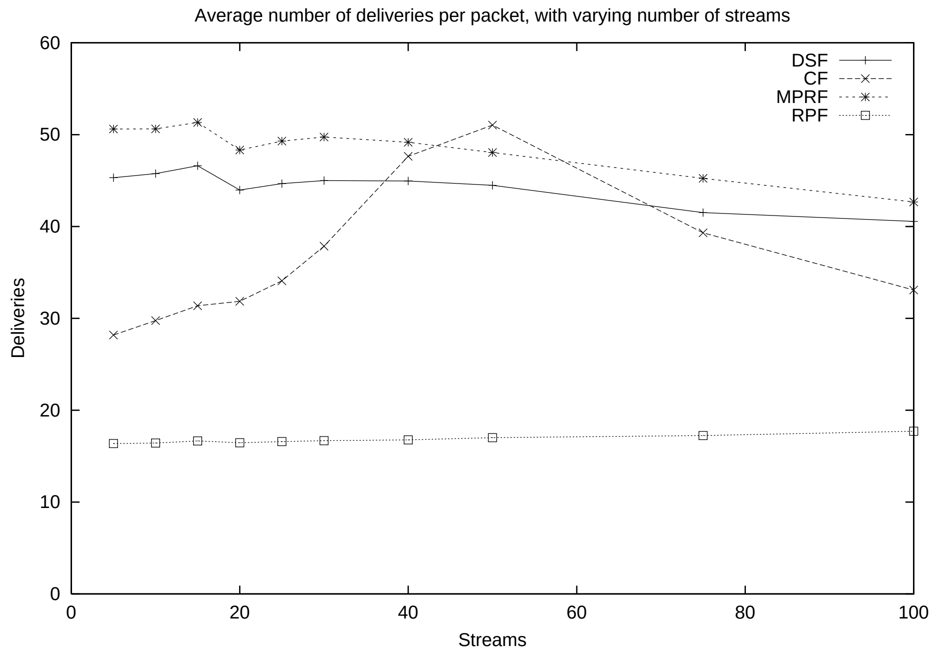

Figure 5.4: Delivery rate at 192 B/s.

At 5 streams, CF is 38% lower than DSF, increasing to 15% higher than DSF at 50 streams (and 6% higher than MPRF), after which CF drops to 18% lower than DSF.

Figure 5.5 shows the delivery rates at 768 B/s. Again, the maximum delivery rate achieved is 51% by MPRF and CF. Also, a peak is appears again in the CF graph. At 5 streams CF is 30% lower than DSF, increasing to 18% higher than DSF at 15 streams, and then CF drops to 26% lower than DSF at 100 streams.

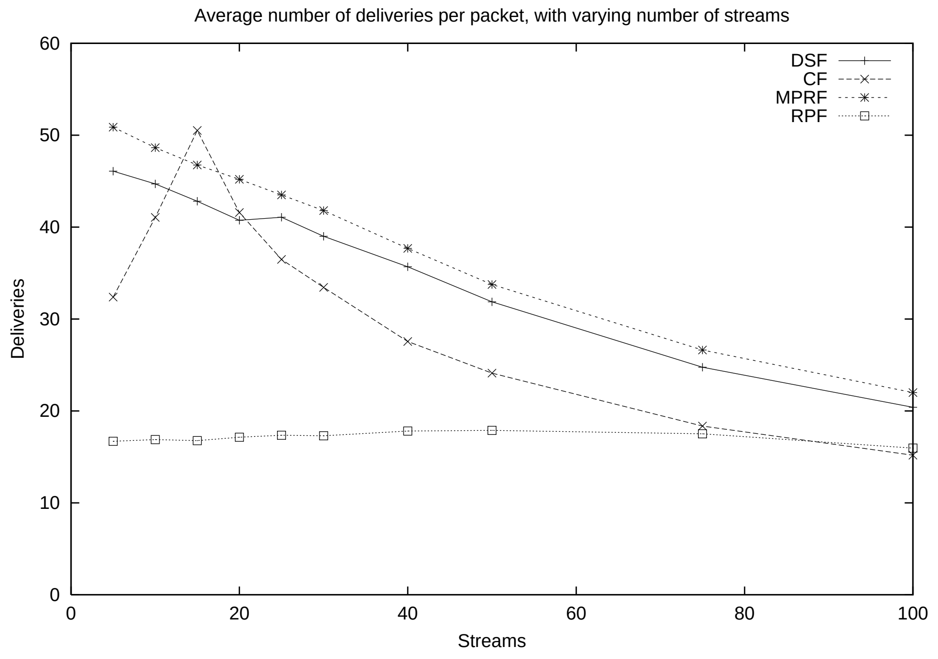

Figure 5.5: Delivery rate at 768 B/s.

RPF has the lowest delivery rate at all stream-counts except 100, where it surpasses CF by 5%. At this point, RPF remains 22% lower than DSF. At all stream-counts, DSF is 22-64% lower than DSF.

MPRF remains 6-11% higher than DSF at all stream-counts.

Figure 5.6 shows the delivery rate at 7680 B/s, which is the highest load simulated. At this load condition, MPRF and DSF remain at a higher delivery rate than the two other protocols, MPRF achieving a 6-10% higher rate than DSF.

At 5 streams, CF is 22% higher than RPF, and at 10 streams or more, RPFlood is 4-19% higher than CF.

At 100 streams the maximum delivery rate (achieved by MPRF) is 4%, and the lowest (CF) is 3%. As there are 100 nodes in the network, this can be translated directly into 3-4 nodes receiving the packets, which is fewer than the average 1-hop-neighbourhood size. Figure 5.7 confirms this observation by showing that at 7680 B/s and with 100 streams, the average path length varies from 1.1 hop (RPF) to 1.5 hops (CF).

Figure 5.7 also shows that at 7680 B/s CF has longer paths than any

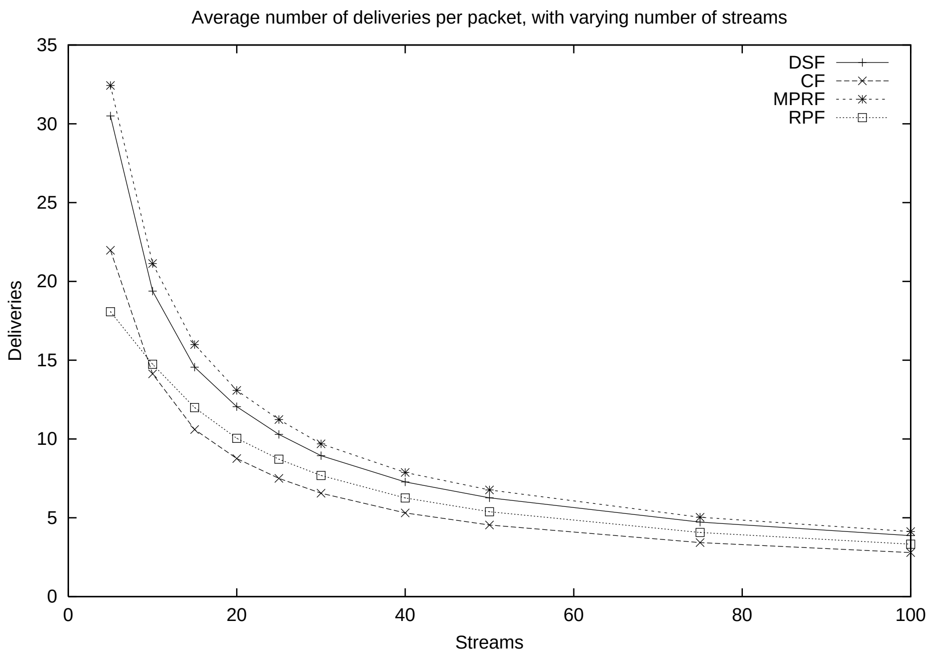

Figure 5.6: Delivery rate at 7680 B/s.

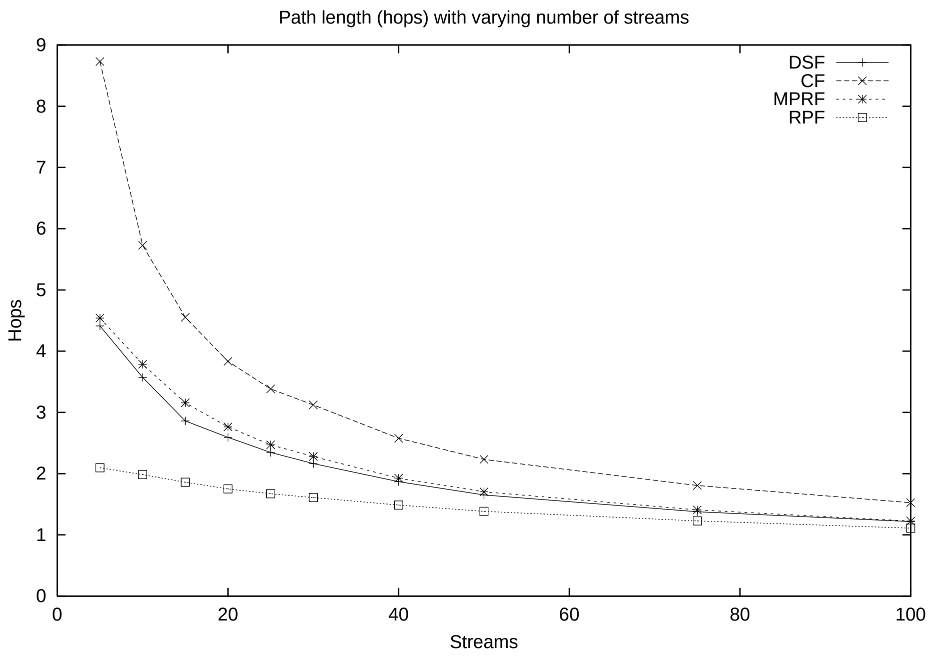

Figure 5.7: Average path length at 7680 B/s.

other protocol (38-317% longer than RPF, and 24-92% longer than MPRF), meaning that CF succeeds in bringing packets further into the network.

Together with figure 5.7, figure 5.8 illustrates that RPF has the shortest path lengths in all load/stream-count conditions.

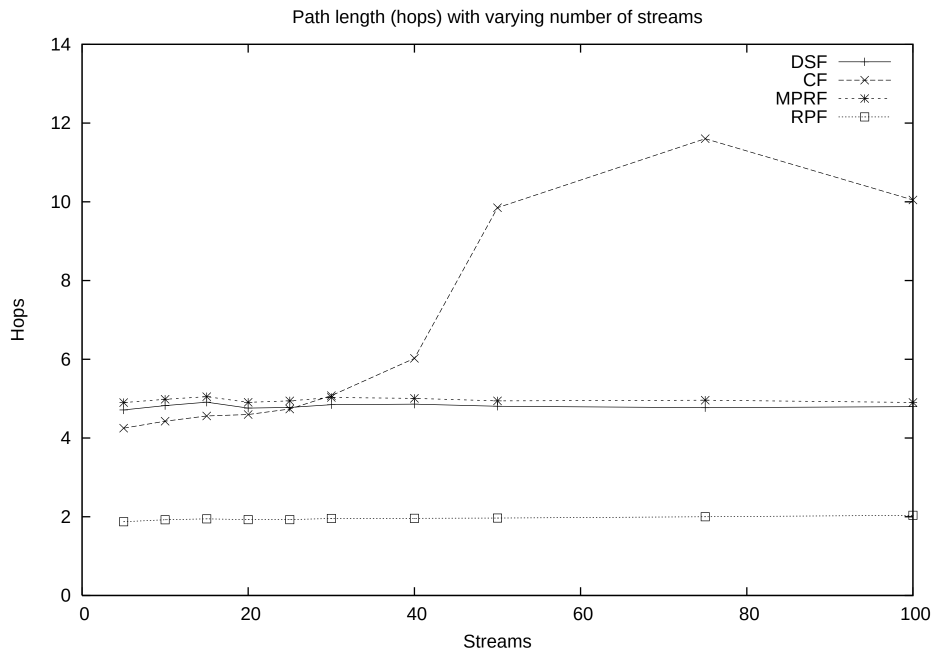

Figure 5.8: Average path length at 192 B/s.

This indicates either that RPF performs well in finding short paths, or that nodes far away from the source do not receive packets at all, which explains the low delivery rate. Figure 5.9 shows the average fraction of the total number of delivered packets that are delivered at each distance (in hops) from the source by each protocol. RPF stands out from the other protocols by obtaining 41% of the deliveries at the immediate neighbours to the source. DSF, MPRF, and CF obtains only 25%, 24%, and 22% respectively.

Figure 5.9 also indicates that some packets have traversed between 45 and 50 hops. From the result files it is found that the maximum path length traversed by any packet in this test is 133 hops. Clearly, this should not be possible in a network with only 100 nodes, and where duplicate elimination ensures that any packet is forwarded at most once by each node. It will be examined in test 2 whether this result is due duplicate packets that escape the duplicate elimination.

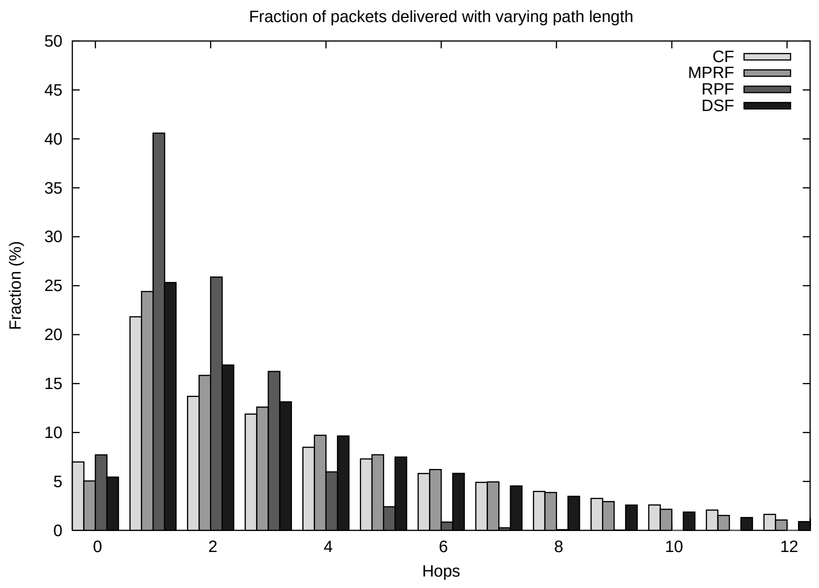

Figure 5.9: Average fraction of total deliveries registered at specific distances (in hops) from the source.

Bandwidth Utilisation

Figure 5.10 is an average graph showing the bandwidth consumption for each protocol. MPRF, DSF, and RPF achieve similar results, the distance from CF to the nearest protocol varies from 117% with 5 streams to 30% with 100 streams.

5.4.3 Conclusions

MPRF has been shown the best average reliability result and efficiency result (surpassed by CF in efficiency at 5-10 streams). As a result, MPRF also has the best effectiveness (again, except at 5-10 streams). Second to MPRF is DSF.

CF is found not to perform as poorly as expected compared to the other protocols, especially, RPF is generally outperformed by CF, except in bandwidth overhead.

Finally, the test has shown that none of the tested protocols achieve more than a 51% delivery rate, and thus, although MPRF and DSF perform better than CF, further improvements may be possible.

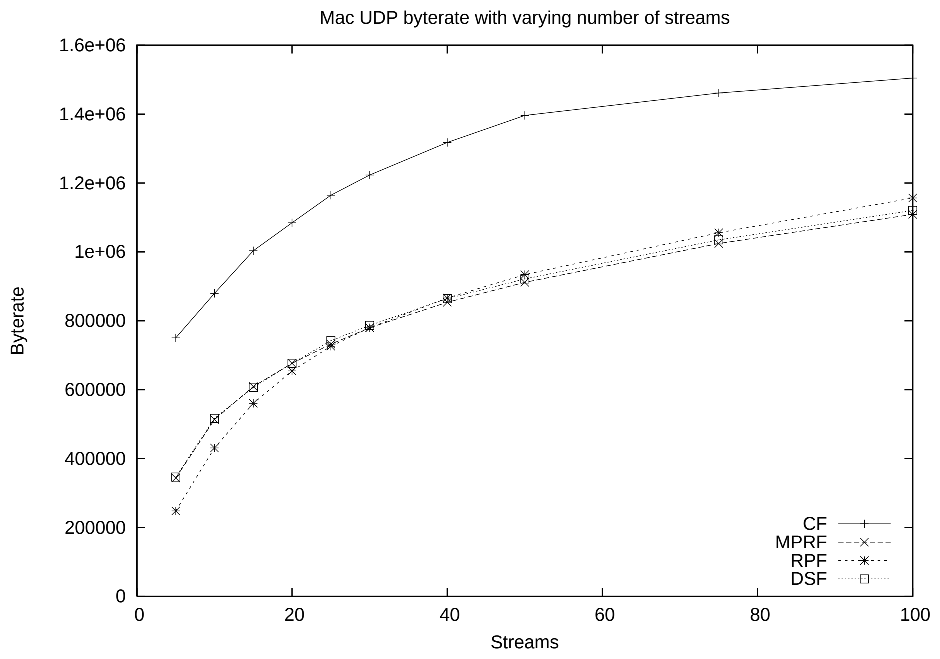

Figure 5.10: Average UDP bandwidth consumption.

5.4.4 Summary

In this test, the four selected broadcast protocols have been simulated, and MPRF has been found to overall perform better than the other protocols. In particular when compared with RPF and CF. The DSF protocol showed results that approximate MPRF well enough to select DSF as the protocol that will be used in the other tests. The background for this selection is the implementational difficulties involved with MPRF (requires changes to the IP-stack). This cost has been found to outweigh the increased performance displayed by MPRF.

5.5 Test 2: Classic Flooding with OLSR

In section 5.4 DSF, MPRF, and RPF were utilising an underlying implementation of the OLSR unicast routing protocol, suffering from an overhead which was not present in the CF simulations. For completeness a test is devised to determine the effects of OLSR traffic on CF, to be able to compare the protocols on an “equal basis”, disregarding the presence of OLSR.

In this test the same scenarios are used as in Test 1, yielding 2100 simulations.

5.5.1 Expected Results

In low-load scenarios it can be expected that the extra traffic has little impact on the transmission of data packets. As the network load increases, so will the number of collisions caused by the presence of OLSR packets, resulting in lower reliability because packets are lost when colliding (there is no collision detection), and lower efficiency because both the increase in collisions itself, and the decrease in reliability, causes a decrease in efficiency.

5.5.2 Results

The starting point is again the effectiveness metric. First, the reliability metric and the efficiency metric are examined independently, followed by the effectiveness metric. Last, the delivery rate and bandwidth utilisation are examined.

Reliability

Figure 5.11 illustrates a 9-22% higher average reliability for CF+OLSR when 40 simultaneous streams or less are present in the network. With 50 streams or more the gain is reduced to 1.4-3.0%. Reliability is calculated as the average delivery rate. It is examined later in this section why adding extra traffic improves the delivery rate of CF.

Efficiency

Figure 5.12 shows that in average, CF+OLSR achieves a much lower efficiency with 40 streams or less. At 5 streams CF is more than 2.4 times as efficient as CF+OLSR. At 40 it is reduced to a 13% increase. Notice that the efficiency graph of CF+OLSR behaves very differently from that of CF, which remains comparatively stable at approximately 0.18.

The efficiency is calculated from several factors, which can be divided into two groups: those concerning deliveries (the enumerator of the efficiency formula), and those concerning bandwidth consumption (the denominator). A decrease in efficiency must be reflected by either a smaller enumerator, a larger denominator, or both.

A decreased enumerator ($d - o$) for CF+OLSR can only result from CF+OLSR obtaining a large fraction of it's deliveries at the originator (a large o), since the reliability results (figure 5.11) show that CF+OLSR has a larger *total* amount of deliveries (d) than CF. However, figure 5.13 illustrates that this is not the case. CF and CF+OLSR deliver an almost identical fraction of the deliveries at the originator, and in fact CF+OLSR obtains 2.4%

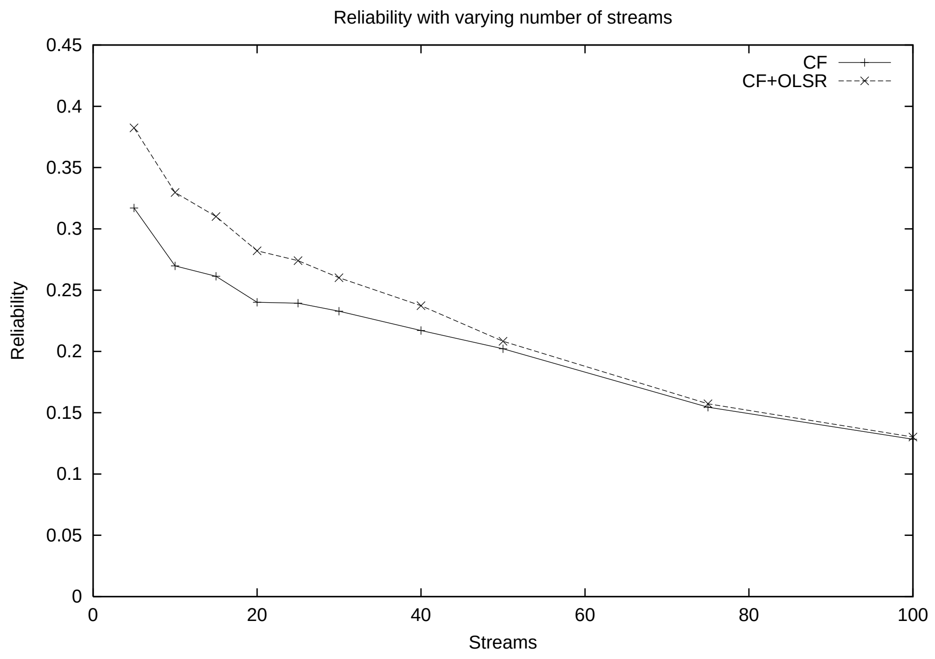

Figure 5.11: Average reliability.

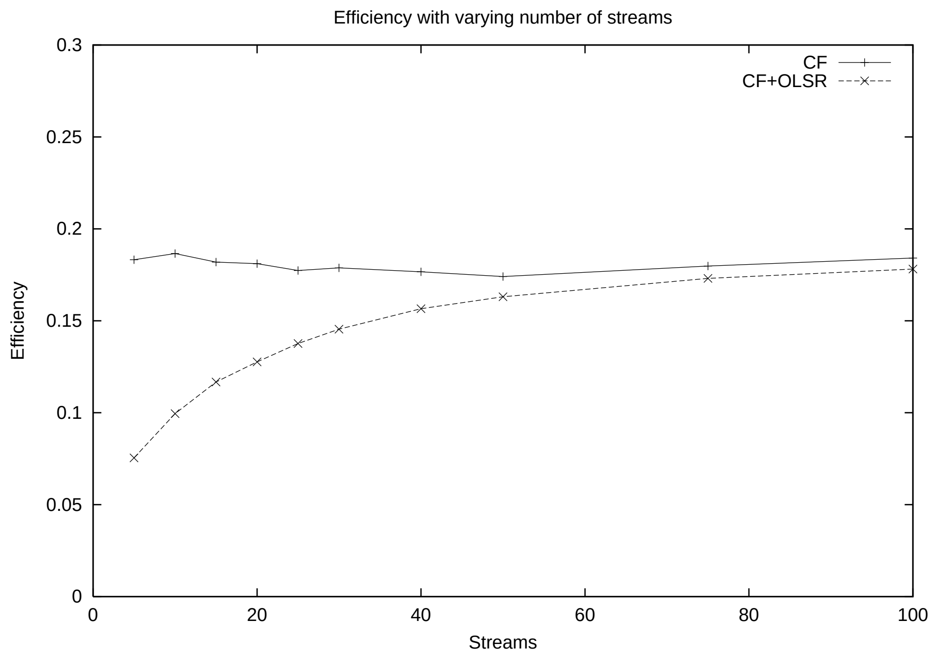

Figure 5.12: Average efficiency.

less originator deliveries than CF. Hence, the decreased efficiency must be the result of a larger denominator.

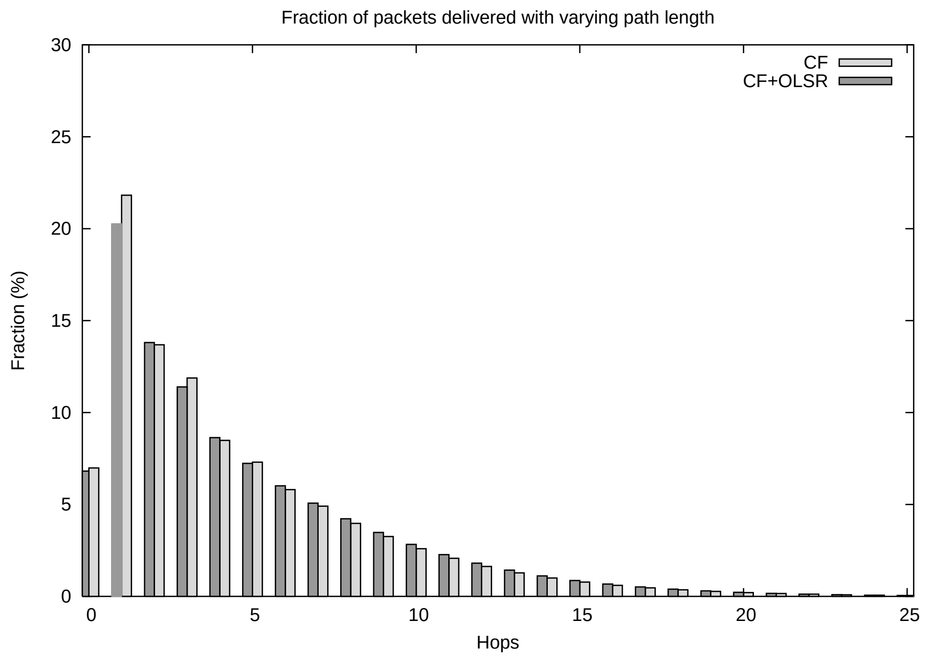

Figure 5.13: Average fraction of total deliveries registered at specific distances (in hops) from the source.

A larger denominator means a combination of larger amounts of transmissions, receptions, and collisions. Figure 5.14 illustrates the average bandwidth utilisation measured in terms of transmissions. The figure shows that difference between CF and CF+OLSR is 2.8% or less at all stream-counts.

Figures 5.15 and 5.16 show that at any stream-count, CF+OLSR has more packet receptions and more collisions than CF. With 40 streams or less, CF+OLSR has 16-86% more receptions than CF, and 11-20% more collisions.

These results cause the "expense" of the extra deliveries CF+OLSR achieves to outweigh the number of extra deliveries, thereby decreasing the efficiency of CF+OLSR.

Effectiveness

The reliability and efficiency results yield the effectiveness illustrated in figure 5.17. At 5 streams CF is 2.2 times more effective than CF+OLSR, dropping to an advantage of 20% at 40 streams, and less than 5% at 75 and 100 streams.

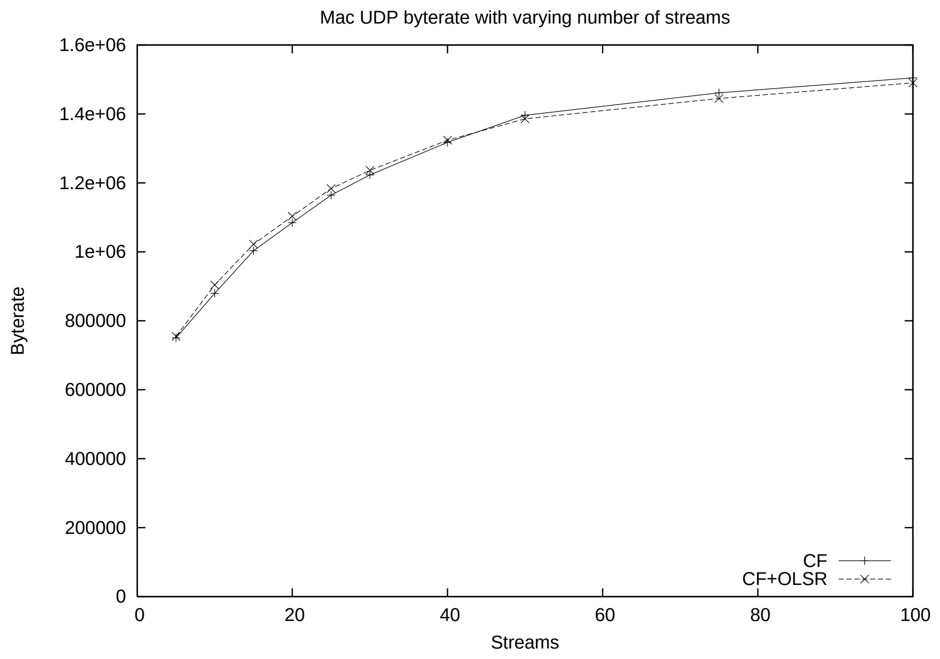

Figure 5.14: Average bandwidth utilisation (number of transmissions).

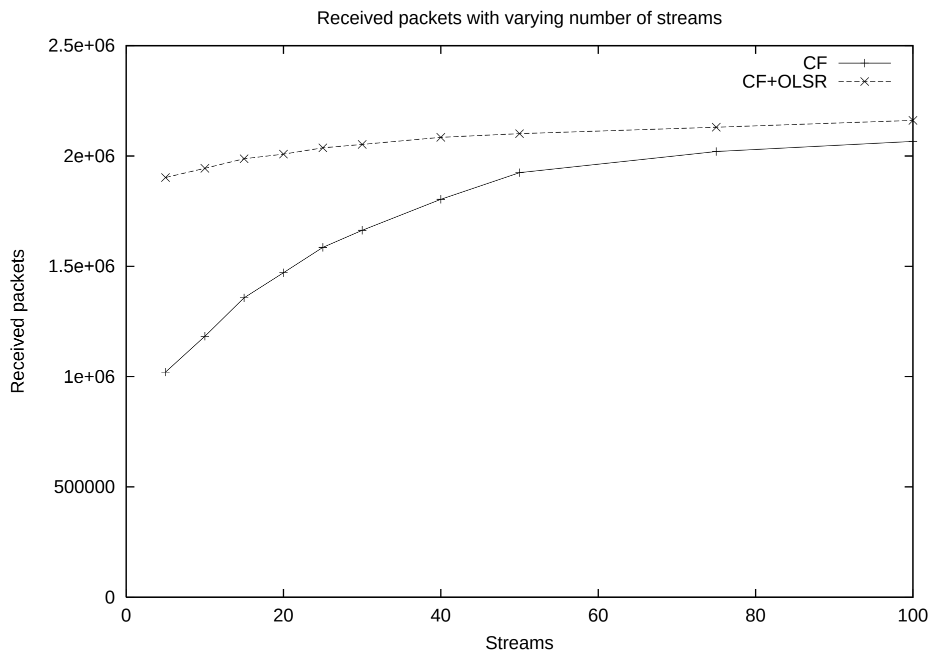

Figure 5.15: Average number of packet receptions.

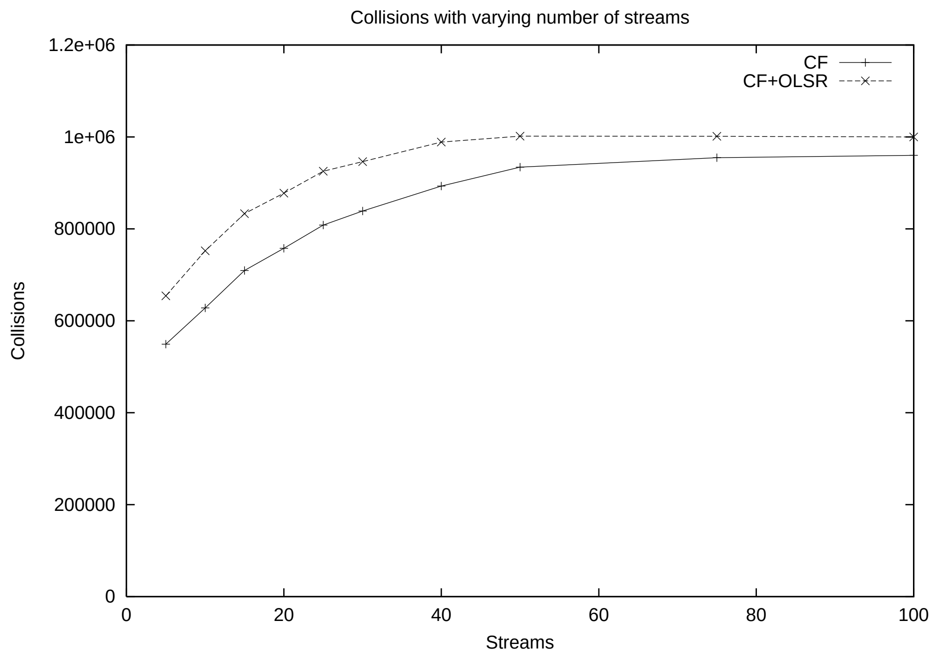

Figure 5.16: Average number of collisions.

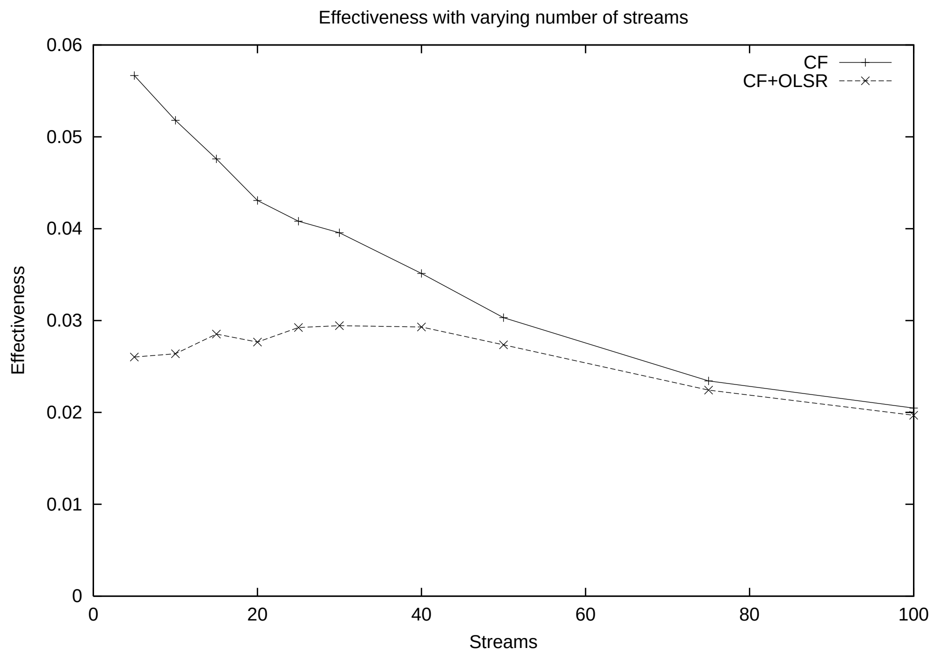

Figure 5.17: Average effectiveness.

The reason for this behaviour is that while the delivery rate of CF+OLSR is higher than that of CF, the expense per delivery is also higher, and the increase in reliability is not large enough to outweigh the decrease in efficiency.

Delivery Rate

Figures 5.18, 5.19, and 5.20 show the delivery rates with stream byte rates of 192, 768, and 7680 B/s respectively.

With 192 B/s and 25 simultaneous streams or less, CF+OLSR shows a significant advantage of 42-49%. From 30-40 streams a difference of 26-38% is found, and at 50 streams and more, 2-6% more deliveries are found.

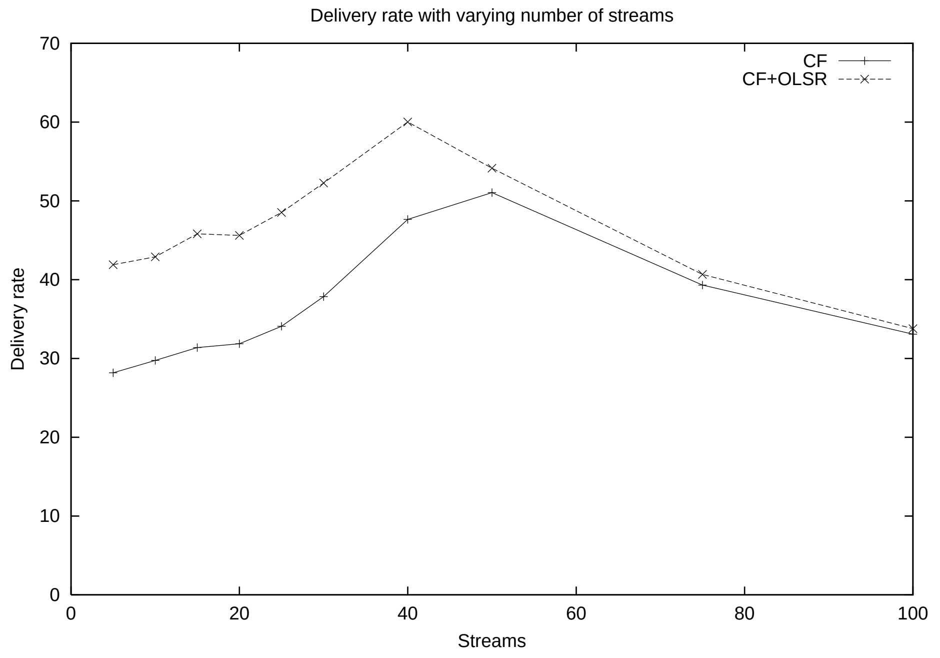

Figure 5.18: Delivery rate at 192 B/s.

With 768 B/s per stream, only at 15 streams or less is a significant difference observed, where CF+OLSR achieves 9-42% more deliveries. At 20 streams or more, the difference drops to 0.6-3.0%

With 7680 B/s per stream CF+OLSR also achieves more deliveries than CF, but the difference is less than 2% at all stream-counts.

Bandwidth Consumption

The bandwidth consumption at byte rates of 192, 768, and 7680 B/s are shown in figures 5.21 through 5.23.

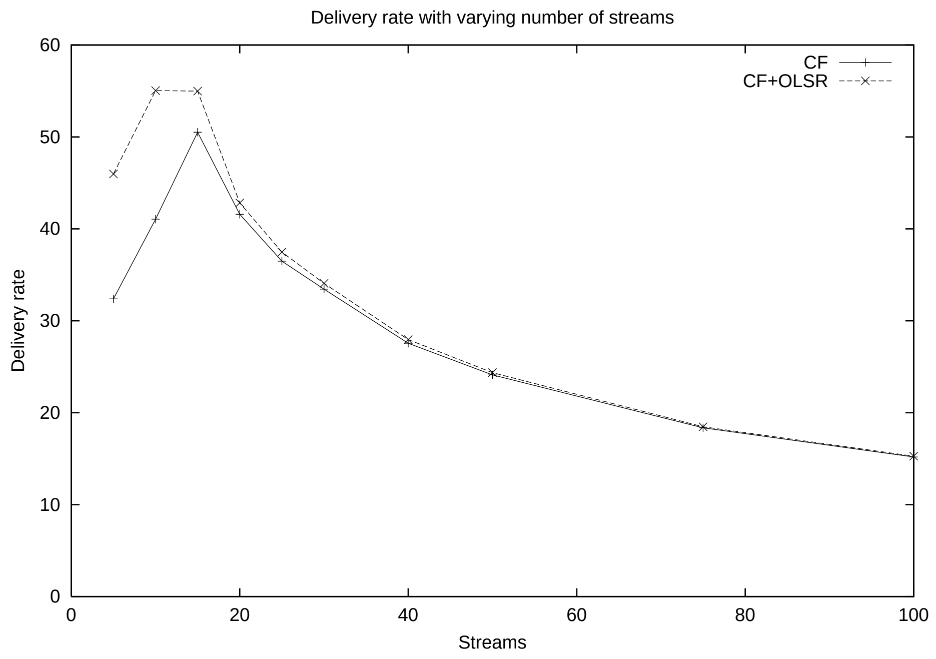

Figure 5.19: Delivery rate at 768 B/s.

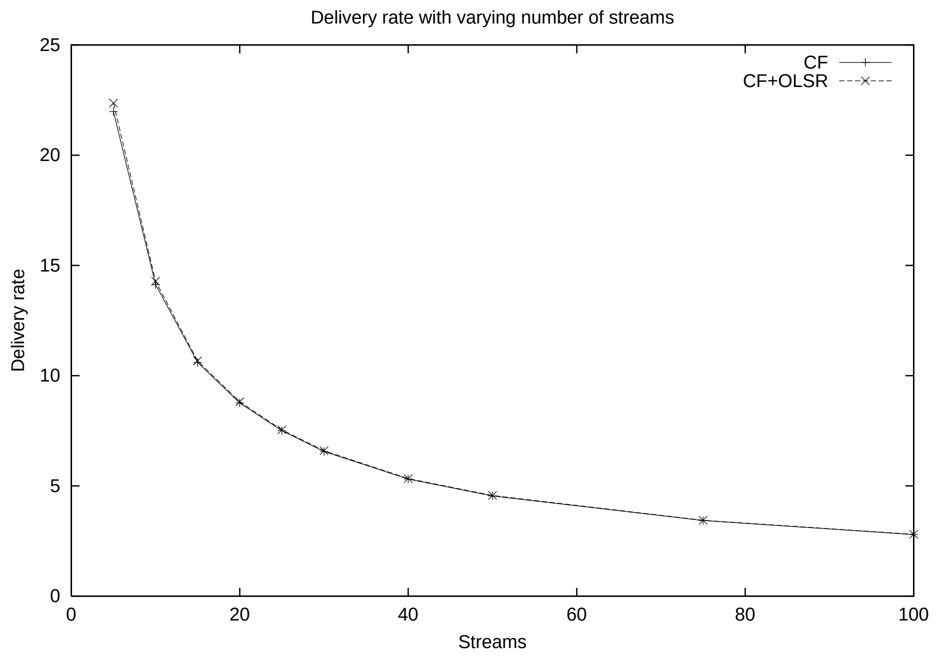

Figure 5.20: Delivery rate at 7680 B/s.

Figure 5.21 shows that at a byte rate of 192 B/s, and with 50 streams or less, CF+OLSR transmits 24-45% more bytes than CF, but with 75 streams or more, CF transmits 2-3% more bytes than CF+OLSR. Notice that the graphs cross just as they begin to “flatten”. The flattening of the graphs indicates an overload situation.

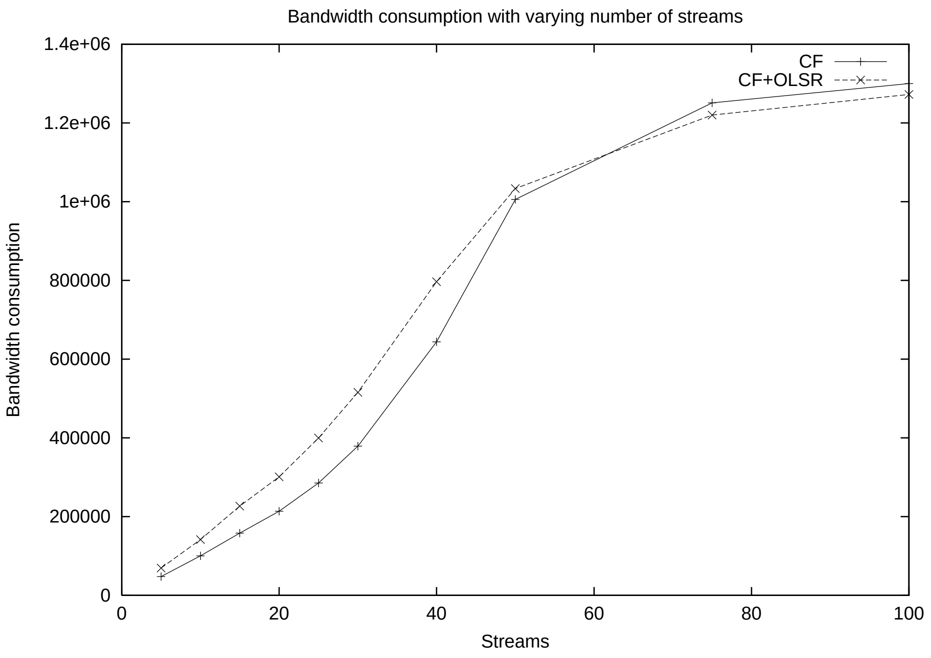

Figure 5.21: bandwidth utilisation at 192 B/s.

Similar behaviour is found at 768 B/s (figure 5.22). The graphs cross between 15 and 20 streams, and the same proportional differences as with 192 B/s are observed: before the crossing, CF+OLSR transmits 3-39% more bytes than CF, and after the crossing, CF transmits up to 2% more bytes than CF+OLSR. Again, the graphs cross just as they begin to flatten.

At 7680 B/s (figure 5.23) CF makes more transmissions than CF+OLSR at all stream-counts, but only 0.3-1.9%. The graphs indicate an overload situation.

To sum up, adding OLSR traffic does not reduce the number of data packets transmitted significantly. The reason is that only in low load conditions does the OLSR traffic represent a noticeable fraction of the total traffic, but at low loads there is bandwidth enough for both kinds of traffic. At high loads the relatively small proportion of traffic that OLSR generates is not enough to disturb the data traffic noticeably.

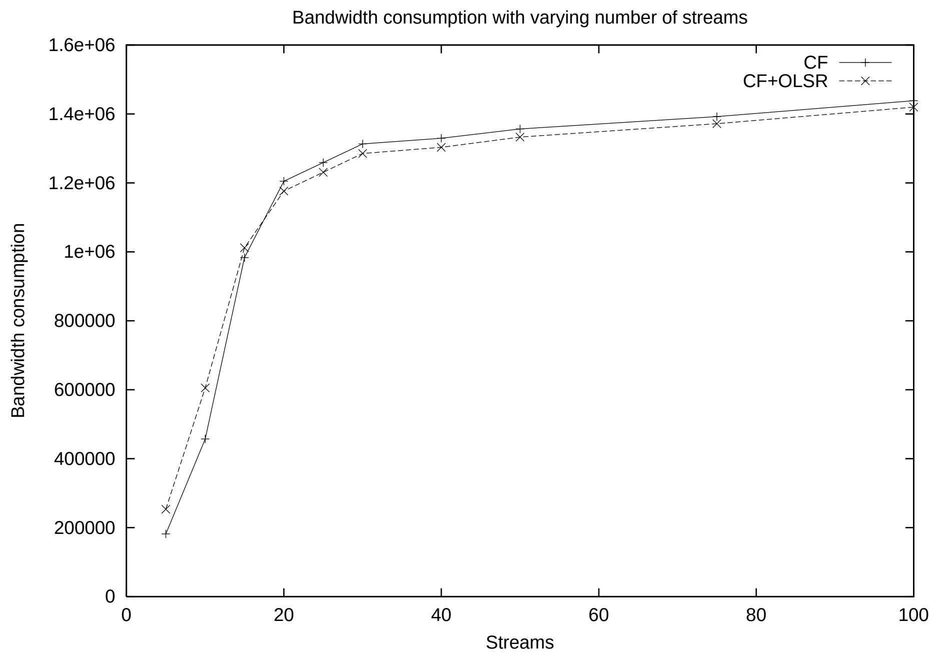

Figure 5.22: Bandwidth utilisation at 768 B/s.

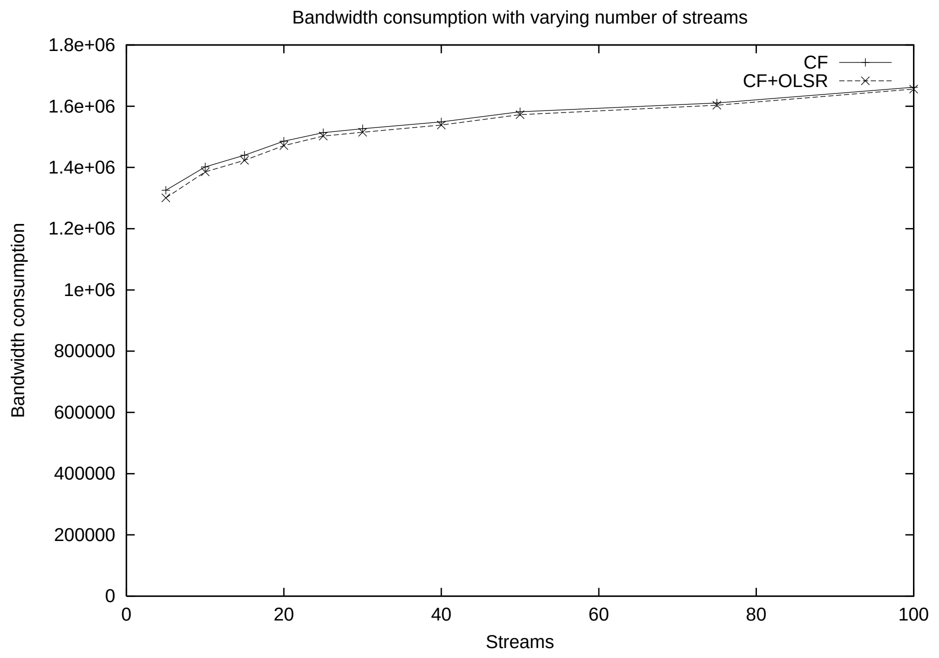

Figure 5.23: Bandwidth utilisation at 7680 B/s.

CF+OLSR: Same bandwidth utilisation, but more deliveries

Comparing the average bandwidth graphs (5.14) with the average reliability graphs (5.11) raises an interesting question regarding CF+OLSR: why does CF+OLSR achieve more deliveries with the same amount of transmitted packets[1]?

The question was, in part, answered when examining the efficiency, by observing that CF+OLSR obtained the most *received* packets.

Another part of the answer is found in undetected duplicates. Figure 5.24 shows that 8-15% more undetected duplicate packets are received with CF than with CF+OLSR. This means that out of the total bandwidth consumption in CF and CF+OLSR respectively, a larger portion in CF are contributed by (possibly spinning) duplicates, that is, packets that take up bandwidth, but do not contribute deliveries.

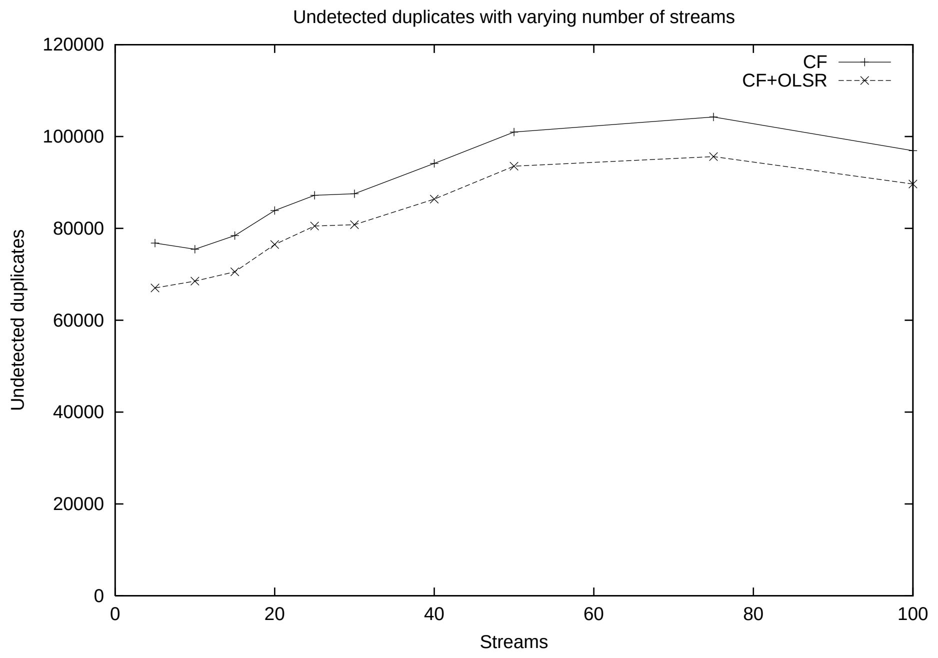

Figure 5.24: Average number of undetected duplicate packets.

This observation provides an answer for the question, but it also raises another question: why do fewer undetected duplicates appear when extra traffic is added? We have not been able to find an answer to this question.

[1]The graph shows transmitted *bytes*, but packets are equal-sized, so there is a direct proportionality between transmitted bytes and transmitted packets

5.5.3 Conclusions

Adding the control packet of overhead of OLSR does not reduce the bandwidth capacity in terms of transmissions. There is, however, an increase in *received* packets, which is also reflected in the delivery rate, making CF+OLSR more reliable than CF.

The OLSR traffic causes more collisions to occur, which, together with the extra receptions, cause the efficiency of CF+OLSR to decrease, compared to CF.

The efficiency decrease of CF+OLSR is larger than the reliability increase, resulting in a decreased effectiveness.

Fewer undetected duplicate packets are observed with CF+OLSR than with CF, allowing CF+OLSR to deliver more packets with the same bandwidth.

5.5.4 Summary

In this test, OLSR carries out its transmission of control traffic, and configuration of unicast routes, consuming some of the shared bandwidth. It is observed that the presence of OLSR traffic affect the performance of CF in several ways.

The number of undetected duplicate packets decreases, meaning that a larger fraction of the transmitted packets are unique, causing the measured number of received packets to increase. The increase in received unique packets contribute with additional deliveries, thus increasing the delivery rate and hence the reliability.

Adding OLSR traffic causes causes collisions, which, along with the extra packet receptions, decreases the efficiency. The efficiency decreases more than the reliability increases, resulting in a lower effectiveness result when OLSR traffic is present.

It remains unexplained why adding OLSR's control traffic to the network decreases the occurrences of undetected duplicate packets.

5.6 Test 3: Full History Duplicate Elimination

Duplicate elimination ensures that once a packet has been received (and perhaps forwarded) by a node, that node will not process any subsequent receptions of the same packet. The duplicate elimination scheme devised in chapter 2 is time limited, however, meaning that if a packet is in transit longer than the packet identifier remains in the duplicate elimination history, the packet may be processed, and thus forwarded, several times by one node.

The consequence when this happens is “noise” in the simulation results: the results, although realistic, indicate drawbacks of the duplicate elimination scheme, and not the flooding protocol.

The result files from test 1 (section 5.4) contain information about packets that have traversed as many as 133 hops, which should not be possible as there are only 100 nodes in the network. This indicates that duplicate packets are present in the network.

To determine the frequency of this problem, a test is conducted where the duplicate elimination history stores all received packet identifiers for the full duration of the simulation.

This test is conducted on one byte rate (384 B/s) with CF and DSF, yielding 600 simulations.

5.6.1 Expected Results

It is expected that the maximum number of hops traversed by any packet decreases when enforcing full duplicate elimination. If this is the case, it will indicate that the packets traversing more than 100 hops in test 1 are duplicates.

5.6.2 Results

As expected, all duplicate receptions of packets are intercepted by duplicate elimination in these tests – the number of undetected duplicates is zero in all tested conditions.

The result files from this test contain information about packets that have traversed up to 39 hops, showing that a large decrease compared with the 133 hops with 15 seconds duplicate history timeout.

5.7 Test 4: Jitter on Data Packets

The 802.11 MAC specification states that no RTS/CTS is performed when transmitting broadcast and multicast packets. Consequently, if neighbour nodes attempt to broadcast simultaneously, a collision will occur. A network with broadcast/multicast traffic will be prone to many collisions.

Furthermore, when a node, N, is part of a unicast route, and N forwards the packet, the packet will usually be forward by at most one of N's neighbours.

If N forwards a broadcast packet, a subset of neighbours to N (as defined by the broadcast protocol) will forward the packet. If some of these neighbours are within transmission range, it is a possibility that some of them forward the packet at the same time, causing collisions.

One way to try to remedy this problem is to enforce *jitter* upon the transmissions. Jitter means that the transmission is delayed for some short, random time period. It has been shown in [CHCB01] that jitter has a positive effect on OLSR control traffic. It is desired wish to investigate the effects of enforcing jitter on flooding, and in particular, to investigate whether jitter reduces the number of collisions in the network.

Jitter works by delaying the transmission for some randomly selected delay between 0 and *Max_Jitter* seconds. To get a broad perspective, tests are conducted with several selections of *Max_Jitter*. The following *Max_Jitter* values (in seconds) are tested: 0.002, 0.01, 0.05, and 0.1-0.9 with increments of 0.2 seconds.

The jitter test is conducted with 8 different values of *max_jitter* on two protocols (CF and DSF), yielding 16 protocols. One byte rate is tested (768 B/s), amounting to 4800 simulations.

5.7.1 Expected Results

Adding jitter is expected to increase the end-to-end delay of broadcast operations, since a delay is manually enforced on every node on the path. Only if jitter causes shorter paths to be used can the delay be reduced.

In low-load scenarios, enforcing jitter is expected to prevent some of the collisions that occur due to the reasons just described. However, jitter will leave vacant gaps in the ether when none of the nodes in a neighbourhood are transmitting, because of the enforced waiting period. Therefore it is expected that jitter reduces the effective bandwidth of the network, which is expected to be visible in high-load scenarios.

Performance improvements resulting from applying jitter have already been observed through simulations in the work described in [CH01]. However, these results were achieved by introducing jitter on the control traffic of OLSR, which is very sparse compared even to the lowest traffic loads that are simulated. According to the discussion above, this low traffic load should lend itself well to jitter. Hence a performance increase of similar proportions in these simulations is *not* expected.

5.7.2 Results

In accordance with the expected results, the delay will be examined first, followed by the number of collisions. These metrics are expected to change under influence of jitter. After that, the effectiveness is analysed.

Delay

As expected, adding jitter causes additional delay proportional with the amount of jitter introduced. This is illustrated by figures 5.25 and 5.26

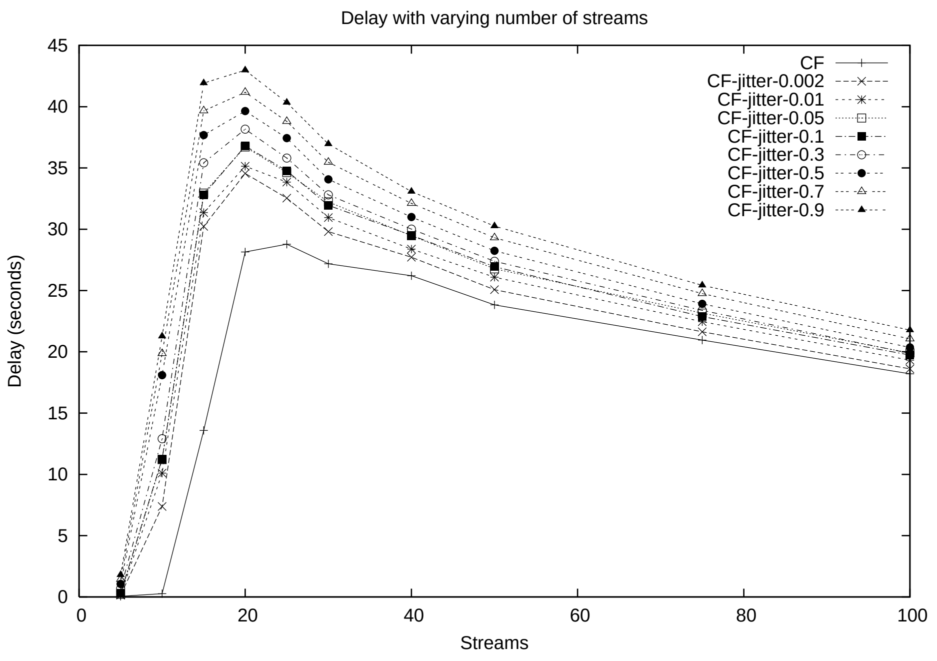

Figure 5.25: End-to-end delay with CF.

Collisions

Figures 5.27 and 5.28 show the number of collisions occurring. With both protocols, jitter reduces the collisions at high stream-counts.

For CF the reduction is 5-8% by introducing jitter at 20 streams or more. At less than 20 streams, jitter causes a 2-40% increase in collisions.

For DSF adding jitter reduces the collisions 8-11% at 25 streams or more. At less than 25 streams, no significant changes are observed.

Effectiveness

Figures 5.29 and 5.30 show the reliability for CF and DSF.

For CF at 5 streams, jitter provides a reliability increase of 184%. This increase drops to 2% at 15 streams, and at 20 streams or more, the difference between CF with and without jitter is less than 1%.

For DSF, adding jitter increases the reliability at all stream-counts. At 5 streams DSF is 73% higher, and at 100 streams DSF is 4% higher.

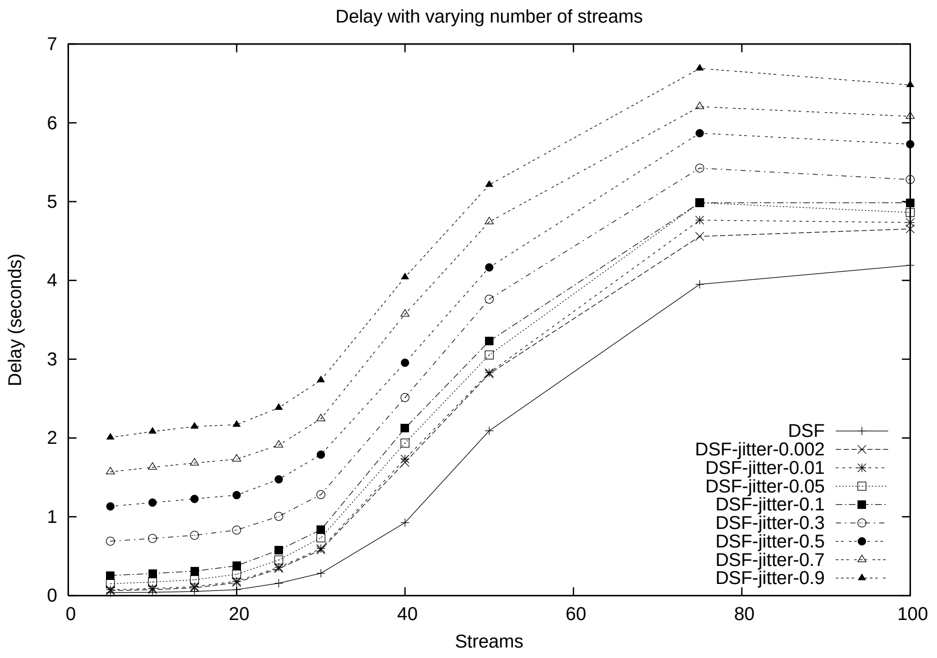

Figure 5.26: End-to-end delay with DSF.

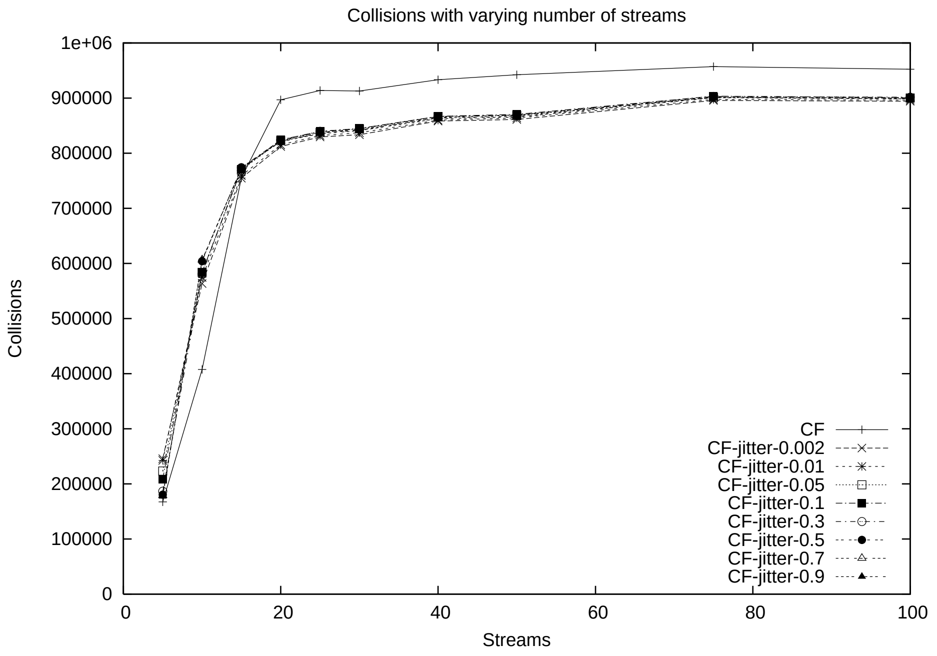

Figure 5.27: Collisions with CF.

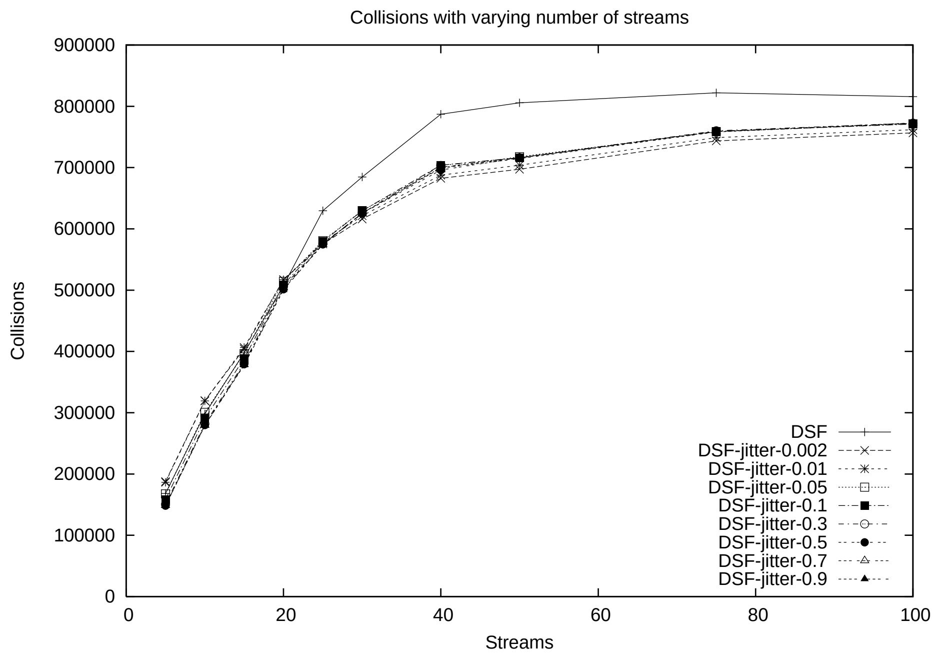

Figure 5.28: Collisions with DSF.

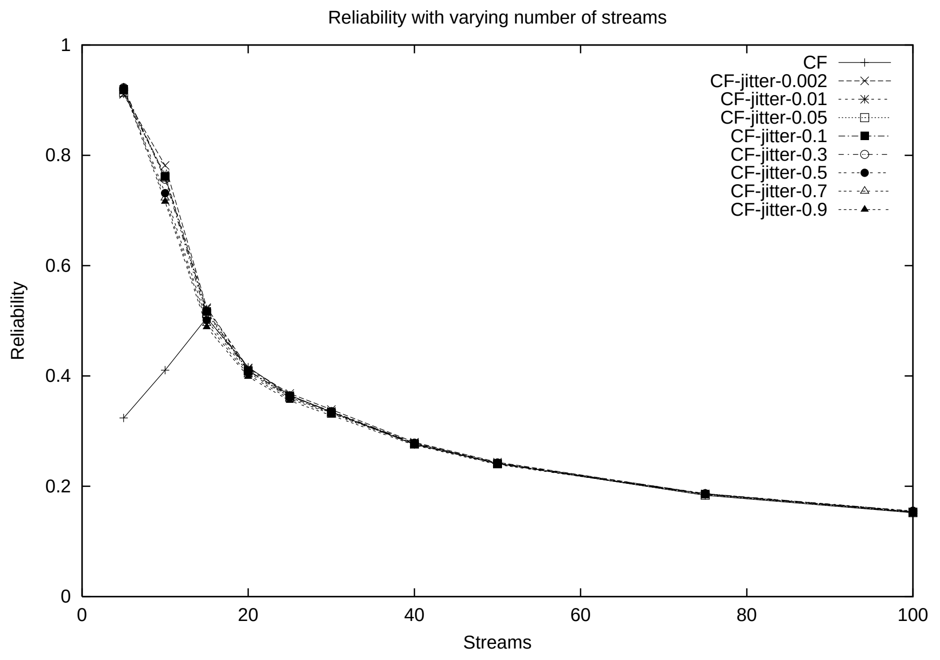

Figure 5.29: Reliability of CF.

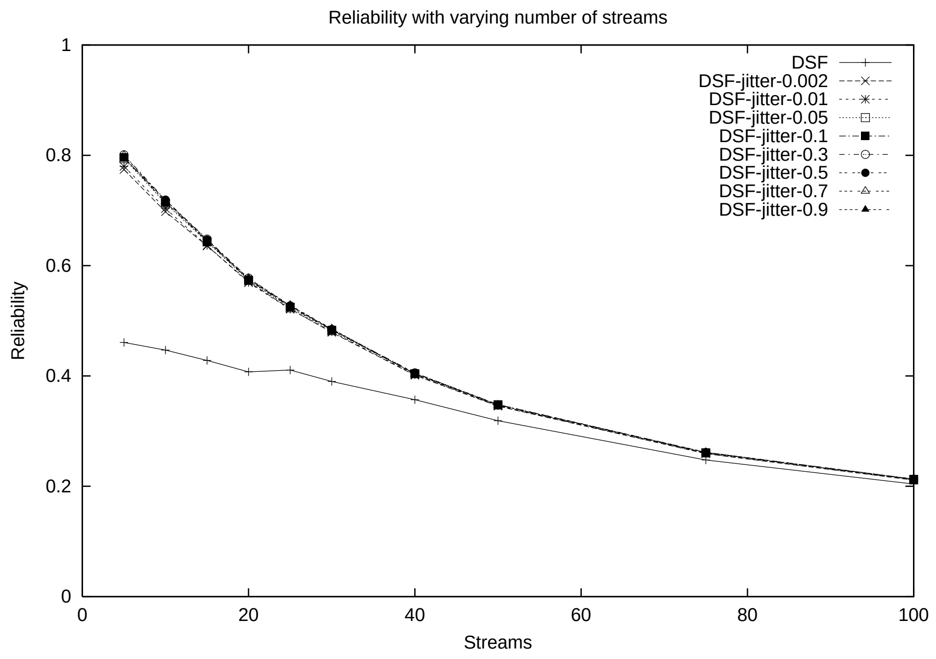

Figure 5.30: Reliability of DSF.

Figure 5.31 illustrates that adding jitter to CF decreases the efficiency 3-49%, whereas Figure 5.32 shows that adding jitter to DSF *increases* the effectiveness 5-59% (at 40 streams or less). At more than 40 streams, DSF is not affected by adding jitter.

The reliability and efficiency of CF and DSF with jitter results in the effectiveness results show in figures 5.33 and 5.34.

For CF the effectiveness is only higher with jitter at 5 streams (45%). At 15 to 75 streams, CF without jitter is 2-28% higher.

For DSF the effectiveness is higher when jitter is used at all stream-counts. The increase varies from 175% at 5 streams to 4% at 100 streams.

5.7.3 Conclusions

Adding jitter when forwarding broadcast data packets is shown to decrease the amount of collisions that occur when 25 streams or more are active.

Also, jitter increases the delivery rate (reliability), particularly at low loads. The largest increase is found with CF at 5 streams, where the delivery rate is increased from 32% to 92%. For DSF at 5 streams, the delivery rate is increased from 46% to 80%.

The primary cost of adding jitter is the associated end-to-delay. However,

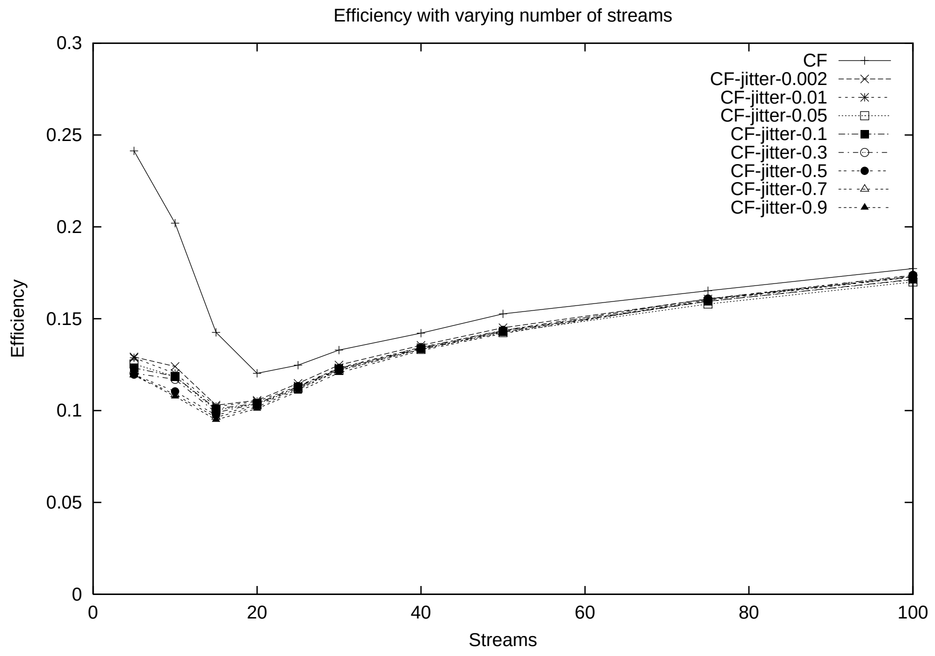

Figure 5.31: Efficiency of CF.

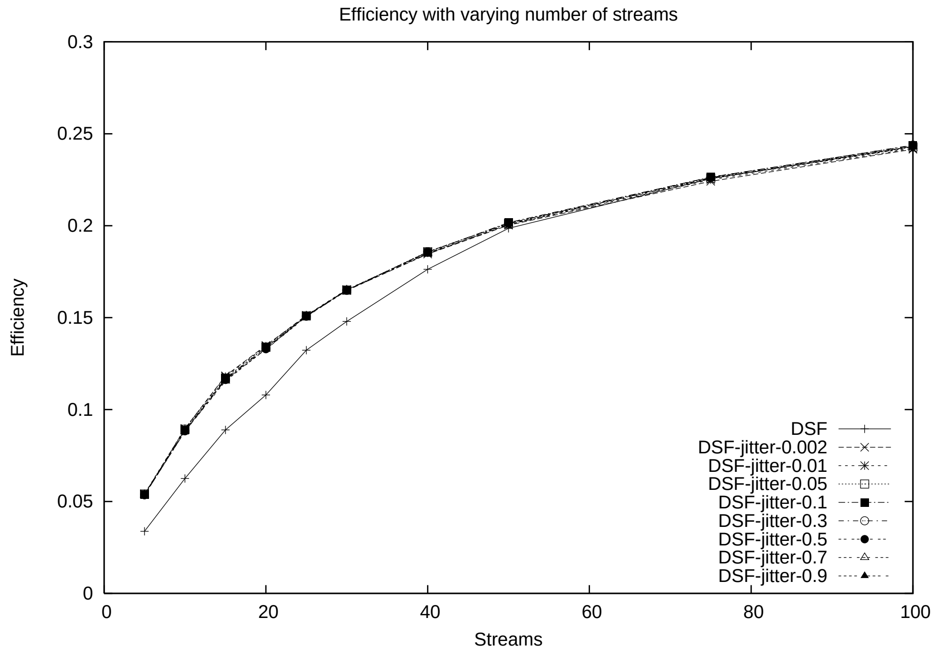

Figure 5.32: Efficiency of DSF.

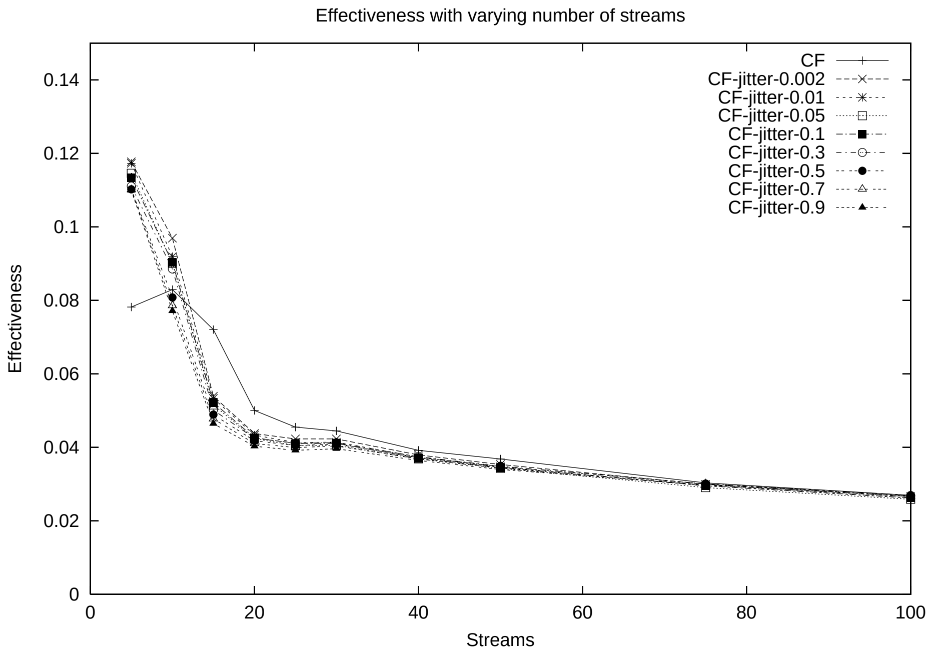

Figure 5.33: Effectiveness of CF

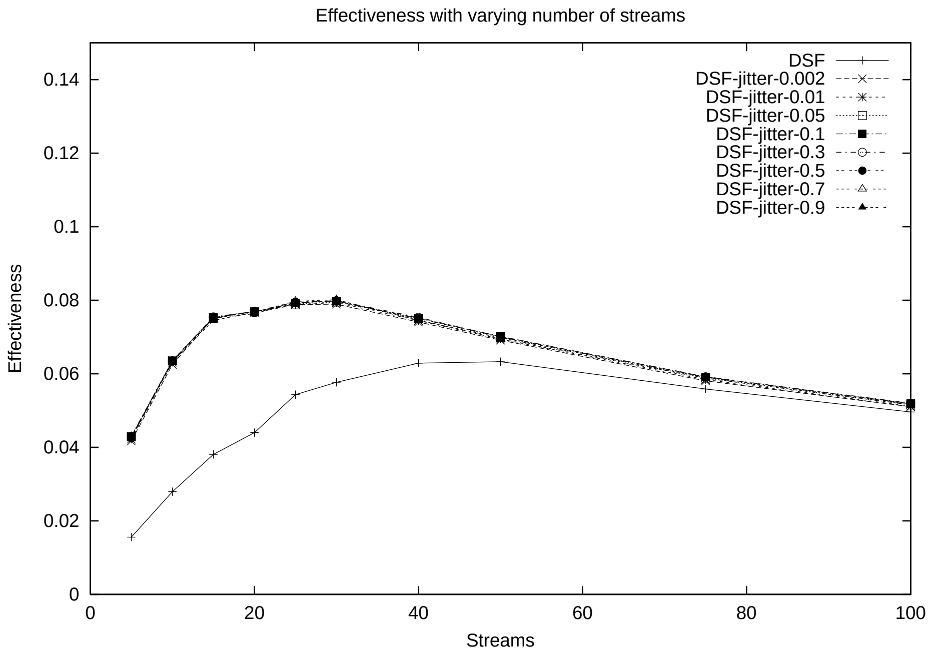

Figure 5.34: Effectiveness of DSF

it is observed that delay is the only metric where the *amount* of jitter is important, and the lowest amount of jitter provides the same benefits as the highest amount of jitter, while providing shorter end-to-end delays.

5.7.4 Summary

In this test the effect of adding jitter to broadcast data packets was examined, and three primary observations are made:

1. Adding jitter decreases the occurrence of collisions.
2. With few streams jitter provides a significant increase in reliability.
3. Increasing the amount of jitter added does not increase the benefit of jitter, but only adds to the end-to-end packet delay.

5.8 Test 5: Multipacket Flooding

A problem inherent in MANETs is reliability. The packet loss rate in a MANET is higher than in wired networks [CM99]. We wish to investigate a simple technique that might improve the reliability: sending multiple copies of each packet. This approach will naturally incur a greater overhead (linear in the number of packet copies), but how many additional successful deliveries can, say, two extra copies of each packet buy?

If a node receives several copies of the same packet, only one is counted as a successful delivery – the other ones are overhead.

To implement multipacket flooding two issues must be settled: First, how many copies of each packet should be sent. Second, when should the packets copies be sent? We have decided to test three packet multipliers (2, 3, and 5).

The test includes byte rates 192, 384, and 768, and the protocols CF and DSF. This yields six protocols in three load conditions, amounting to 5400 simulations.

5.8.1 Expected Results

We expect that sending multiple copies of a packet in a low-load scenario will result in more packets reaching their destinations, since the media should be able to carry the extra packets. We do, however, expect the price of the extra delivered packets to be high. In high-load scenarios the additional traffic might only disturb the existing packet flow, causing collisions and resulting in a decreased delivery rate.

5.8.2 Results

The following sections describe the effectiveness metric, followed by an overview of the collisions and the bandwidth utilisation.

Effectiveness

Figures 5.35 and 5.36 show the reliability of CF and DSF with and without multipacket flooding.

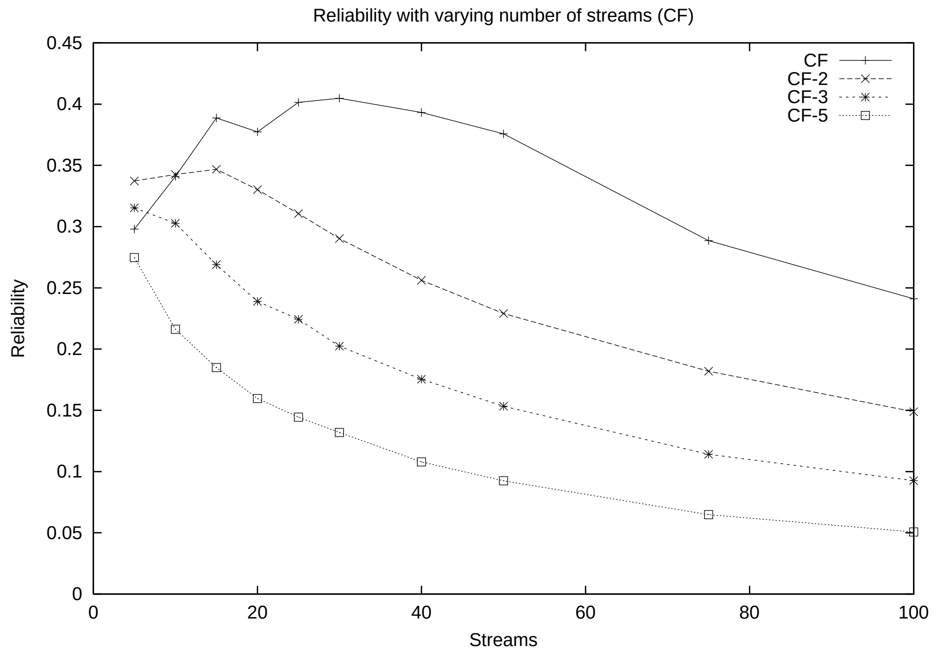

Figure 5.35: Reliability of CF.

For CF, multipacket flooding decreases the reliability significantly at 15 streams or more, and the more copies that are sent, the lower the reliability. At 5 and 10 streams, multipacket flooding (using a multiplier of 2) provides up to a 13% better reliability.

DSF displays a similar behaviour, except that even at 5 and 10 streams, multipacket flooding has a lower reliability than DSF without multipacket flooding.

Figures 5.37 and 5.38 show the efficiency of CF and DSF with and without multipacket flooding.

With 40 streams or less, CF without multipacket flooding shows the highest efficiency. With more than 40 streams a multiplier of 2 provides an efficiency increase of 7-19%.

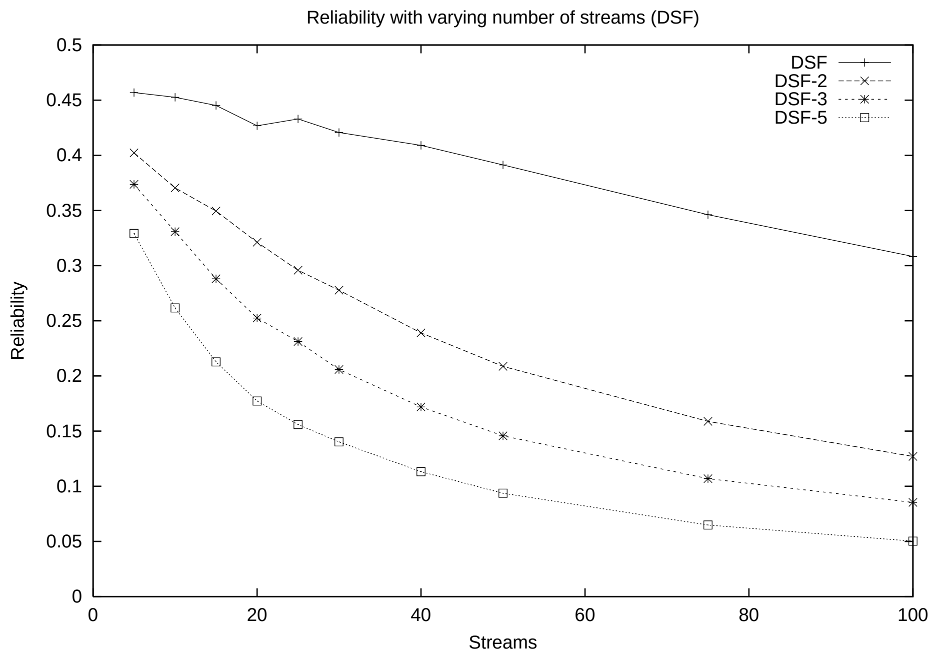

Figure 5.36: Reliability of DSF.

DSF has very different characteristics. At 40 streams or less, a multiplier of 5 provides an increase of 11-198%, but at 50 streams or more the multipacket flooding multiplier with the highest efficiency (a multiplier of 2) decreases efficiency by 4-22% compared to DSF without multipacket flooding.

The effectiveness graphs resulting from these reliabilities and efficiencies are shown in figures 5.39 and 5.40.

What is interesting is that for CF in all conditions, multipacket flooding decreases the effectiveness significantly, and the more packet copies that are sent, the worse gets the effectiveness result.

With DSF multipacket flooding provides the best effectiveness at 15 streams and less, and at 5 streams the best effectiveness is found when transmitting 5 copies of each packet, which provides a 107% increase.

Collisions

Figures 5.41 and 5.42 show the average number of collisions that occur with CF and DSF. The figures illustrate that multipacket flooding causes collisions.

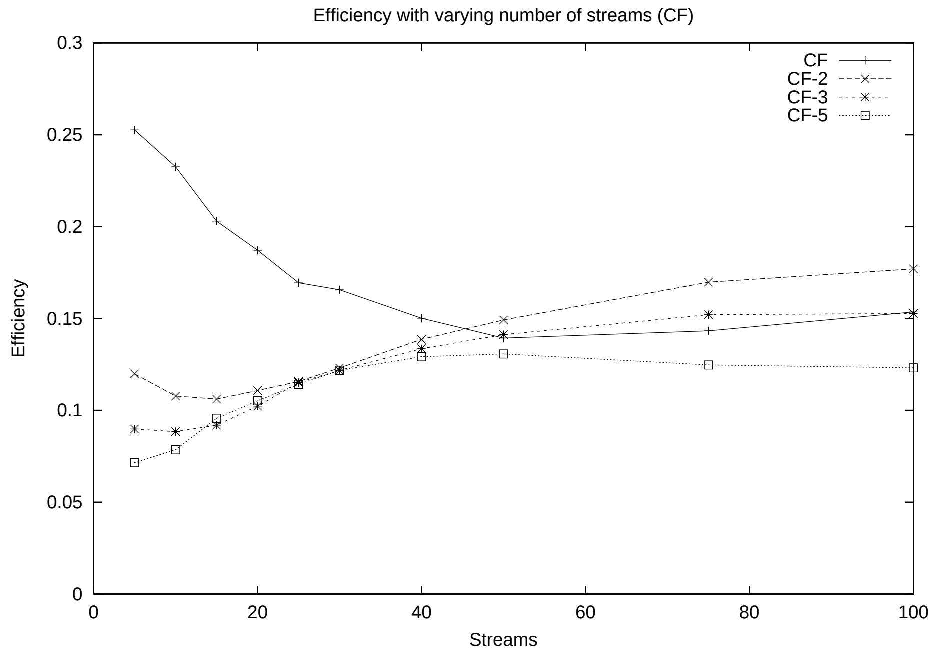

Figure 5.37: Efficiency of CF.

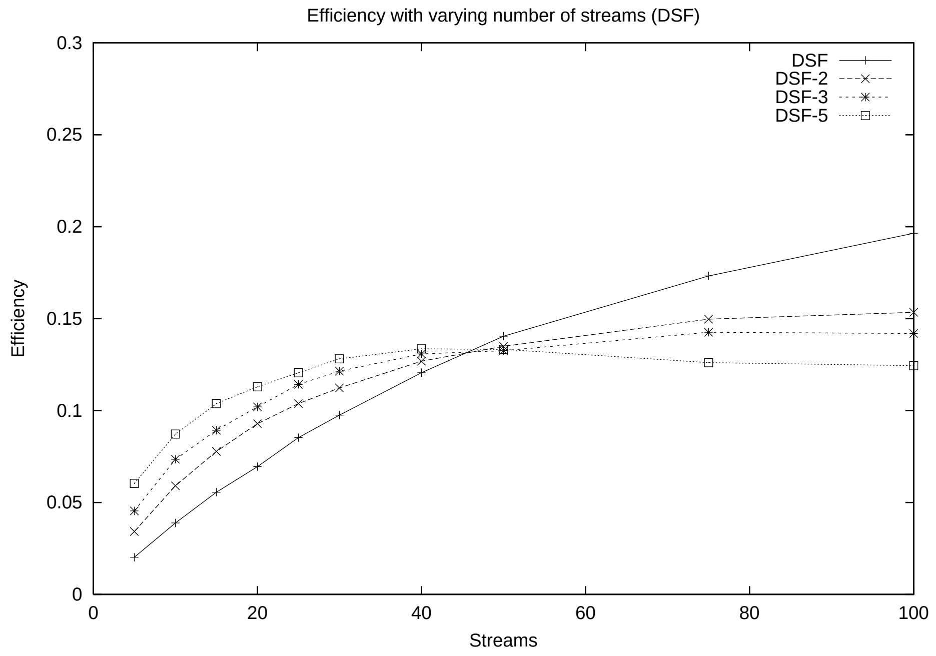

Figure 5.38: Efficiency of DSF.

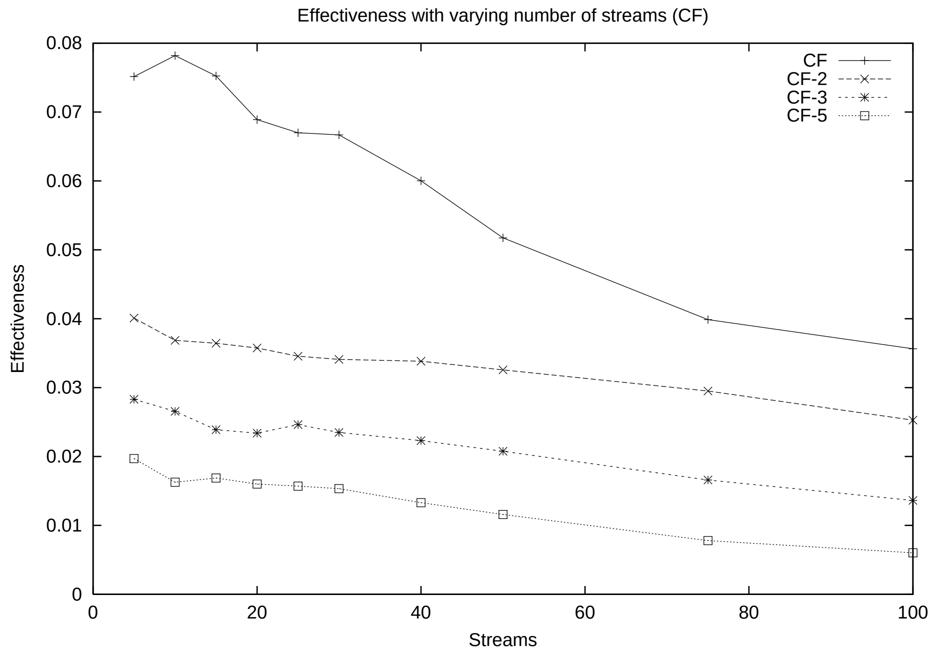

Figure 5.39: Effectiveness of CF.

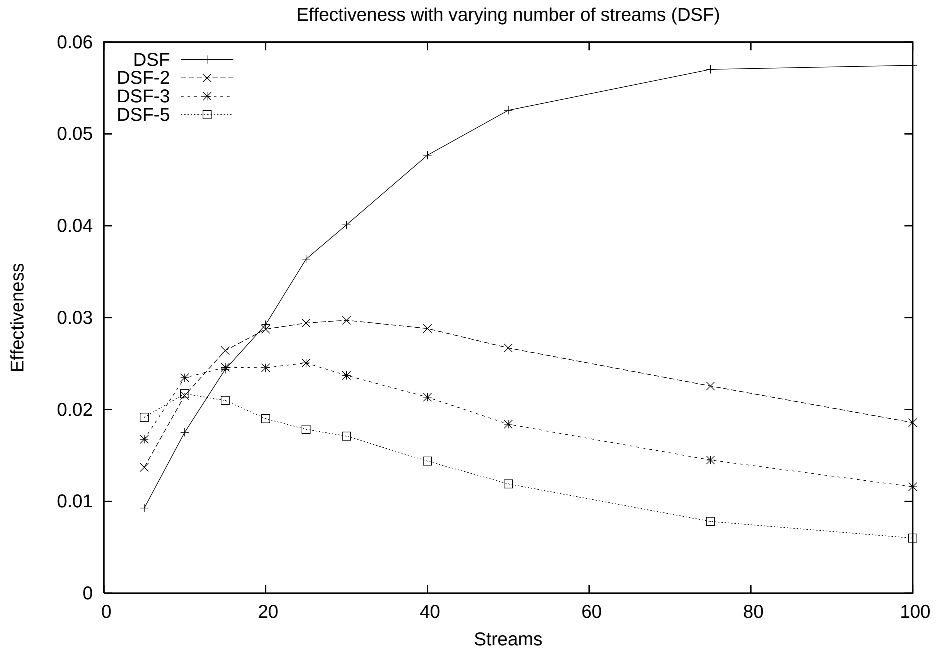

Figure 5.40: Effectiveness of DSF.

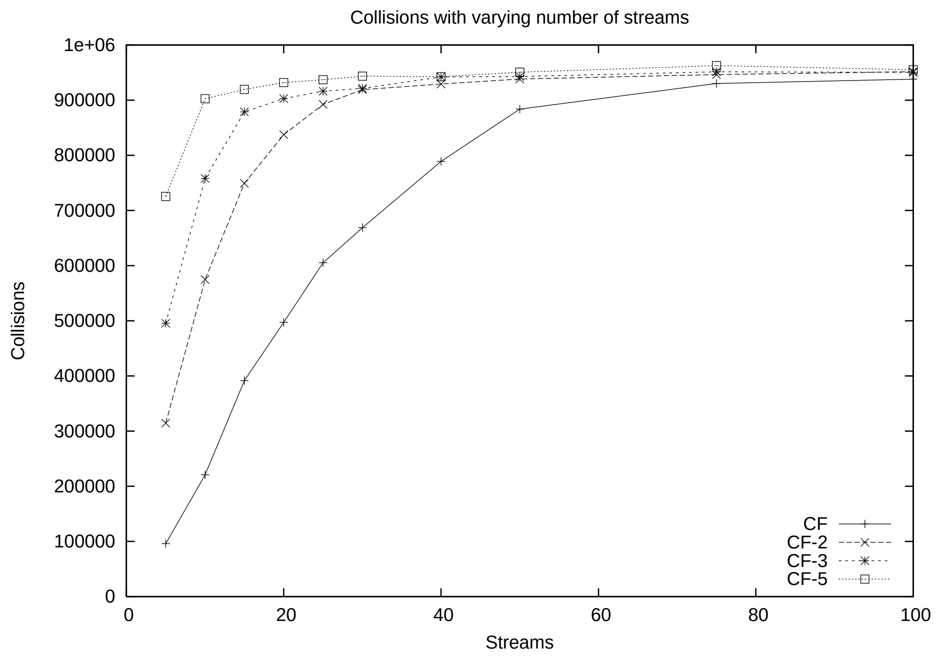

Figure 5.41: Average number of collisions for CF.

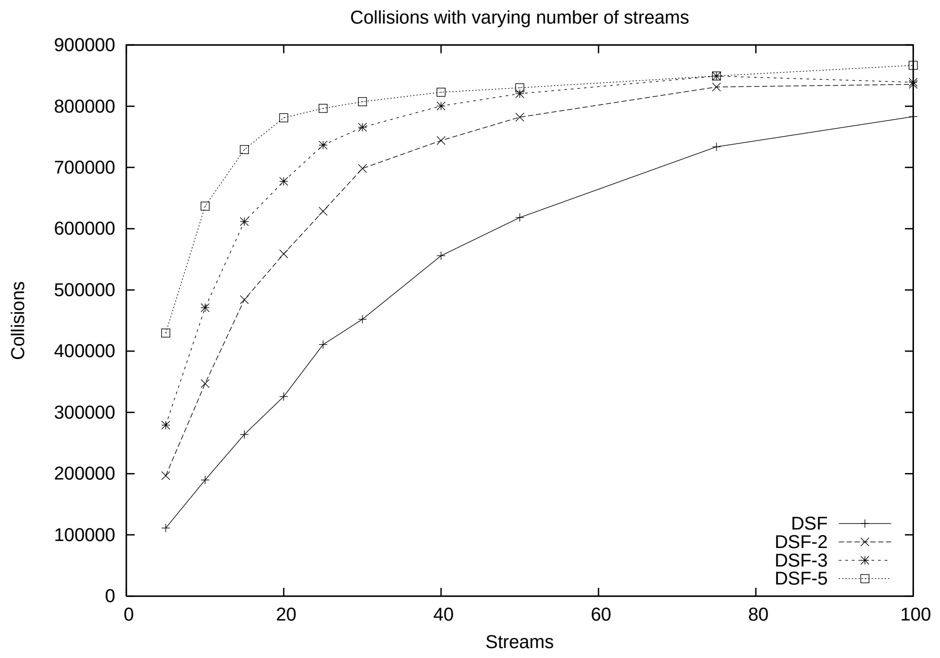

Figure 5.42: Average number of collisions for DSF.

Bandwidth Utilisation

Figures 5.43 and 5.44 show the average bandwidth utilisation for CF and DSF with and without multipacket flooding. These figures illustrate that – as expected – an increased multiplier results in increased bandwidth.

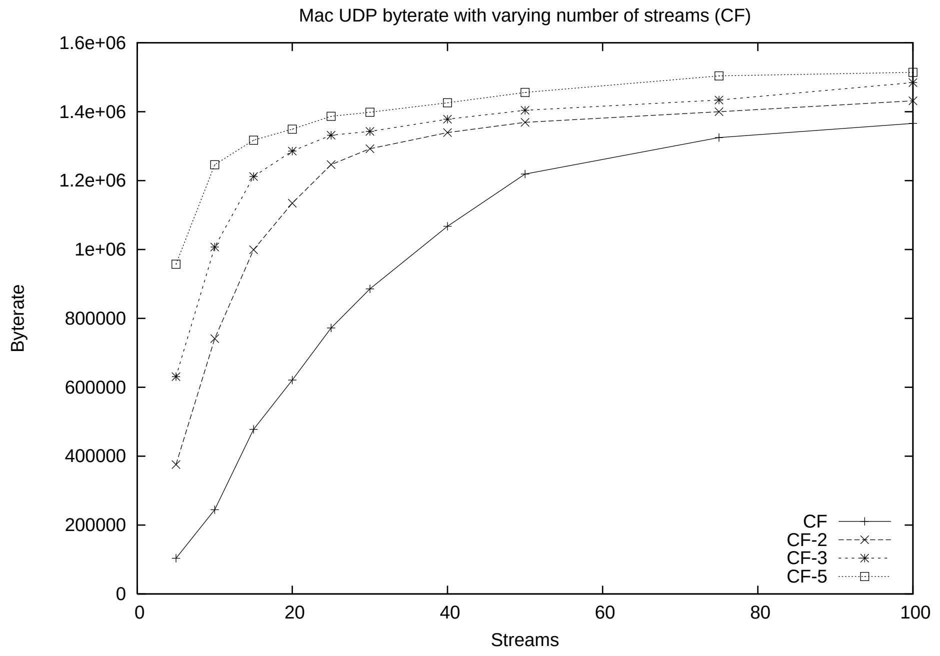

Figure 5.43: Average bandwidth for CF.

5.8.3 Conclusions

It is observed that increasing the packet multiplier causes more collisions in the network, and results in increased bandwidth utilisation.

In the majority of the tests, multipacket flooding lessens the reliability of a protocol significantly.

The protocols achieve very different results regarding efficiency. With CF, multipacket flooding can increase the efficiency at 50 streams or more, while with DSF, multipacket flooding can increase the efficiency at 40 streams or less.

Regarding effectiveness, multipacket flooding results in a significant decrease.

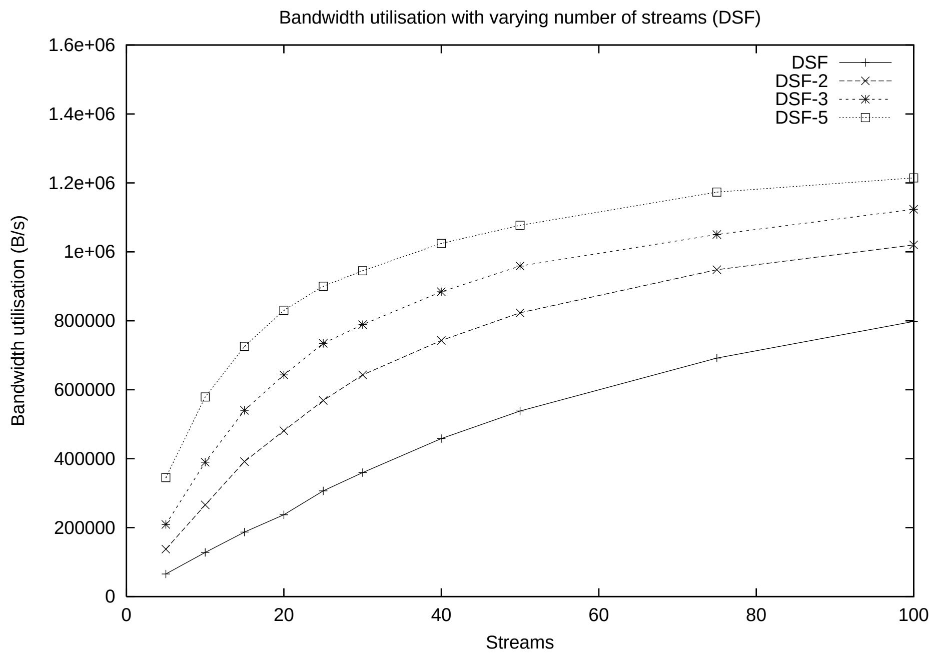

Figure 5.44: Average bandwidth for DSF.

5.8.4 Summary

Multipacket flooding was expected to increase reliability, at least at low network load conditions. This has been shown not to be the case. Multipacket flooding decreases the reliability, and increases the bandwidth overhead.

5.9 Conclusions

Test 1 identifies MPRF as the protocol that achieves the best performance. MPRF achieves both the highest delivery rate, the highest efficiency, and the the highest effectiveness. RPF is found to have the lowest delivery rate and effectiveness. DSF achieves results close to those of MPRF. The test also reveals that none of the protocols yield a performance gain over CF that suffices for using them for reliable data delivery.

Test 2 shows that the presence of OLSR control traffic in CF simulations increases the reliability of the protocol. The reason has been found to be due to fewer undetected duplicate packets. Also, the OLSR traffic adds collisions to the network, causing the efficiency of CF to degrade in the presence of OLSR.

Test 3 confirms that the packets which traverse paths longer than 100 hops may be undetected duplicate packets.

Test 4 determines that jitter on broadcast data decreases the number of collisions in the network, and increases the delivery rate. It is also shown that increasing the amount of jitter does not yield further improvements.

Test 5 evaluates multipacket flooding as a mechanism for improving reliability, and reveals that multipacket flooding does not yield improvements to this end. Instead, multipacket flooding *degrades* the delivery rate, and adds bandwidth overhead.

5.10 Summary

Five tests have been conducted to evaluate the performance of four broadcast protocols, and two generic protocol extensions intended to reduce collisions and improve reliability.

The tests have revealed that the MPRF protocol achieves the best overall results regarding delivery rate and bandwidth consumption. Also, the maximum delivery rate achieved by any protocol is 51%. Applying jitter on transmissions increases the delivery rate. In the best case observed, the delivery rate is increased from 38% to 92%.Multipacket flooding degrades the performance of the protocol to which it is applied. The bandwidth overhead is increased, and the reliability is decreased.

The next chapter evaluates the simulation framework, based on the experiences gained in conducting the simulations.

Chapter 6

Simulation Framework Evaluation

This chapter evaluates the simulation framework, with respect to how well it fulfils the task of automating the simulating process. Section 6.1 introduces the aspects which are evaluated. Section 6.2 evaluates individual components of the framework. Section 6.3 presents issues discovered while adapting the framework to work across multiple platforms, and section 6.4 documents the experiences with using the framework, gathered throughout this work. Last, section 6.5 summarises the chapter.

6.1 Introduction

The simulation framework presented in chapter 4 has been developed and applied to conduct the simulations simulations presented in chapter 5. Through this, experiences as to the performance and convenience of the framework has been gathered

The purpose of the framework evaluation is to examine whether it fulfils the goal it is intended for: automation of the simulation process. Three aspects of the framework are evaluated: the individual components, the cross-platform portability of the framework, and the general experiences gained through using the framework for conducting simulations.

6.2 Evaluation of Individual Components

This section evaluates three of the framework applications are concerned: the job scheduler, *Tafat*, and *Grace*. Also, the method applied for data storage is evaluated.

Two of the framework applications, ns2 and *Sump* (the summary processor), are not evaluated in detail. Evaluating the quality of ns2 as a network simulator is outside the scope of this work. At this point it is noted that ns2 has fulfilled its role as the network simulator in in the simulation framework. As for *Sump*, it is noted that the tool fulfilled its purpose in the framework, generating summaries of the result files.

6.2.1 Job Scheduler

The scheduler has been used to schedule all of the simulations for this work, distributed across 26 machines in three different locations. The functionality present in the scheduler has proven sufficient for scheduling jobs among a set of machines residing on the same local area network: on several occasions, a set of machines has been left unattended for days, while performing a large batch of simulations.

During the use, a number of issues present in the current version of the scheduler have been discovered, and ideas for future extensions has emerged. The two major issues present in the current version of the scheduler are as follows:

- Scheduling of jobs does not work between different hardware platforms. The cause of this issue has not been exactly determined, but is very likely related to the different byte order of the Intel and Sparc platforms.

- The protocol used for network communication is not robust to network failures: it is based on TCP connections between the job manager and the job servers, and does not tolerate that these connections are broken due to, e.g., network partitioninig.

Using an environment of both Intel and Sparc machines, the first issue has been addressed by running a scheduler for each hardware platform. The second issue has been addressed by running separate schedulers for each local area network in use. In combination, these workarounds made it necessary to run four separate schedulers, as illustrated in figure 6.1, where one could ideally have been sufficient.

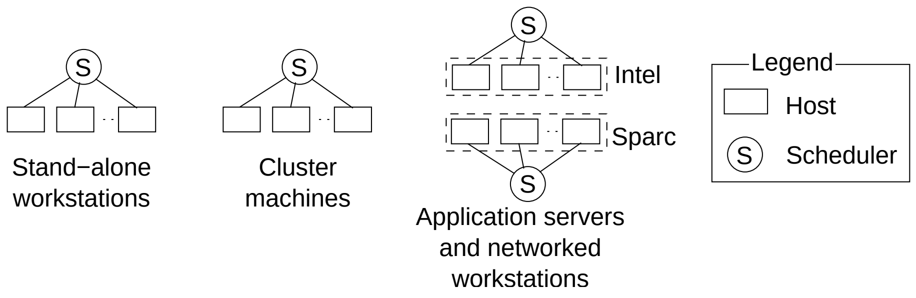

Figure 6.1: Four schedulers were necessary to keep simulations on all machines running.

Obviously, the existing issues must be addressed to make the scheduler capable of scheduling jobs across different hardware platforms, and tolerant to network failures.

Naturally, the existing issues should be addressed to obtain a more robust scheduling system, enabling cross-platform job scheduling. Further experience has revealed ideas for improvements that would add to the convenience of the job scheduler:

- adding journaling to enforce "at-least-once" execution of jobs, *and*
- adding multi-user capabilities to manage the scheduling of jobs on shared machines.

The scheduler has provided a considerable aid in the conduction of simulations in this work. The experiences with job scheduling motivates the use of batch scheduling systems for conducting simulations, and an improved version of the scheduler developed in this work will be suitable for such uses.

6.2.2 Trace File Analysis Tool

Tafat was implemented to suit all the trace file analysis requirements for the simulations in this work. With the goals of performance and modularity in mind, *Tafat* were intended to replace two other tools for trace file analysis: one implemented in Ruby, providing modularity but suffering in performance, the other implemented in `awk`, yielding acceptable performance, but lacking modularity. The performance and modularity of *Tafat* will be evaluated independently.

Performance evaluation

To evaluate the performance of *Tafat*, a comparison with the `awk` and Ruby trace file analysis implementations is conducted. The comparison illustrates the scalability of the three implementations, by observing their running time on various sizes of trace files.

For the comparison, a trace file of 627 MB (6.467.300 lines) has been generated by simulating one of the scenarios used for evaluating the MPR Flooding protocol[1]. Trace files of this size are not uncommon for the simulations in this work; the smallest trace files are around 130 MB, and some simulations of some scenarios has been observed to generate trace files exceeding two GB.

[1]The common scenario parameters are as specified in section 5.3. The traffic load in the evaluated scenario was generated by 50 streams of 2304 KB/sec each.

The *Tafat*, `awk` and Ruby implementations provides different suites of analysis modules. This evaluation measures the performance of each tool, using only the modules for each implementation that provide a set of common results.

Figure 6.2 shows the time spent executing user space processes for each of the *Tafat*, `awk` and Ruby implementations, while performing trace file analysis on inputs of different sizes. The results have been obtained using a 733 MHz Pentium III PC, reading the the tracefile from disk cache to avoid disk I/O overhead.

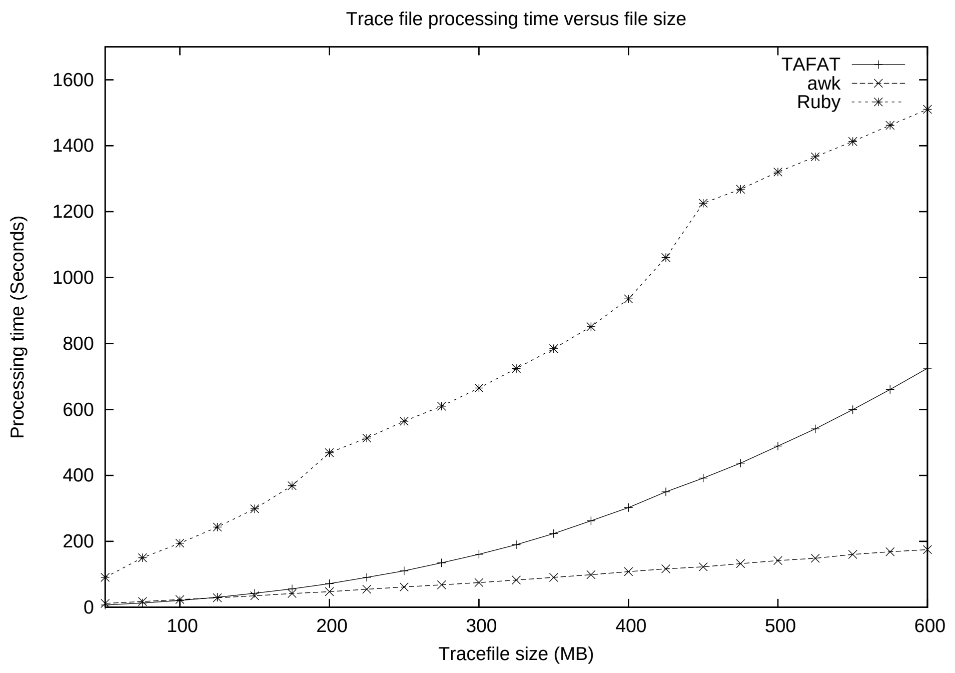

Figure 6.2: Scalability of trace file analysis tools.

From figure 6.2, it is observed that the Ruby implementation performs significantly worse than both *Tafat* and the `awk` implementation. The `awk` implementation is faster than *Tafat* for trace file sizes exceeding approximately 125 MB, and linearly when the file size is increased. the runnig time of *Tafat* grows faster than linearly, ending at 414% of that of the `awk` implementation at the maximum file size of 600 MB.

This result show that *Tafat* scales better than the Ruby implementation, but for tracefiles larger than 125 MB, *Tafat* is outperformed by the `awk` implementation.

Modularity

The modular structure of *Tafat* is ensured by separating the functionality for parsing trace files from the task of performing certain types of analysis on the parsed data. The latter task is further divided into a number of modules (called *executables).* Ordered activation of these is ensured by grouping them into the ordered sets: "pre-analysis", "analysis" and "post-analysis".

Comparing this solution with the object-oriented organisation present in the Ruby implementation, it is observed that *Tafat* has achieved a corresponding degree of modularity. The main differences between *Tafat* and the Ruby implementation is the absence of object orientation in *Tafat*, and the fact that memory management must be handled explicitly, a task solved automatically by the Ruby interpreter.

Tafat is successful in providing the modularity necessary for implementing executables that maintain their internal state independently, but at the same time provides functionality for sharing the information collected by certain executables among the remaining ones.

6.2.3 Graph Compilation Environment

Given a set of results from *Tafat*, *Grace* generates a standard set of graphs from these results. Using the input and output modules described in section 4.3.6, these standard graphs have been plotted for all the simulation results produced in this work. Through the use of *Grace*, a printable overview containing all the automatically generated graphs can be obtained through the following steps:

1. define a configuration file describing the input for *Grace*,
2. run *Grace*, using this configuration file,
3. run LaTeX on the output files of *Grace*, to obtain a printable postscript file.

Although it is not regarded as being critical with respect to *Grace*'s functionality, one notable issue of is that *Grace*'s memory usage grows large (observations show memory usages between 100 and 200 MB, depending on the summary data for the graphs) when the summary data is stored in memory for graph generation.

The existing interface between the sample space and the input/output modules in *Grace* is simple, and does not do any sanity checks on queries for storing or retrieving data. This simplicity requires considerable robustness of the input parsing, and of the output modules to retrieve the data correctly. The existing version of *Grace* lacks robustness in both parts, resulting in errors when generating graphs from a set of result files containing unexpected or no values, or when using an erroneous configuration file.

The issue of robustness should be addressed to increase the usability of the current version of *Grace*. As for the scheduler, several ideas for future development have emerged during the use of *Grace*:

- development of a more advanced interface for the sample space to ease the integration of new summary data formats and graph types,
- a more scalable approach to storing the sample space – possibly organised as a database stored on disk.

6.2.4 Data Storage

Despite the trace files from ns2 not being stored persistently, large quantities of files are still stored by the current methods for generating computing statistics, and plotting graphs. Scenario parameter files, scenarios, result files, summary files and finally, graphs are stored to persistently.

A network-accessible Rsync data repository has been used for the purpose of centralising file storage and enabling remote machines to access the files. Organising the diverse set of different data into files has proven a difficult task. It has been solved by combining file and directory names to obtain a hierarchical storage, where information about the *contents* of a file is expressed by *the path* of the file, as, e.g., the following path of a result file:

```
/flooding-020325/results/load_192_40/load_192_40_MPRFlood_12.results
```

This path is result file is from the set of simulations called "flooding-020325", and contains results for the 12th simulation in a set of 30 scenarios, all having 40 CBR streams sending transmitting 192 B/s each.

The semantics encoded in the directory and file names are reflected in the applications and scripts that navigate the directory tree and read or write the files. This dependency is undesirable, as changes in the file organisation must be reflected in corresponding changes in the applications.

For future work, a more desirable solution would be to develop a "simulation data storage API", which *unites* the information about the contents of a file with the contents themselves, and provides a layer of abstraction over disk files, through which more advanced and efficient query techniques than those possible by ordinary file system usage are possible.

Summarising the evaluation of the data storage, network access to a central data repository has proven practical when conducting simulations across several machines in distinct locations. The current organisation of data the repository is not sufficiently flexible for general use. One way of improving the generality of the data storage, but its generality could be improved by use of a general simulation data storage API

6.3 Portability Issues

The requirement for running common sets of simulations on both Intel/Linux, Intel/Solaris and Sparc/Solaris machines has presented a few issues, which will be shortly clarified in this section.

It is noted that the applications in the simulation framework has been implemented in several programming languages: ns2 is implemented in C++ and Tcl, *Tafat* is implemented in C, whereas Ruby has been selected as language for the scheduler, and for *Sump* and *Grace*. In addition, a suite of shell scripts and makefiles automate several routine tasks.

One minor issue experienced while porting *Tafat* from Linux to Solaris required a solution: *Tafat* uses the error reporting functionality provided by GNU Libc library [MD02]. This library is absent on the Solaris development platform, which forced the implementation of a small piece of replacement code used when compiling *Tafat* for Solaris.

The Tcl and Ruby code is interpreted, and as interpreters exist for both Linux and Solaris, no problems with respect to either of these languages has been experienced.

Differences in the possibilities for obtaining hardware status information required some operating system specific code to be integrated in the scheduler. This enables job servers to automatically discover the number of CPUs and amount memory present. On Linux machines, such information may be obtained via the `/proc` hierarchy of the file system, while use of certain executables, e.g., `psrinfo` is necessary on the Solaris machines.

Overall, only minor portability issues were experienced during the framework development process, and solutions have been established to solve the few issues that were observed.

6.4 Overall Experiences

In several situations, manual work was necessary for starting simulations. This ought to be a simple and straightforward task, but a number of reappearing problems were present.

Generality. If a script or an application is implemented to fit specific machines used for simulation, this is likely to cause problems. Code appearing in the framework are likely to be run at different machines at some point during their "lifetime". For this to be possible, generality must be prioritised.

Robustness and verbosity. When the framework applications are used for conducting several different sets of simulations, it is not unlikely that at some point a situation occurs where one application is, e.g., provided with input which it cannot process. Robust applications are desired if the errors are non-critical, so that the processing can continue. In case of critical errors, a verbose error message is desired, to help locating the cause of the problem quickly.

Further, when unattended batches of simulations are executed, the failure of a single simulation should be logged for later investigation and possible re-scheduling, but tolerated to the degree that the remaining simulations can be completed.

Consistent Application Configuration. Some applications in the framework, e.g. ns2 and *Tafat*, are configured partly at compile-time, by editing the source code, and partly by supplying command line options and configuration files when started. This leaves open the possibility for errors to go undetected, due to accidental reuse of binaries compiled with the wrong compile-time configuration.

To avoid such errors, a consistent pattern should be applied for application configuration, preferably avoiding compile-time configuration to the largest possible extent.

Version Control. It is important to have strict control of the different versions of the framework utilities, as some versions are not compatible, and others contain bugs which makes them unusable. For this purpose, "snapshots" of the code that has been used for production were created.

On several occasions, this system proved useful, as old versions of especially ns2 and *Tafat* were retrieved with the purpose of re-running sets of simulations.

6.5 Summary

Using the simulation framework described in chapter 4, all the simulations for the broadcast protocol evaluation presented in chapter 5 has been performed.

The general experience gained through this work is, that the wide range of conditions under which the framework must operate, require a generic, easily configurable and robust set of applications to constitute a succesful framework.

Further, the framework has reduced the manual work required to conduct large quantities of simulations, as was the overall goal for its functionality. None the less, the framework is still subject for future improvements.

The current data organisation is less flexible than could be desired, and issues to be resolved exist for several applications in the framework. Also, ideas for new improvements have emerged. Addressing those issues and ideas would result in increased robustness and improved functionality of the framework.

Chapter 7

Conclusions

Two problems have been solved in this work: the selection, specification and evaluation of MANET broadcast protocols, and the development and use of a simulation framework to aid the simulation based evaluation of the protocols.

Each of these solutions are concluded in the following: section 7.1 lists the products of this work, and section 7.2 concludes on the results which have been achieved. Section 7.3 gives directions for future work, and section 7.4 summarises the conclusions.

7.1 Products

This section gives an overview of the products established during this work, divided in two categories: products established during specification and evaluation of MANET broadcast protocols, and products established during development of the simulation framework.

7.1.1 Broadcast in MANETs

Four broadcast protocols are selected for evaluation, all fulfilling the criteria of requiring only the functionality present on basic MANET nodes. The protocols are selected from existing MANET routing protocols, as well as from the field of broadcasting in wired networks. Furthermore, two generic protocol extensions are selected for evaluation. During the study of the broadcast protocols and protocol extensions, the following products have been established:

- a scheme for duplicate packet elimination is specified,
- simulations of the selected protocols and extensions have been conducted, *and*
- metrics for evaluating the protocols have been specified and applied.

Additionally, the main results of the protocol evaluation is to appear on WPMC 2002 [wpm02], in a paper co-authored by the authors of this work.

7.1.2 Simulation Framework

A simulation framework has been developed to reduce the manual work involved when conducting simulations, by automating the simulation process. The development of the framework encompasses the following products:

- modification of the existing scenario generator *wsg* [CHCB01], to support scenario generation of MANET broadcast scenarios,
- modification of the existing network simulator ns2 [pro02], to support simulations using broadcast to all nodes in wireless networks, and implementation of the selected broadcast protocols,
- development of a job scheduling system for distributed conduction of simulations,
- development of *Tafat*, a trace file analysis tool,
- development of *Sump*, a result summary generator, *and*
- development of *Grace*, a graph generator.

7.2 Results

The main results of this work are the observed properties of the broadcast protocols and extensions, leading to a set of conclusions about the performance and behaviour of each protocol, and the experience gained from development and use of the simulation framework. Results for each part are summarised in sections 7.2.1 and 7.2.2 respectively.

7.2.1 Broadcast Protocols

The four broadcast protocols, Classic Flooding, MPR Flooding, Dominating Set Flooding, and Reverse Path Flooding have been simulated using the simulation framework developed. From the simulation results, an evaluation has been carried out, leading to the following conclusions:

Test 1 shows that MPR Flooding achieves the best results regarding reliability, efficiency and thus effectiveness. Dominating Set Flooding achieves results similar to those of MPR Flooding. Due to the fact that MPR Flooding has implementational disadvantages, Dominating

Set Flooding is selected as the protocol used for further testing. Reverse Path Flooding is outperformed by all the other protocols with respect to delivery rate and effectiveness. The highest delivery rate achieved by any of the four basic protocols is 51%, which is achieved by both MPR Flooding and Classic Flooding.

Test 2 shows that the presence of OLSR traffic in Classic Flooding simulations increases the amount of collisions, increases the delivery rate, and decreases the amount of undetected duplicate packets.

Test 3 simulates duplicate elimination with infinite history timeout, and shows that the duplicate elimination scheme has a problem when the history timeout is too low. A 15 second timeout is not sufficient to capture all duplicate packets, which makes it possible for duplicate packets to spin in the network until the TTL reaches zero, introducing bandwidth overhead.

Test 4 shows that applying jitter to the transmission of broadcast data leads to a significant increase in reliability (the best observed case is an increase from 38% to 92%), as a result of reducing the collisions in the network. The minimum amount of jitter tested was 0.002 second, and increasing the amount of jitter beyond this value does not increase the reliability further, but contributes to longer end-to-end delays. It is concluded that data jitter is a viable candidate for improving the delivery rate, and for reducing the amount of collisions.

Test 5 shows that multipacket flooding degrades performance with respect to reliability and bandwidth overhead, and is not a viable candidate for improving the reliability of broadcast protocols.

The two major conclusions drawn from the tests are that MPR Flooding achieves the best performance, and that jitter increases reliability.

7.2.2 Simulation Framework

The framework has been applied to conduct the 21,300 simulations from which the results are presented in this work. Through generation of 30 scenarios for each set of scenario parameters, the chance of scenarios that favours one protocol over another, thereby dominating the results, has been reduced.

All the simulations have been conducted on a platform of 26 machines in three different locations connected by networks. The key to achieving this parallelisation is the job scheduler, which provides a central point of management for the execution of simulations.

The current trace file analysis tool, written in C, provides better modularity than what have previously been achieved using `awk`, and better performance than achieved by the Ruby implementation, which has similar modularity properties.

A visual representation of the simulation results is a useful method for quickly gaining an overview over tendencies expressed by large quantities of results. The summary and graph generators enable automatic generation of such a representation from the results of the trace file analysis.

The overall conclusion is, that the framework has been successful in reducing the amount of work required to conduct simulations, and that such a reduction is desirable with respect to conducting extensive, simulation based protocol evaluations.

7.3 Future Work

The present work leaves topics to be addressed in the areas of both broadcasting in MANETs, and development of the simulation framework. This section gives suggestions for future work within each of these topics.

7.3.1 Broadcast Protocols

The following directions for future work in the field of MANET broadcast protocols are suggested:

Investigating open questions. Some questions raised from the simulation results in this work remain unanswered and could be the subject for further investigations. Some examples are mentioned in the following.

It is unanswered why a peak appears in many of the Classic Flooding graphs in test 1 (e.g., figure 5.4).

Also, the question remains open why adding the control traffic of OLSR to the Classic Flooding simulations results in decreases the number of undetected duplicate packets.

Last, the minimum amount of jitter tested is 0.002 second, which is shown to provide the same delivery rate increase as the other jitter amounts tested (up to 0.9 second). It is also shown that the higher jitter amounts result in higher end-to-end delays. A subject for further investigation could be to determine the minimum amount of jitter necessary to achieve the benefits observed in this work.

Testing other scenario parameters. Further simulations could be conducted with different parameter settings, or adopting new parameters.

For example, the effect of varying mobility has not been evaluated in this work.

New protocols and extensions. A course of action could be to investigate other broadcast protocols and other protocol extensions, to find or construct a protocol that achieve higher delivery rates than the protocols evaluated in this work.

7.3.2 Simulation Framework

The following directions for future work related to development of the simulation framework are suggested:

Addressing current issues in the simulation framework. The current issues present in the framework applications could be addressed, to improve the overall robustness of the framework. Important tasks in this respect are to address the network communication issues of the the job scheduler, and to improve the robustness and flexibility of input/output for *Grace*.

Simulation framework data storage. The file storage organisation used in the framework could be changed, to provide a common API for data access, available to all applications in the framework. Challenges in this task include providing a cross-platform, cross-language, networked solution with a performance matching that of the existing solution.

Scenario generation. Various new mobility models, with different properties than the random waypoint model, have been suggested. These mobility models could be implemented in the scenario generator. Furthermore, theoretical results on scenario node density calculation could be integrated to ease the specification of scenarios. Related work on both subjects are present in [Bet02].

Job scheduling. The present job scheduler could be extended to enable support for journaled scheduling, and allowing multiple users to share a common set of job queues. To accomplish the latter, a change in the current, single-user interactive interface to a set of shell commands accessible to multiple users could be convenient.

Graph generation. The graph generator could benefit from a more convenient organisation of the data storage. If queries for certain data are made possible, graphs could be generated by querying the data storage directly from persistent storage. This could reduce the memory requirements for generating graphs, solving one immediate scalability issue of the current implementation of *Grace*.

7.4 Summary

Two main problems have been considered in the present work: Protocol evaluation through simulations, and development of a simulation framework to aid the conduction of large amounts of simulations.

A set of MANET broadcast protocols and protocol extensions are evaluated through an extensive simulation-based study. It is observed that none of the protocols achieve delivery rates higher than 51%. MPR Flooding outperforms the other protocols in terms of delivery rate, efficiency, and effectiveness, and is concluded to be the best of the tested protocols. Jitter improves the delivery rate, and reduces the amount of collisions, and is so concluded to be a viable candidate for improving the reliability of broadcast protocols. Multipacket flooding, on the other hand, degrades the protocol performance by decreasing the delivery rate and introducing bandwidth overhead. Hence, multipacket flooding is not a viable candidate for broadcast protocol improvement.

A simulation framework is developed and applied to automate the process of conducting the simulations. The framework reduces the work of conducting the simulations considerably, and enables reductions in the simulation time through parallel execution of computationally intensive tasks. It is concluded that the automated approach to conducting simulations is preferable, and that it has proven successful for the purpose of this work.

Chapter 8

Appendix

A Glossary

Bandwidth Consumption: A metric expressing the blocking of a node involved in, or overhearing transmission or reception of a packet.

Bandwidth Overhead: A metric expressing the number of unnecessary forwards involved in delivery of a broadcast packet to all nodes in the MANET under consideration.

Bandwidth Utilisation: A metric expressing how many bytes per second are transmitted in an entire MANET.

Broadcast: The process of delivering a packet to every node within the MANET under consideration.

CBR Traffic source: Constant Bit-Rate traffic source.

Delivery: A reception of packet p on a node n, where n has not previously received p (i.e., a delivery occurs at most once per node).

Destination: The final target for a given packet. May be one, multiple or all nodes in the MANET.

Dominating Set: A set of vertices in a graph, such that every other vertex in the graph is adjacent to at least one vertex in the dominating set.

Efficiency: A metric expressing how many percent of the consumed bandwidth that result in packet deliveries.

Effectiveness: A metric weighing the efficiency and reliability metrics, with the goal of expressing the performance of a protocol as one single number.

Forward: The action of re-sending a received packet, with the intention of propagating the packet to other nodes.

Limited Broadcast: The process of delivering a packet to the immediate neighbours of the originator.

Link: The connection between two nodes in a MANET.

MANET: Mobile Ad-hoc Network.

MPR: Multipoint Relay.

MPR Flooding: Flooding of a packet using OLSRs MPR nodes for forwarding.

Message: Information unit exchanged as, e.g., control traffic between nodes running the OLSR protocol.

Multicast group: An abstraction of a set of nodes that participate in the same multicast session. A multicast group has a number of members, and is identified by a group address. All data sent to a multicast group is destined to the group address.

Multicast: Communication between groups of computers. Multicast packets are sent once, addressed to a group of nodes.

Neighbour: A node X is the neighbour of the node Y if Y is within the transmission range of node X.

Node: The encapsulation of a host and a router in a MANET.

OLSR node: A node running the Optimized Link State Routing protocol.

Originator: The node which originally created the first instance of a given packet.

Packet: The unit of data exchanged between the network and data link layer. A packet may contain a complete datagram, or a fragment thereof.

Proactive routing: Ongoing discovery and maintenance of routes to other MANET nodes.

Reactive routing: On-demand discovery of routes to other MANET nodes.

Receiver: A node accepting a packet. The receiver may be different form the destination node, i.e., an intermediate node on the path between originator and the destination.

Reliability: A metric expressing how many percent of all nodes in a MANET receives each byte (or packet) that is broadcast.

Sample: The simulations necessary to draw all the graphs for a test, but with only one simulation per sample point. I.e., one *simulation* for each scenario parameter combination.

Scenario: A model of a MANET, possibly generated from a set of *scenario parameters*, used for performing a simulating in, e.g., ns2.

Scenario Parameters: Values that define a pattern for, e.g., communication patterns, node mobility and scenario size. *Scenarios* may be generated automatically from scenario parameters.

Sender: The node which sends a packet. The sender may be a different than the originator, when packets are forwarded.

Simulation: A single execution of ns2, with a given scenario as input.

Test: 30 samples of a set of scenarios generated from common scenario parameters, using a common protocol.

Topology Control (TC) message: A control message type used by OLSR to communicate partial topology information among nodes.

Transmit: The action of sending a packet for the first time, conducted by the originator.

Two-hop neighbour: A node X is the two-hop neighbour of a node Y if X is the neighbour of one of X's neighbours, and X is not a one-hop neighbour of Y.

Two-hop neighbourhood: The set of all two-hop neighbours of a node.

Unicast: Transmission of packets from one single node to another.

B Selection of Dominating Sets

Figure B1 illustrates how selection of a certain MPR node ID to be used for Dominating Set Flooding, may result in delivery of packets via non-shortest paths.

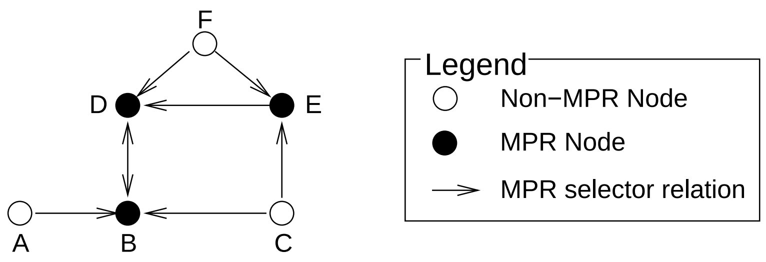

Figure B1: Example of non-shortest paths obtained via the dominating set of a MANET.

In the tree traversed by the node B's TC messages, only node D forwards the TC messages. Although being an MPR, node E does not forward them, as it has only recorded nodes F and C as its MPR selectors.

Consider the situation where node F broadcasts a packet, which is forwarded by the MPRs that has previously forwarded TC messages for node B. In this situation, the broadcast packet will traverse the path $F \rightarrow D \rightarrow B \rightarrow C$, rather than being delivered via the shortest path to node C, namely $F \rightarrow E \rightarrow C$.

C Simulation Platform Details

This appendix contains details on the simulation platform used for conducting the simulations presented in this work. Appendix C.1 describes the hardware platform, and section C.2 contains a calculating of the total execution time of all the simulations.

C.1 Hardware Platform

Cluster Machines		
sister1	2×i386 P3 733 MHz	2048 MB RAM
sister2	2×i386 P3 733 MHz	2048 MB RAM
sister3	2×i386 P3 733 MHz	2048 MB RAM
sister4	2×i386 P3 733 MHz	2048 MB RAM
sister5	2×i386 P3 733 MHz	2048 MB RAM
sister6	2×i386 P3 733 MHz	2048 MB RAM
sister7	2×i386 P3 733 MHz	2048 MB RAM
Application Servers		
atto	2×sparcv9 296 MHz	512 MB RAM
borg	4×sparcv9 450 MHz	4096 MB RAM
luke	2×sparcv9 450 MHz	2048 MB RAM
mega	2×i386 500 MHz	768 MB RAM
micro	2×sparcv9 296 MHz	512 MB RAM
obiwan	2×sparcv9 296 MHz	512 MB RAM
peta	2×sparcv9 450 MHz	2048 MB RAM
pico	2×i386 1024 MHz	1152 MB RAM
tera	2×sparcv9 296 MHz	1024 MB RAM
Shared-disk workstations:		
bird29	1×i386 P3 1088 MHz	256MB RAM
bird30	1×i386 P3 1088 MHz	256MB RAM
bird6	1×i386 P3 1088 MHz	256MB RAM
blade1	1×sparcv9 502 MHz	256MB RAM
blade2	1×sparcv9 502 MHz	256MB RAM
blade3	1×sparcv9 502 MHz	256MB RAM
Stand-alone workstations:		
tuborg	1×i386 P3 996 MHz	512MB RAM
carlsberg	1×i386 P3 863 MHz	256 MB RAM
sybaris	1×i386 P3 448 MHz	384 MB RAM
impression	1×i386 P3 863 MHz	878 MB RAM

Table C1: The machines present in the hardware platform used for the simulations conducted in this work.

C.2 Total Simulation Time

This appendix presents a simple calculation of the quantity of CPU-time spent for the simulations of this work.

The average simulation time of the simulations conducted in this work is approximately 30 minutes. With a total of 21.300 simulation, this amounts to 639.000 minutes, or 1.2 CPU-years of computation, on a CPU with average capacity of the 46 CPUs which have been used for this work.

Distributing the computations across 46 CPUs, the ideal computation time would be approximately 9.6 days could be achieved (assuming equal-capacity CPUs). In this work, the simulations were conducted during approximately 25 days rather than 9.6 days. The reason for a longer simulation time is that the machines were shared between multiple users, and due to this, on many occasions heavily loaded.

Bibliography

[AWF02] Khaled M. Alzoubi, Peng-Jun Wan, and Ophir Frieder. Message-Optimal Connected Dominating Sets in Mobile Ad Hoc Networks. In *Proceedings Of The Third ACM Internatinal Symposium on Mobile Ad Hoc Networking And Computing*, pages 157–164, June 2002.

[Bak95] F. Baker. Requirements for IP Version 4 Routers, June 1995. RFC 1812.

[Bet02] Christian Bettstetter. On the Minimum Node Degree and Connectivity of a Wireless Multihop Network. In *Proceedings Of The Third ACM Internatinal Symposium on Mobile Ad Hoc Networking And Computing*, pages 80–91, June 2002.

[Blu02] Bluetooth, 2002. Available as of May 21 2002, at `http://www.bluetooth.com/`.

[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 *Mobile Computing and Networking*, pages 85–97, 1998.

[BOT01] Bhargav Bellur, Richard G. Ogier, and Fred L. Templin. Topology Broadcast based on Reverse-Path Forwarding (TBRPF). Available as of June 14 2002, at `http://www.watersprings.org/pub/id/draft-ietf-manet-tbrpf-01.txt`, 2001. *Internet-draft,* Expired as of september 8, 2001.

[CH01] Lars Christensen and Gitte Hansen. The Optimized Link State Routing Protocol - Performance Analysis through Scenario-based Simulations. 2001. M.Sc. Thesis.

[CHCB01] Thomas Heide Clausen, Gitte Hansen, Lars Christensen, and Gerd Behrmann. The Optimized Link State Routing Protocol - Evaluation through Experiments and Simulation. 2001.

[CL02] Yuanzhu Peter Chen and Arthur L. Listman. Approximating Minimum Size Weakly-Connected Dominating Sets for Clustering Mobile Ad Hoc Networks. In *Proceedings Of The Third ACM Internatinal Symposium on Mobile Ad Hoc Networking And Computing*, pages 165–172, June 2002.

[CM99] S. Corson and J. Macker. Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations, January 199.

[DM78] Yogen K. Dalal and Robert M. Metcalfe. Reverse path forwarding of broadcast packets. *Communications of the ACM*, 21(12):1040–1048, 1978.

[DPR00] Samir R. Das, Charles E. Perkins, and Elizabeth M. Royer. Performance Comparison of Two On-demand Routing Protocols for Ad Hoc Networks. *Proceedings of INFOCOM 2000*, 2000.

[DRWT97] R. Dube, C. Rais, K. Wang, and S. Tripathi. Signal stability based adaptive routing (ssa) for ad-hoc mobile networks. 1997.

[glo02] GloMoSim. Available as of May 21 2002, at `http://pcl.cs.ucla.edu/projects/glomosim/`, May 2002.

[GPR01] Mobile GPRS, 2001. Available as of May 27, at `http://www.mobilegprs.com`.

[GSM02] GSM Association, 2002. Available as of May 21 2002, at `http://www.gsmworld.com`.

[JL99] Philippe Jacquet and Anis Laouiti. Analysis of Mobile Ad-hoc network routing protocols in random graph models. 1999.

[JLH$^+$99a] Per Johansson, Tony Larsson, Nicklas Hedman, Bartosz Mielczarek, and Mikael Degermark. Scenario-based Performance Analysis of Routing Protocols for Mobil Ad-hoc Networks. 1999. Appeared at Mobicom '99, Seattle, Washington, USA.

[JLH$^+$99b] Per Johansson, Tony Larsson, Nicklas Hedman, Bartosz Mielczarek, and Mikael Degermark. Scenario-based performance analysis of routing protocols for mobile ad-hoc networks. In *Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking*, pages 195–206. ACM Press, 1999.

[JMHJ02] David B. Johnson, David A. Maltz, Yih-Chun Hu, and Jorjeta G. Jetcheva. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks. Available as of June 14 2002, at `http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-07.txt`, February 2002. *Internet-draft,* Expires August 21, 2002.

[JMQ$^+$02] Philippe Jacquet, Paul Muhlethaler, Ami Qayyum, Anis Laouiti, Laurent Viennot, and Thomas H. Clausen. Optimized Link State Routing Protocol. Available as of June 14 2002 at `http://www.ietf.org/internet-drafts/draft-ietf-olsr-06.txt`, March 2002. *Internet-draft,* Expires September 1, 2002.

[Joh81] John Postel (editor). Internet Protocol, September 1981. RFC 791.

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

[JV00] Philippe Jacquet and Laurent Viennot. Overhead in Mobile Adhoc Network Protocols. June 2000.

[LO02] Nicolai Larsen and Tue Brems Olesen. Development of a MANET Multicast Routing Protocol and Simulation Environment. January 2002.

[man02] Manet IETF Working Group charter, 2002. Available as of May 21 2002, at `http://www.ietf.org/html.charters/manet-charter.html`.

[Mat02] Yukihiro Matsumoto. Ruby. Available as of June 6 2002, at `http://www.ruby-lang.org`, 2002.

[MD02] Roland McGrath and Ulrich Drepper. The GNU C Library. Available as of June 7 2002, at `http://www.gnu.org/directory/Libraries/C_libraries/glibc.html`, 2002.

[NTCS99] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu. The Broadcast Storm Problem in a Mobile Ad Hoc Network. 1999.

[opn02] OPNET Modeler. Available as of May 21 2002, at `http://www.opnet.com/products/modeler/home.html`, May 2002.

[PB94] Charles E. Perkins and Pravin Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. 1994.

[PBRD02] Charles E. Perkins, Elizabeth M. Belding-Royer, and Samir R. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. Available as of June 14 2002, at `http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-10.txt`, January 2002. *Internet-draft,* Expires Juli 9, 2002.

[pim02] Protocol Independent Multicast (pim) IETF Working Group Charter. Available as of June 14 2002, at `http://www.ietf.org/html.charters/pim-charter.html`, 2002.

[pro02] The VINT project. ns2 v2.1b8a, May 2002. Available as of May 21 2002, at `http://www.isi.edu/nsnam/ns`.

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

[qua02] QualNet by Scalable Network Technologies. Available as of June 14 2002, at `http://www.scalable-networks.com/`, 2002.

[QVL00] Amir Qayyum, Laurent Viennot, and Anis Laouiti. Multipoint relaying: An efficient technique for flooding in mobile wireless networks. March 2000.

[Rob02] Arnold Robbins. gawk - String manipulation language. Available as of June 2 2002, at `http://www.gnu.org/directory/gawk.html`, 2002.

[SK96] Guha Sudipto and Samir Khuller. Approximation Algorithms for Connected Dominating Sets, 1996.

[soc99] IEEE Computer society. ANSI/IEEE std. 802.11, 1999 edition. Published through the Get802.11 programme., August 1999.

[SRDG02] Supercluster Research and Development Group. Maui. Available as of May 22 2002, at `http://www.supercluster.org/maui`, May 2002.

[Ste94] W. Richard Stevens. *TCP/IP Illustrated, Volume 1: The Protocols.* Addison-Wesley Publishing Company, 1994.

[Wag92] Susan F. Wagner. *Introduction to Statistics.* HarperCollins Publishers Inc., 1992.

[WC02] Brad Williams and Trady Camp. Comparison of Broadcasting Techniques for Mobile Ad Hoc Networks. 2002. Presented at MOBIHOC'02, Lausanne, Switzerland.

[wpm02] The 5th international symposium on wireless personal media communications. Available as of June 14 2002, at `http://www.wpmc02.gatech.edu`, 2002.