Universidade do Minho
Escola de Engenharia

Jorge Carlos dos Santos Cardoso

Probabilistic Estimation of Network Size and Diameter

Tese de Mestrado
Mestrado em Sistemas Móveis

Trabalho efectuado sob a orientação do
Professor Doutor Carlos Miguel Ferraz Baquero Moreno

Novembro de 2008

DECLARAÇÃO

Nome

_____

Endereço electrónico: _____ Telefone: _____ / _____

Número do Bilhete de Identidade: _____

Título dissertação □/tese □

_____

_____

_____

Orientador(es):

_____

_____ Ano de conclusão: _____

Designação do Mestrado ou do Ramo de Conhecimento do Doutoramento:

_____

Universidade do Minho, ____/____/_____

Assinatura: _____

Acknowledgements

I would like to thank my advisor, Carlos Baquero, for his insight and dedication during this work.

I would also like to thank Paulo Sérgio Almeida for the initial idea on the diameter estimation technique, José Orlando Pereira for his help with the NeEM platform; Paulo Jesus for making his pre-PhD thesis available to me; and Nuno Carvalho for the precious help with the configuration of Modelnet and for the helpful shell scripts.

Abstract

Probabilistic Estimation of Network Size and Diameter

Determining the size of a network and its diameter are important functions in distributed systems, as there are a number of algorithms which rely on such parameters, or at least on estimates of those values. Although important, determining the size of a network, in a distributed manner, is not trivial. Algorithms that do this should be fast, to cope with high churn; fault-tolerant, to cope with link and node failures; and use a small number of messages, in order not to impose a high overhead in network bandwidth.

The Extrema Propagation technique presented in [Baquero et al., 2007] allows the estimation of the size of a network in a fast, distributed and fault tolerant manner. The technique was studied in a simulation setting where rounds advance synchronously and where there is no message loss.

This work presents two main contributions. The first, is the study of the Extrema Propagation technique under asynchronous rounds and integrated in the Network Friendly Epidemic Multicast (NeEM) framework. The second, is the evaluation of a diameter estimation technique associated with the Extrema Propagation. This study also presents a small enhancement to the Extrema Propagation in terms of communication cost and points out some other possible enhancements.

Results show that there is a clear trade-off between time and communication that must be considered when configuring the protocol—a faster convergence time implies a higher communication cost. Results also show that its possible to reduce the total communication cost by more the 18% using a simple approach. The diameter estimation technique is shown to have a relative error of less than 10% even when using a small sample.

The Extrema Propagation technique was partly integrated into the NeEM framework. Full integration would simply be a matter of feeding the estimated values back into the gossip and overlay layers which could use the estimated values for a better configuration of the fanout and time-to-live parameters.

Resumo

Estimação Probabilística do Tamanho e Diâmetro de Redes

A determinação do tamanho de uma rede e do seu diâmetro são funções importantes em sistemas distribuídos, uma vez que existem vários algoritmos que assentam nestes parâmetros, ou, pelo menos, numa estimativa desses valores. No entanto, embora importante, determinar o tamanho de uma rede, de uma forma distribuída, não é trivial. Os algoritmos devem ser rápidos, para lidarem com a entrada e saída de nodos; tolerantes a faltas, para lidarem com faltas nas ligações ou nos nodos; e usar o mínimo de mensagens, para não imporem uma sobrecarga na comunicação.

A técnica Extrema Propagation apresentada em [Baquero et al., 2007] permite a estimação do tamanho de uma rede, de forma rápida, completamente distribuída e tolerante a faltas. Esta técnica foi estudada por simulação de rondas síncronas e sem faltas.

Este trabalho apresenta duas contribuições principais. A primeira, é o estudo da técnica Extrema Propagation sob rondas assíncronas e integrada no sistema Network Friendly Epidemic Multicast (NeEM). A segunda, é a avaliação de uma técnica de estimação do diâmetro de uma rede associada com a técnica Extrema Propagation. Este estudo também apresenta um melhoramento à Extrema Propagation em termos de custo de comunicação e aponta outros possíveis melhoramentos.

Os resultados mostram que há um compromisso claro entre tempo e custo de comunicação que deve ser considerado ao configurar o protocolo—uma convergência mais rápida implica um maior custo de comunicação. Os resultados também mostram que é possível reduzir o custo total de comunicação em mais de 18% usando uma técnica simples. A avaliação da técnica de estimação do diâmetro mostra que o erro relativo se situa nos 10% mesmo utilizando uma amostra pequena.

A técnica Extrema Propagation foi parcialmente integrada no sistema NeEM. A integração completa implicaria apenas realimentar os valores estimados às camadas de difusão epidémica e de "overlay" que poderiam utilizar os valores estimados para melhor configurarem os parâmetros "fanout" e tempo-de-vida das mensagens.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Determining the size of a network is an important function in a distributed systems. There are a number of algorithms which rely on an estimate of the network size, or that would at least benefit from such an estimate; [Jesus, 2008] enumerates some of these algorithms:

- Distributed hash tables, for example, can take advantage of having an estimate of the network size to adjust the size of the routing table that each node keeps.
- Gossip-based protocols [Eugster et al., 2001, Ganesh et al., 2003] can use an estimate of the network size to better adjust the gossiping fanout parameter.
- In [Dolev et al., 2006], a group communication for ad-hoc networks based on random walks is presented. The algorithm uses an upper bound on the actual number of nodes. Better estimates of this upper bound would benefit the algorithm.
- The autonomous generation of node identifiers in mobile networks, based on a simple random number generation has been studied in [Jesus et al., 2006] and the knowledge of the size of the network is a fundamental parameter to define the length of the identifier or guaranteeing a predefined probability of collision.

Although important, determining the size of a network, in a distributed manner, is not trivial. Algorithms that do this should be fast, to cope with high churn; fault-tolerant, to cope with link and node failures; and use a small number of messages, in order not to impose a high overhead in network bandwidth.

Another important network parameter is its diameter—the maximum shortest-path length between any two pairs of nodes. Knowing the diameter, or at least having an estimate of its value is important to configure, for example, the time-to-live field on many protocols.

1.1 Motivation

The Extrema Propagation technique presented in [Baquero et al., 2007] allows the estimation of the size of a network. The technique is fast because it produces estimates in a number of steps close to the theoretical minimum; completely distributed, because every node determines the estimate by itself; does not require global identifiers; and tolerates message loss.

The Extrema Propagation technique has been studied in a simulation setting where rounds advance synchronously and where there is no message loss. However, in a real scenario, network nodes are often not synchronous, some nodes may fail, links have different latency and messages may be lost in transit.

One of the motivations for this study is thus to understand the behaviour of the Extrema Propagation under more realistic settings.

Also, the structure of the messages used by the technique suggests a new way to estimate the diameter of the network at the same time that its size.

The need to study and evaluate this technique to estimate the diameter is another motivation.

1.2 Contribution of This Work

This work presents two main contributions. The first, is the study of the Extrema Propagation technique under asynchronous rounds and integrated in the Network Friendly Epidemic Multicast (NeEM) framework. The second, is the evaluation of a diameter estimation technique associated with the Extrema Propagation.

This study also presents a small enhancement to the Extrema Propagation in terms of communication cost and points out some other possible enhancements.

1.3 Structure of This Document

This document is organized in the following way:

- Chapter 2 presents some work related to this study. It starts by presenting some graph theory concepts used throughout this document; it describes the NeEM framework; presents some other techniques for estimating the size of a network; and describes Modelnet—the simulation framework used in this study.

- Chapter 3 describes the Extrema Propagation technique with some detail. It also outlines a way to improve the communication cost of the technique.

- Chapter 4 presents the diameter estimation technique.
- Chapter 5 describes how the Extrema Propagation and diameter estimation techniques were adapted to NeEM.
- Chapter 6 describes the evaluation procedure and presents the results of the evaluation of the diameter estimation technique.
- Chapter 7 describes the evaluation procedure and presents the results of the evaluation of the Extrema Propagation technique integrated into NeEM.
- Chapter 8 concludes and presents some possible future work for enhancements.

Chapter 2

Related Work

2.1 Some Graph Theory Definitions and Concepts

The following are standard definitions for graph theory concepts used throughout this document.

Path A path is a sequence of vertices $x_1, x_2, ..., x_n$ such that

$(x_1, x_2), (x_2, x_3), ..., (x_{(n-1)}, x_n)$

are edges of the graph and the x_i are distinct. The path length is the number of graph edges $(x_1, x_2), (x_2, x_3), ..., (x_{(n-1)}, x_n)$ in the path.

Vertex Degree The degree of vertex v in graph G is the number of edges that connect v to other vertices in G.

Shortest Path The shortest path between two vertices in a graph is the path with the minimum length between the two vertices.

Vertex Eccentricity The eccentricity of a graph vertex v is the maximum shortest path length between v and all other vertices in the graph.

Graph Diameter The diameter of a graph G is the maximum eccentricity of all vertices in G.

Graph Radius The radius of a graph G is the minimum eccentricity of all vertices in G.

Graph Periphery The periphery of a graph is the set of vertices with eccentricities equal to the graph diameter.

Random Graph A graph—$G(n, M)$—chosen at random from the set of all graphs that have n vertices and M edges.[1]

[1] This corresponds to one variant of the Erdõs-Rényi [Erdös and Rényi, 1959] random graph model.

Preferential Attachment Graph A graph where vertices are connected preferentially to certain vertices. Vertices with higher degrees have a higher probability of being connected to vertices added to the graph. In this document, these graphs are modeled according to the Barabási-Albert [Barabási and Albert, 1999] model.

Random Geometric 2D Graph A graph that models the connectivity of points in a plane, positioned at random. A point is represented by a vertex and there is an edge between two vertices only if the distance between the corresponding points is smaller than a given radius.

2.2 NeEM

NeEM—Network Friendly Epidemic Multicast [Pereira et al., 2003]—is an epidemic multicast protocol which relies on connection-oriented transport connections (TCP/IP) in order to take advantage of the built-in end-to-end congestion control.

When gossiping, NeEM combines different strategies. If the message to be gossiped is small (smaller than a predefined value), then it is always pushed until its time-to-live (TTL) has expired. If the message is large, then it is pushed during $PushTTL$ rounds, and then *advertised* until TTL has expired. Both TTL and $PushTTL$ are preconfigured.

NeEM has been implemented in Java[2] and its main components are:

Multicast channel Provides applications with an interface for joining and leaving the network and for receiving and multicasting messages. In order to join the network, a peer needs to know, at least, another peer already in the network which will act as the entry point. A peer is identified by a randomly generated universally unique identifier (UUID).

Gossip layer Takes care of gossiping messages using the strategies described earlier. Keeps a cache of messages so that it can respond to pulling by peers. It also keeps a list of known message advertisers in order to pull advertised messages. Messages are identified by a randomly generated UUID. When gossiping, it randomly selects $Fanout$ peers from the partial membership kept by the overlay layer.

Overlay layer Manages the overlay network and a partial membership list. The overlay tries to maintain a fixed number of peers in the local membership list (in NeEM's implementation this is also called $Fanout$ but is different from the gossip $Fanout$). Membership is dynamic as the overlay will shuffle peers periodically, by randomly selecting two

[2]See the project's sourceforge web page: `http://neem.sourceforge.net/index.html`.

peers from the current list and informing them of one another. Those peers will then randomly decide if they add the other one to the current list, eventually purging another.

Transport Layer The transport layer manages TCP connections between peers, maintaining queues of messages per connection.

NeEM's structure can be seen as layer stack as depicted in Figure 2.1.

Application
Multicast Channel
Gossip
Overlay
Transport

Figure 2.1: NeEM's layer stack.

NeEM is implemented in a way that allows easy integration of other protocols. The transport layer allows different handlers to be registered and will demultiplex messages based on a logical port and deliver them to the appropriate handler.

2.3 Network Size Estimation Protocols

There are numerous algorithms for estimating the size of a network or, more generally, for performing data aggregation across a network. In [Jesus, 2007] a chronological overview of aggregation protocols is presented and algorithms are classified according to a proposed taxonomy that considers two perspectives: communication and computation. Some algorithms, more directly related to this study are described next.

[Psaltoulis et al., 2004] presents two methods for estimating the size of a network: the Hops Sampling and the Interval Density methods.

In the Hops Sampling method, an initiator node starts the process by sending a message to *gossipTo* nodes. The message contains a hop count that is incremented prior to being sent. Messages sent from a node p are only sent to nodes from which p has not received any message. After *gossipResults* rounds have elapsed the initiator collects a sample of hop counts from *gossipSample* other nodes and uses the average of the hop counts as an estimate of $log(N)$. This methods requires that each node is able to maintain a membership list chosen uniformly at random from the system. Also, only the initiator node will have an estimate of the size.

In the Interval Density method, the process identifiers are hashed and mapped to a point in the interval $[0, 1]$. The initiator node then calculates the number of processes, X, in an interval $I < 1$. The estimate of the network size is then calculated as X/I. In order to collect the samples

(identifiers) the authors propose piggybacking messages from a membership maintenance protocol with information about the interval used in the estimates. In this method, the difficulty is in defining the interval I, since it depends on the network size N.

[Kempe et al., 2003] uses a different approach for computing sums and averages. Each node keeps two variables: s and w initialized to x_i—the contribution of node i to the sum—and 1, respectively. In each round, nodes select another node at random and send $\frac{s}{2}$ and $\frac{w}{2}$ to that node (and keep $\frac{s}{2}$ and $\frac{w}{2}$ to themselves). Each pair of s and w received by a node is added to the current ones. In each round, the estimate of the average is given locally by $\frac{s}{w}$. After a number of rounds all nodes achieve the same result—the global average. To calculate the sum, the only difference is that only one node starts with $w = 1$ and all others start with $w = 0$ (to calculate the size of the network, $s = 1$ in all nodes).

[Jelasity et al., 2005] introduced a similar approach but in this case information is exchanged in a symmetric manner. A node A randomly selects another node B to exchange its value v_a and sends him a Push message with v_a. Upon receipt of a Push message, node B will send a Pull message with its own value v_b to the node A. After receiving a Push or a Pull message both nodes will update their value v:

$$v = \frac{v_a + v_b}{2}$$

This technique (as well as the one by [Kempe et al., 2003]) is sensitive to message loss. Both algorithms require that the total system "mass" is kept constant. A message loss will break this requirement.

[Jesus, 2007] proposes a different algorithm—Flow Updating, also based on averaging, but which is immune to message loss. In this case, instead of exchanging the original value v_0, nodes exchange "flows". Locally, the estimate of the aggregate value is given by the sum of the initial value with all the flows:

$$\hat{v} = v_0 + \sum_{i=1}^{n} \mathcal{F}_i \tag{2.1}$$

The basis of the algorithm (for two nodes only) is the following. Node A sends his estimate e_a to node B. Node B calculates his new estimate as $e'_b = \frac{e_a+e_b}{2}$ and flow as $e'_b - e_b$ and sends the symmetric of the calculated flow to node A. Node A updates his flow with that sent by node B. In the end node A will calculate his estimate as (using equation 2.1):

$$e_a - (\frac{e_a + e_b}{2} - e_b) = \frac{e_a + e_b}{2}$$

and node B as:

$$e_b + (\frac{e_a + e_b}{2} - e_b) = \frac{e_a + e_b}{2}$$

So, in the end, both nodes have the same value—the global average. The procedure can be adapted for exchange of values among several nodes simultaneously. Also, the loss of messages is naturally tolerated causing only a delay in achieving the final estimate.

2.4 Simulation of Wide-area Networks

There are many network simulation and emulation tools such as NS-2[3], Dummynet[4], NIST Net[5] or PlanetLab[6]. However they all either require special purpose scripts to be written as is the case of the network simulator NS-2; do not easily support emulation of several instances of a program in a single machine as is the case of Dummynet and NIST Net; or require a registration and submission process that is too strict for this kind of study as is the case of PlanetLab [Bavier et al., 2004].

Because of these limitations, Modelnet [Vahdat et al., 2002] was chosen as the emulation platform for this study.

2.4.1 Modelnet

Modelnet is a large-scale network emulator that can be used with a cluster of machines in a laboratory to simulate a wide-area network.

> "Modelnet is designed to run on a machine room cluster to evaluate wide area distriubted [*sic*] systems. One or more of the cluster machines is set aside for traffic emulation, while the remainder can be used to operate as nodes in the application. When these node [*sic*] communicate with each other, those IP packets are sent through the emulators to create the illusion that application packets are crossing the wide area Internet."—[Becker and Yocum, 2003].

To use Modelnet it is necessary to create a virtual network topology that will be used by the traffic emulator machine.

Overview

Modelnet uses two kinds of machines: traffic emulators and application hosts. Traffic emulators (emulators for short) work as routers and apply the defined bandwidth, delay and drop rate parameters defined in the network topology. Application host machines run the application to test; a single application host may host several *virtual nodes*.

[3] `http://www.isi.edu/nsnam/ns/`
[4] `http://info.iet.unipi.it/~luigi/ip_dummynet/`
[5] `http://snad.ncsl.nist.gov/nistnet/`
[6] `http://www.planet-lab.org/`

Modelnet works by assigning multiple IP addresses (in the `10.0.0.0/8` subnet) to the same application host—each IP address is assigned to a virtual node in that machine. The application host's routing table is modified to route traffic sent from this subnet to the emulator machine which will apply the network settings defined in the network topology and route the packet to its destination. Even if the source and destination virtual nodes reside in the same machine, routing will always go through the traffic emulator.

The emulator machine runs a custom kernel module that performs the traffic emulation.

The host application machines need a special shared library—`libipaddr`—that applications must load and which changes some networking functions such as `bind()` and `sendto()`. This allows Modelnet to change the source and destination IP of each packet so that it will be correctly forwarded to the emulator.

Modelnet requires FreeBSD as the operating system for the emulator machine and FreeBSD or Linux for the host application machine.

Routing

Routing within Modelnet works as follows. In this example, a given virtual node A was assigned IP `10.0.0.1` and it wants to communicate with virtual node B with IP `10.0.0.2`. Both virtual nodes reside in the same machine. When starting the application, the modified shared library will define the source address of node A as `10.0.0.1`. When sending, node A will put `10.0.0.2` in the destination address, but the shared library will turn on bit 23 of that address, changing it to to `10.128.0.2`. This is done so that the packet can be routed to the emulator. Otherwise, it would go through the loopback interface since `10.0.0.2` is on the same machine. The application host's routing table defines that all packets to `10.128.*` go to the emulator machine. When the packet reaches the emulator, it is processed (queuing, bandwidth, delays and drop rates are applied) and the destination address is changed back to `10.0.0.2`. The emulator's routing table defines that this packet will to the application host. When the packet arrives at the application host it is delivered to the target application.

One of the advantages of Modelnet is that most applications can be tested without being modified. The shared library can be preloaded when launching the application and the routing mechanism described will be applied transparently to the application.

Network Topology

A topology must be created so that Modelnet knows how to route packets and apply network parameters. Although the topology can be created by any tool (Modelnet uses an XML file format), Modelnet directly supports the

Inet tool [Jin et al., 2000]. The Inet2xml Modelnet tool uses data from Inet to generate a graph of the network topology. Modelnet can assign default bandwidth, latency and drop rate, or a range of values can be assigned to these parameters. Latency can also be inferred from the link length in the Inet output file.

Using this graph file, Modelnet creates a route file with the shortest path for all pairs of virtual nodes.

Modelnet also needs a file with machine names to use in the emulation and what kind (emulator or host) of machine they are.

Using the machines and graph files, Modelnet assigns virtual nodes to hosts and links to emulators. This is saved in a model file.

Deployment and Running

Modelnet has a number of shell scripts that automate most the deployment and running process. Deployment is a matter of installing the kernel module in the emulators and creating the correct routing table based on the model and route files. In the hosts machines it is necessary to create the routing table. This can be done by running the following script in each machine:

```
deployhost example.model example.route
```

Deployment across several machines can be automated with the `deploy` script, if `gexec`[7] is installed:

```
deploy example.model example.route
```

For starting the emulation, the application to test must be launched and the `libipaddr` must be preloaded. This can be done with the `vnrun` script:

```
vnrun all example.model app
```

This script will execute a script in one or all virtual nodes and it also takes care of the `libipaddr` library preloading.

[7] A cluster remote execution tool. See `http://www.theether.org/gexec/`.

Chapter 3

Extrema Propagation Technique

The Extrema Propagation technique is a probabilistic data aggregation technique which works as follows.

> "[...] if we generate a random real number in each node using a known probability distribution (e.g. Gaussian or exponential), and aggregate across all nodes using the minimum function, the resulting value has a new distribution which depends on the number of nodes. The basic idea is then to generate a vector of random numbers at each node, aggregate each component across the network using the pointwise minimum, and then use the resulting vector as a sample from which to infer the number of nodes (by a maximum likelihood estimator)."— [Baquero et al., 2007]

Basically, each node generates a vector of K exponentially distributed random numbers (`Minimums` vector in the rest of this document) and sends it to its neighbours. Each node aggregates and resends vectors from neighbours using the pointwise minimum—an idempotent operation. After convergence (determined by each node, after a predefined number of rounds have passed without changing the minimums vector) the resulting vector—with the K global minimums—can be used to estimate the number of nodes in the network.

The technique's focus is on speed not on accuracy, since aiming at very low errors would need very large vectors. For example, an error of 1% would require

$$K = 2 + \left(\frac{1.96}{error}\right)^2 = 2 + \left(\frac{1.96}{0.01}\right)^2 = 38418.$$

A 10% error, however, only requires $K = 387$.

In order to optimize the message's size, the technique encodes the values using only the exponent of it's (mantissa, exponent) representation. Furthermore, only 5 bit are used to represent the exponent. This means that a vector with $K = 387$ needs only

$$5 \times 387\ bit = 242\ byte.$$

Convergence is determined by a predefined number of rounds without changes (no news rounds—T) occurring in the `Minimums` vector. The minimum number of no news rounds to wait depends on the network topology and size, but a safe value can be used to accommodate, at least, a wide range of expected network sizes.

Algorithm 1 presents this technique in algorithmic form (adapted from [Baquero et al., 2007]).

The technique was studied by simulating runs (with synchronous rounds) for several values of K and for different types of networks (geometric 2D, random and preferential attachment networks) and sizes (100, 1000 and 10000 nodes). For each run, the average and maximum number of rounds needed to converge where recorded, as well as the maximum number of rounds with no news.

Results show that, for random and preferential attachment networks, $T = 5$ is sufficient when $K = 10$ or $K = 100$ and $T = 4$ is sufficient when $K = 1000$. For geometric 2D networks, K should be 100 or 1000 since $K = 10$ would lead to a large overhead of the no news rounds over the average number of rounds needed to converge. For $K = 100$ a $T = 19$ would suffice and for $K = 1000$, $T = 11$. These are all conservative values to ensure that all nodes have converged.

3.1 Slow Links and Message Loss

The Extrema Propagation technique is resilient to message failures: if a message (with vector x) from node A to node B is lost, in the next round, it will be superseded by another message with $x' \leq x$.

The algorithm presented, though, does not cope with message loss very well as it waits for every neighbour, forever. A message loss will deadlock the system.

As suggested in [Baquero et al., 2007], the algorithm can be modified to, instead of waiting for every neighbour, forever:

- Wait for all neighbours, until a timeout occurs.
- Wait for all neighbours minus F.
- Wait for all neighbours, until a timeout and then wait for all minus F.

For sake of completeness, there is one more variant that can be used:

Algorithm 1 Extrema Propagation

```
const K, T
var Neighbours
var Minimums[1..K]
var converged, nonews
procedure INIT                                  ▷ Initialize the protocol
    Neighbours ← neighbours(self)
    nonews ← 0
    converged ← False
    for i ∈ 1..K do
        Minimums[i] ← rExp(1)
    end for
    Send Minimums to every n ∈ Neighbours
end procedure
procedure RECEIVEFROMALL(m[1..K])  ▷ Received from all neigbours
    var oldm ← Minimums
    for i ∈ 1..K do
        if m[i] < Minimums[i] then
            Minimums[i] ← m[i]
        end if
    end for
    if oldm ≠ Minimums then
        nonews ← 0
    else
        nonews ← nonews + 1
    end if
    if nonews ≥ T then
        converged ← True
    end if
    Send Minimums to every n ∈ Neighbours
end procedure
function QUERY                  ▷ Return estimation of N, if converged
    if converged = True then
        return N̂(x)
    else
        return False
    end if
end function
```

- Wait for all neighbours minus F, until a timeout occurs.

These variants, besides allowing to cope with message failures, make the algorithm more robust in face of slow links. In the original algorithm, although not fatal, a single slow link would slow down the entire system. By introducing timeouts and the possibility of not waiting for every neighbour, a slow link can be regarded as a message failure. This means that the algorithm may not wait for nodes behind slow links, thus not slowing the entire round. Messages sent on slow links will arrive and be accounted for in the following rounds.

Proper configuration of the timeout and F is one of the purposes of this study.

3.2 Improvement of the Message Format

The standard message in the Extrema Propagation is a message with K values, 5 bit each, corresponding to the minimums vector that each node stores. This vector is updated when one (or several) new minimum is received, but is sent in every round even if it is equal to the last vector sent. Also, the same vector is sent to every neighbour, regardless of what that neighbour already "knows".

This means that potentially old information is sent in each message. A simple way to mitigate this inefficiency is to store the last vector that each neighbour sent and use it as a way to determine what needs to be sent to that neighbour in the next round. This way, messages are differentiated by neighbour. A message only needs to contain values that are smaller than the ones in the current copy of that neighbours' vector.

Instead of sending $K * 5$ bits, each node can send only N pairs of $(index, value)$ entries—the number of values smaller than the ones in the current copy of that neighbour's vector. In order to correctly interpret the message, the receiving node needs to know also how many pairs were sent. This message will only save bandwidth if the number of pairs $(index, value)$ is sufficiently small. If not, the standard format should be used instead.

Optimizing the message size in this way means trading off bandwidth usage for memory usage since each node now has to keep the last vectors received from each neighbour.

The final message structure uses one of two types:

Type 0	1 bit
Number of pairs (N)	$ceil(log_2(K))$ bit
(Index, Value) pairs	$(ceil(log_2(K)) + 5) * N$ bit

Type 1	1 bit
Minimums vector	$K * 5$ bit

K	N
10	5
100	41
1000	332

Table 3.1: Maximum values of N that compensate sending (*index*, *value*) pairs.

In order for Type 0 to be used it must save bandwidth, which means that:

$$ceil(log_2(K)) + (ceil(log_2(K)) + 5) * N \leq K * 5 \tag{3.1}$$

This results in:

$$N \leq \frac{K * 5 - ceil(log_2(K))}{ceil(log_2(K)) + 5} \tag{3.2}$$

Table 3.1 shows, for different K, the maximum values of N that compensate the usage of Type 0 message.

In the initial phase of the protocol the number of new values will probably not compensate using the new message format, but as the protocol runs and the vector stabilizes in each node, less information will be communicated. In the final runs, when every node has converged to the final vector, only $1 + ceil(log_2(K))$ bit will be sent. The overall savings in network bandwidth usage should compensate the introduction of a 1 bit overhead in the original message size.

Chapter 4

Diameter Estimation Technique

The proposed diameter estimation technique works as follows. For each entry in the `Minimums` vector of the Extrema Propagation technique, there is a corresponding entry in a new vector `Hops`. The `Hops` vector is initialized at every node with zeroes. Every time that a node updates the `Minimums` vector because it has received a smaller value from one of its neighbours, it also updates the corresponding entry in the `Hops` vector with the value received from that neighbour. The `Hops` vector is also updated when a node receives a smaller entry in the `Hops` vector for an equal value in the `Minimums` vector. A copy of the `Hops` vector, with values increased by 1, is sent along with the `Minimums` vector, to every neighbour, in each round.

A particular value in the `Hops` vector of a node may only decrease during the convergence phase of the `Minimums` vector. This means that the maximum of the `Hops` vector also decreases, which means that the maximum can only be aggregated after the `Hops` vector has stabilized (otherwise, the final maximum might be greater than the true final maximum). There is no guarantee that the `Hops` vector becomes stable at the same time that the `Minimums` vector has, so a different no news counter must be used.

Algorithm 2 lists the pseudo-code for this algorithm.

Algorithm 2 Extrema Propagation and Diameter Estimation Algorithm

```
const K, T, TH
var Neighbours
var Minimums[1..K], hops[1..K]
var converged, nonews, nonewsHops
procedure INIT                                        ▷ Initialize the protocol
    Neighbours ← neighbours(self)
    nonews ← 0, nonewsHops ← 0
    converged ← False
    for i ∈ 1..K do
        Minimums[i] ← rExp(1), hops[i] ← 0
    end for
    Send (Minimums, hops + 1) to every n ∈ Neighbours
end procedure
procedure RECEIVEFROMALL(m[1..K], h[1..K])            ▷ Received from all
    var oldm ← Minimums
    var oldh ← Hops
    for i ∈ 1..K do
        if m[i] < Minimums[i] then
            Minimums[i] ← m[i], hops[i] ← h[i]
        else if Minimums[i] = m[i] ∧ h[i] < hops[i] then
            hops[i] ← h[i]
        end if
    end for
    if oldm ≠ Minimums then
        nonews ← 0
    else
        nonews ← nonews + 1
    end if
    if oldh ≠ Hops then
        nonewsHops ← 0
    else
        nonewsHops ← nonewsHops + 1
    end if
    if nonews ≥ T then
        converged ← True
    end if
    if nonewsHops ≥ TH then
        AggregateMaximum(max(hops))
    end if
    Send (Minimums, hops + 1) to every n ∈ Neighbours
end procedure
```

In this algorithm, we assume the existence of an `AggregateMaximum()` primitive that aggregates the maximum value across all neighbours.

The classical way of finding the network diameter is by having all nodes conduct a breadth-first search in parallel and using the constructed tree to compute the maximum distance, which could then be aggregated across the network to discover the network diameter [Lynch, 1997]. However, this produces a lot of messages that grow in size with the number of nodes and also requires that nodes possess a unique ID.

4.1 Analysis of the Diameter Estimation Technique

For each of the K global minimums, each node will determine the shortest path to the closest node which generated the minimum (a particular global minimum, can be generated by more than one node).

This technique results in the determination of the eccentricity of the nodes which generated unique minimums, i.e., global minimum values generated by only one node in a particular vector position, after convergence and aggregation of the point-wise maximum $Hops$ vector. For non-unique minimums the result may be a value smaller than the eccentricity because some nodes will determine the shortest path to one of the nodes that generated that particular minimum, and others will determine the shortest paths to another node that generated the minimum. This is illustrated in Figure 4.1, for only one minimum ($K = 1$). It is easy to see that when more than one node generates the same global minimum (right graph, coloured nodes), the resulting eccentricity—the maximum of the shortest paths—is not correct. Some nodes calculate the shortest path to node B and others to node D.

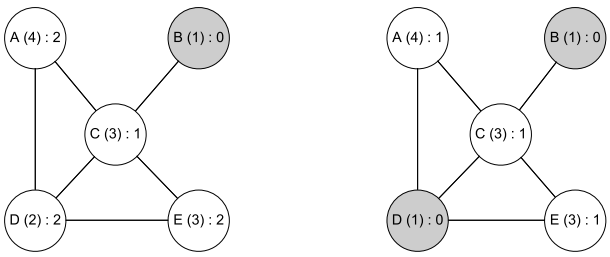

Figure 4.1: Effect of non-unique minimums in the diameter estimation technique. Value in parentheses is the generated minimum, value after colon is the shortest path to the global minimum. Coloured nodes generated the global minimums. Left: unique minimum correctly results in determining the eccentricity. Right: more than one node generating the minimum results in wrong values for eccentricity.

If the following two conditions are met, the technique is guaranteed to

determine the correct diameter of the network:

1. Enough rounds were executed to allow the propagation of the hops counter from the nodes that generated the global minimums to all other nodes and,

2. At least one unique minimum was generated in a peripheral node.

If the first condition is true, than values will always be bounded by:

$$NetworkRadius \leq EstimatedDiameter \leq NetworkDiameter \quad (4.1)$$

Since, by definition, no eccentricity can be smaller that the network radius or larger than the network diameter.

The lower bound for the number of steps in this estimation technique is $2 \times D$, where D is the network diameter. This can the shown informally: If nodes a and b are such that the shortest path between them is D and a is the generator of a minimum, than for b to become aware of a's minimum and consequently its distance to b, it takes at least D exchange steps. For a to become aware of distance calculated by b it takes at least another D exchange steps.

If some previous knowledge about the network exists, one can estimate the accuracy of this method. For this, we need also to know how many unique minimums, from a set of K, are generated by the network, on average.

Specifically, if the probability of a node being in the periphery is known to be p then N—a random variable which denotes the number of nodes in the periphery which generated the final minimums—follows the Binomial distribution:

$$N \sim B(UniqueMinimums, p) \quad (4.2)$$

The probability that at least one of the nodes that generated the unique minimums is in the periphery is:

$$P(N \geq 1) = 1 - P(N = 0) \quad (4.3)$$

Some worst case scenarios can be calculated to get some insight on the expected accuracy of this method. In the worst case, there are only two nodes in the periphery of the network (one, if the graph is directed) and p becomes:

$$p_{worstcase} = \frac{2}{NetworkSize} \quad (4.4)$$

Table 4.1 lists some combinations of $p_{worstcase}$ and K along with the corresponding probabilities of a periphery node generating a global minimum, i.e., the probability of the diameter technique estimating the real diameter.

In this table we use $0.7 \times K$ as $UniqueMinimums$. This value was computed by simulation and is explained in Chapter 6.

For small networks ($\leq$ 500 nodes) the diameter estimation achieves a high probability (greater than 90%) of estimating the real diameter if $K = 1000$, even in the worst case scenario.

It should be noted, however, that these are worst case scenario calculations. Depending on network topology, the probability of a node being in the periphery may be considerably larger. Also, these are probabilities for the case where the technique estimates the real value, i.e., $error = 0$. The study of the error is presented in Chapter 6.

Network size	$p_{worstcase}$	K	$P(N \geq 1)$
10000	$\frac{2}{10000}$	10	0.14%
		100	1.39%
		1000	13.07%
5000	$\frac{2}{5000}$	10	0.28%
		100	2.76%
		1000	24.43%
1000	$\frac{2}{1000}$	10	01.39%
		100	13.08%
		1000	75.37%
500	$\frac{2}{500}$	10	2.77%
		100	24.46%
		1000	93.95%
100	$\frac{2}{100}$	10	13.19%
		100	75.69%
		1000	100.00%
50	$\frac{2}{50}$	10	24.86%
		100	94.26%
		1000	100.00%

Table 4.1: Probability of a periphery node generating a global minimum.

Chapter 5

Integration of Extrema Propagation in NeEM

The Extrema Propagation and the diameter estimation techniques were implemented in the NeEM framework in order to study the effect of asynchronous rounds and the modifications presented in Section 3.1 to cope with message failure and slow links.

5.1 Overview

Both the Extrema Propagation and the diameter estimation techniques were implemented to the extent necessary to perform the study, which means that the implementation only aggregates the `Minimums` vector (it does not really perform the estimation) and that convergence is not determined (only after analyzing the logs it can be determined). Since convergence is not determined, the maximum value of the `Hops` vector is not aggregated, hence the diameter can only be estimated by looking at the logs. This also means that the NeEM framework is not fed back by the estimated network size and diameter, but implementing this would be a simple step.

Extrema was implemented as a separate layer in NeEM's layer stack, at the same level as the Gossip layer, since Extrema only needs the Overlay layer functionality.

NeEM's adapted layer stack becomes the one depicted in Figure 5.1.

<table>
<tr><td colspan="2">Application</td></tr>
<tr><td colspan="2">Multicast Channel</td></tr>
<tr><td>Gossip</td><td>Extrema</td></tr>
<tr><td colspan="2">Overlay</td></tr>
<tr><td colspan="2">Transport</td></tr>
</table>

Figure 5.1: NeEM's layer stack after Extrema implementation.

5.2 Adapting Extrema

The different strategies for deciding when to advance a round, outlined in Section 3.1, where all implemented in order to be compared.

These strategies differ on how the decision to advance a round, i.e., to send the current `Minimums` vector to the node's neighbours, is made: based on how many message we have received in this round; based on expiration of a timeout; based on a combination of the former two.

As described earlier, NeEM maintains a dynamic partial membership list which is shuffled periodically and independently from the Extrema layer, but tries to maintain $Fanout$ peers in that list, so $Fanout$ can be used as the expected number of neighbours for each node.

The decision can be expressed, algorithmically, by the following procedure variations.

In the ONLY_TIMEOUT strategy, decision to advance is based solely on the expiration of a timer (of course the round can also advance if all neighbours have already sent their messages):

```
procedure DecideAdvance                          ▷ Only Timeout Strategy
    if whoSent.size() = Fanout then
        call Advance()
    end if
    if timedOut then
        call Advance()
    end if
end procedure
```

In the ONLY_F strategy, rounds advance as soon as all neighbours minus F have sent their messages:

```
procedure DecideAdvance                          ▷ Only F Strategy
    if whoSent.size() ≥ Fanout − F then
        call Advance()
    end if
end procedure
```

In the TIMEOUT_PLUS_F strategy, nodes wait for every neighbour until a timeout occurs. After the timeout, nodes wait for all neighbours minus F:

```
procedure DecideAdvance                          ▷ Timeout + F Strategy
    if whoSent.size() = Fanout then
        call Advance()
```

```
    end if
    if timedOut then
        if whoSent.size() ≥ Fanout − F then
            call Advance()
        end if
    end if
end procedure
```

In the F_PLUS_TIMEOUT strategy, nodes wait for all neighbours minus F, until a timeout occurs:

```
procedure DecideAdvance                    ▷ F + Timeout Strategy
    if whoSent.size() ≥ Fanout − F then
        call Advance()
    end if
    if timedOut then
        call Advance()
    end if
end procedure
```

The adapted Extrema algorithm is presented in Algorithm 3. For clarity, the code dealing with the determination of no news rounds and convergence was eliminated. Also for clarity, the diameter estimation part is not included.

It is more reasonable for the protocol to be started by one node alone, instead of all nodes at the same time. As a consequence, the `Init()` procedure in this algorithm is invoked in response to a `Receive()` event, if the node's `Init()` has not been invoked before. The node starting the process must have its `Init()` procedure invoked by some other means.

Algorithm 3 adds a *round number* to the message. This round number can be used by nodes to determine if they are receiving an old message from a slow link, or a recent message.

Algorithm 3 Neem Extrema Propagation

```
const K
var timedOut ← False
var inited ← False
var roundNumber
var Minimums[1..K]
var whoSent                                          ▷ Hashtable
procedure INIT
    for i ∈ 1..K do
        Minimums[i] ← rExp(1)
    end for
    roundNumber ← 0
    call Advance()
    inited ← True
end procedure
event RECEIVE (nodeID, roundNumber, m[1..K])
    if not inited then
        call Init()
    end if
    whoSent.add(nodeID, (roundNumber, m))
    call Integrate(m)
    call DecideAdvance()
end event
event TIMEOUT
    timedOut ← True
    call DecideAdvance()
end event
procedure ADVANCE
    Send Minimums to every n ∈ neighbours(self)
    roundNumber ← roundNumber + 1
    call ResetTimer()                ▷ Resets timer and clears timeout.
    whoSent.clear()
end procedure
procedure INTEGRATE(m[1..K])
    for i ∈ 1..K do
        if m[i] < Minimums[i] then
            Minimums[i] ← m[i]
        end if
    end for
end procedure
```

Chapter 6

Evaluation of the Diameter Estimation Technique

In order to evaluate the accuracy of the proposed diameter estimation technique several studies were performed.

First, the accuracy of the method was evaluated in face of different network sizes and topologies and in face of different values for K. Then, other distributions for the generation of minimums were tested in order to determine if the number of unique minimums could be improved. Finally, the trade-off between the vector length (K) and the value encoding size was analyzed.

6.1 Accuracy

6.1.1 Procedure

In order to estimate the expected accuracy of the diameter estimation technique, the method was applied to three different network topologies (random networks, preferential attachment networks and random geometric 2D networks) and sizes (50, 500 and 5000 nodes networks).

For each topology and size, 150 graphs were generated. For random and preferential attachment graphs, average degrees of 5, 10 and 15 were used. For each, 50 graphs were created. For 2D graphs, 1.3, 1.6 and 1.9 relative radius were used and 50 graphs were created for each.

Graphs were generated using a Python script. For random and preferential attachment graphs the NetworkX [Hagberg et al., 2008] Python package was used, specifically the `gnm_random_graph()` and `barabasi_albert_graph()` functions, respectively. For generating random geometric 2D graphs, Algorithm 4 was used.

For each graph, three different values of K were simulated, 50 times each.

Algorithm 4 Random Geometric 2D Graph Generation

```
function GRAPH_2D(Size, Radius)
    var Graph
    for i ∈ 1..Size do
        Graph.add_node(rand.uniform(0,1), rand.uniform(0,1))
    end for
    dist ← Radius/math.sqrt(Size)
    for n1 ∈ 1..Size do
        for n2 ∈ n1..Size do
            if Graph.distance(n1, n2) < dist then
                Graph.add_edge(n1, n2)
            end if
        end for
    end for
    while not Graph.is_connected() do                ▷ Connect graph
        Graph.connect_biggest_component_to_closest_component
    end while
    Return Graph
end function
```

For a graph of size N, the simulation consisted of:

1. Generating K random exponentially distributed values coded according to [Baquero et al., 2007] for each node in the graph;
2. Calculating the K global minimums across the graph;
3. Determining the nodes that generated each of the global minimums;
4. For each global minimum, calculating the shortest paths from all nodes to the nodes that generated the minimum and taking the minimum shortest path length;
5. Calculating the estimated diameter as the maximum of the previous values across all nodes.

For each graph, the diameter, radius and the eccentricities of all nodes were also calculated and recorded. This allowed to calculate the error in the diameter estimation and percentage of nodes in the periphery of the graph.

6.1.2 Results

Table 6.1 summarizes the result of the simulation.

From the absolute error perspective, random geometric 2D networks are clearly the worst type of network for the diameter estimation technique. In these networks, the number of nodes in the periphery is almost always

Network Type	Network Size	Average Diameter	Average Absolute Error	**Average Relative Error**
		$K = 10$		
2d	50	9.9	0.5997	**6.03%**
Attach	50	3.0	0.1116	**3.68%**
Random	50	2.4	0.0340	**1.05%**
2d	500	35.1	3.2631	**9.43%**
Attach	500	3.7	0.3396	**8.14%**
Random	500	4.0	0.4376	**10.10%**
		$K = 100$		
2d	50	9.9	0.0117	**0.12%**
Attach	50	3.0	0.0001	**0%**
Random	50	2.4	0.0017	**0.06%**
2d	500	35.1	0.6669	**2.00%**
Attach	500	3.7	0.1588	**3.73%**
Random	500	4.0	0.1081	**2.65%**
		$K = 1000$		
2d	50	9.9	0	**0%**
Attach	50	3.0	0	**0%**
Random	50	2.4	0	**0%**
2d	500	35.1	0.0116	**0.03%**
Attach	500	3.7	0.0048	**0.11%**
Random	500	4.0	0.0019	**0.05%**

Table 6.1: Diameter estimation accuracy

very low, as show in Figure 6.1 (top-left). This leads to the highest average absolute error, from all types of networks. However, since geometric 2D networks also have the highest diameter, the relative error ends up being in the same order of magnitude (or very close) for all network types.

Preferential attachment networks, having a very low diameter and a distribution of peripheral nodes as depicted in Figure 6.1 (top-right), have the highest average relative error.

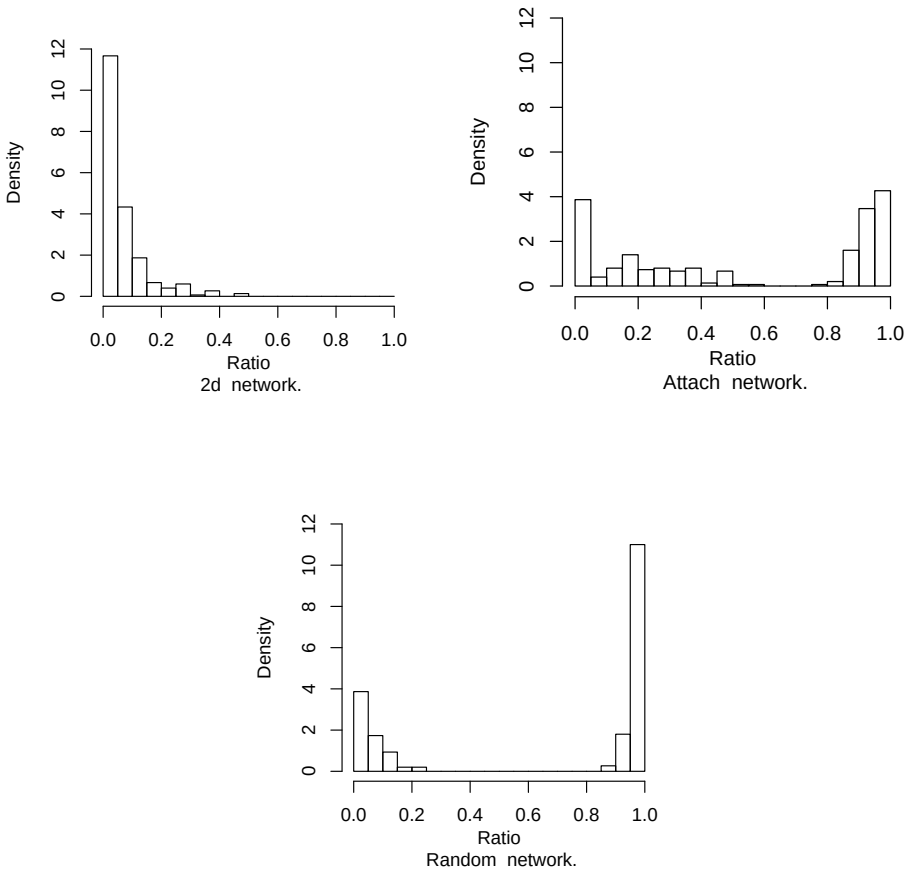

Figure 6.1: Histogram of peripheral nodes for various types of networks. The plot shows the probability density of the ratio of peripheral nodes to network size. Small ratios mean that only a few percentage of nodes are in the periphery, high ratios mean that a high percentage of nodes are in the periphery of the graph. Aggregated from networks with size equal to 50 and 500 nodes.

6.2 Other Distributions of Minimums

The accuracy of the Diameter Estimation technique is determined not only by the length of the minimums vector, K, and by the network topology, but also by the number of unique minimums generated, which depends on the distribution of the encoded values.

The standard technique encodes values using only the exponent of the random exponential value, so a real value v is encoded as $floor(log_2 v)$.

Figure 6.2 shows the histogram of the exponents in the exponential distribution.

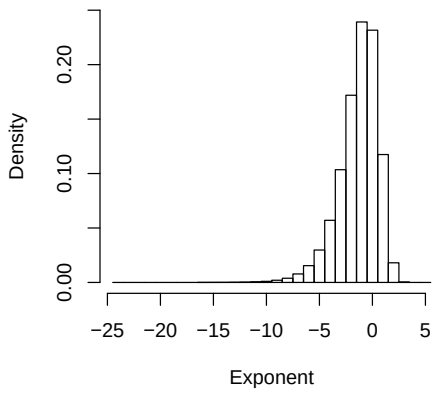

Figure 6.2: Histogram of exponents in exponential distribution ($rate = 1$).

From the point of view of unique minimums generation, this distribution is not the best.

To see why, consider Figure 6.3 which shows the cumulative probability distribution for the distribution of Figure 6.2 and the cumulative distribution for an encoding based on the geometric distribution. Clearly, for the same probability level, the geometric distribution gives a wider range of values than the exponential distribution. A wider range of values will lower the probability of a collision in the final minimums. This means that a better encoding than the default one, from the perspective of unique minimums generation, can be found by using a different distribution.

Although this is a characteristic of the Extrema Propagation technique, in cases where only an estimation of the network diameter is needed (and not data aggregation) the number of unique minimums generated may be improved by using a different distribution for the generation of values.

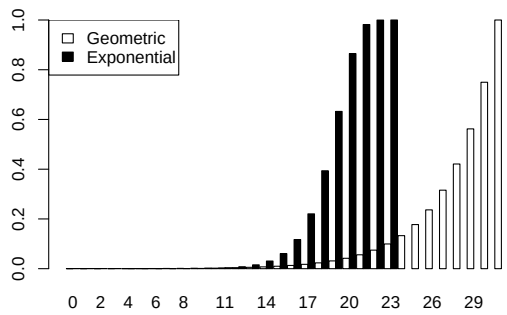

Figure 6.3: Cumulative probability distribution for the exponential and geometric based encodings. The values on the x axis were mapped to the $[0, 31]$ range for easier comparison. Based on a sample of 1000000 values.

6.2.1 Procedure

To study the effect of the probability distribution in the generation of the values used in the diameter estimation technique, three distributions were analyzed: uniform[1], geometric and the exponential distribution.

For each distribution, we simulated the generation of values on networks with size (N) equal to 50, 500 and 5000 nodes and calculated the ratio of unique minimums to K (for $K \in \{10, 100, 1000\}$) with 100 repetitions for each combination of N and K.

For all distributions values were encoded using 5 bit (following the original encoding size).

In the uniform distribution, values were simply generated in the interval $[0, 32[$ and rounded down to the nearest integer. For the geometric distribution, the following algorithm was used for the encoding (which results in a distribution similar to the one depicted in Figure 6.4):

```
function encodedGeometric(p)
    v ← random.geometric(p)
    if v > 31 then
        v ← 0
    end if
    Return 31 − v
end function
```

[1]The uniform distribution was analyzed only for comparison since it is not expected to generate good results.

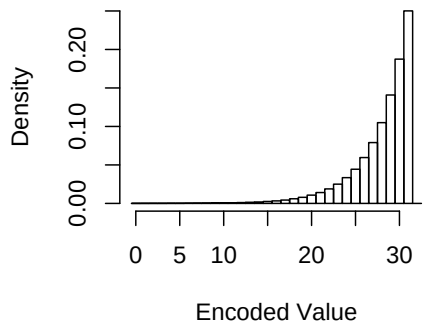

Figure 6.4: Distribution of the encoded values of the geometric distribution.

For the geometric distribution case, several *probability* parameter values were tested in order to determine the best one.

6.2.2 Results

Results show that using a geometric distribution with *probability* = 0.3 gives a higher ratio of globally unique minimums than using the standard exponent encoding or a uniform distribution. Table 6.2 summarizes the results.

Distribution	Average ratio	Standard Deviation
Exponential	0.7219	0.0853
Geometric[a]	0.8402	0.0758
Uniform	0.0923	0.1591

[a] *probability* = 0.3.

Table 6.2: Unique minimums ratio average and standard deviation using different distributions

The standard Extrema encoding produces, on average, $0.72 \times K$ of unique minimums. Higher values of K result is a smaller standard deviation, as can be seen in Figure 6.5.

Figure 6.6 shows the results for the geometric distribution. The chart plots the ratio of unique minimums to K against the *probability* parameter, for three network sizes. There are slight differences on the best parameter value for different network sizes, but *probability* = 0.3 achieves a good common ratio of over 80% for all three network sizes.

There are no significant differences on the average ratio, for different

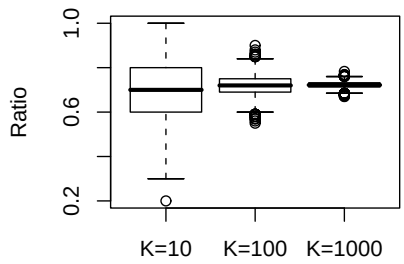

Figure 6.5: Box plot of the ratio of unique minimums to K, using the standard encoding, for several values of K.

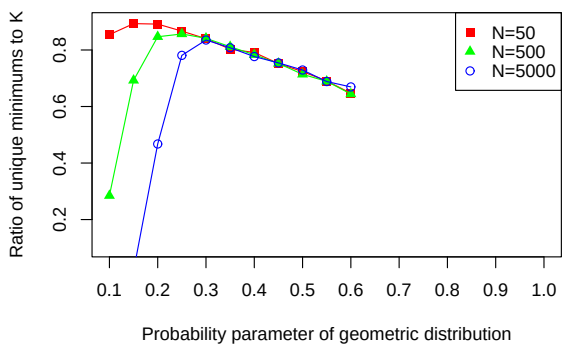

Figure 6.6: Ratio of unique minimums to K as a function of the *probability* parameter, for $N \in \{50, 500, 5000\}$.

K, which varies from 0.842, for $K = 10$ to 0.836 for $K = 1000$. Standard deviation, however, is lower for higher K, as illustrated by Figure 6.7 which shows the box plots for the ratio of unique minimums to K, for several K, for $probability = 0.3$.

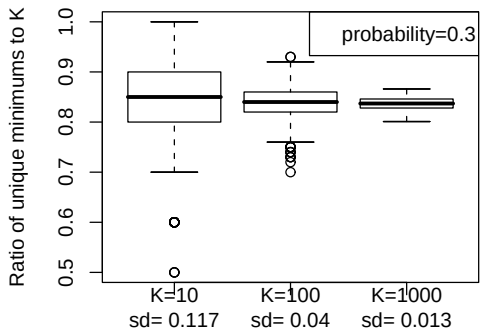

Figure 6.7: Boxplot of ratio of unique minimums to K, for several K, using the geometric distribution. Standard deviation (sd) values indicated below box names.

Using an uniform distribution for the generation of values results in an very low ratio of unique minimums. This is an expected result since only 5 bit were used, giving a range of $[0, 31]$ values which results in a high collision rate, even for very small networks. Table 6.3 shows the results for the uniform encoding.

Network Size	K	Average Ratio
50	10	0.411
500	10	0
5000	10	0
50	100	0.4107
500	100	0
5000	100	0
50	1000	0.4108
500	1000	2e-06
5000	1000	0

Table 6.3: Average ratio of unique minimums using the uniform distribution.

For medium/large networks, the probability of generation of unique minimums is close to zero, even for large values of K.

6.3 Trade-off between vector length and value encoding size

In the previous sections, each value in the `Minimums` vector was encoded using 5 bit. However, for the same size, there are several combinations of encoding size and vector length that can be used.

In order to determine if there is a better combination of encoding size and vector length than the one tested in the previous section, several combinations were tested, using the geometric distribution.

6.3.1 Procedure

In order to be able to compare results, we used a total message size of 500 bit (equivalent to using 5 bit to encode a value and a vector of $K = 100$).

To keep the same message size using different encoding sizes means that the vector length must change. The following table lists the combinations used in the following tests.

Encoding Size (bit)	Vector Length (K)
1	500
2	250
3	166[a]
4	125
5	100

[a] In this case, the vector length was rounded down. The total number of bits is 498.

6.3.2 Results

From Figure 6.8 it is apparent that there is no single optimum solution for the combination of encoding size and vector length across different network sizes. However, it is still possible to find a better result than 5 bit and $K = 100$, as can be observed in Figure 6.9.

Using 4 bit to encode each value (which means $K = 125$) and $p = 0.5$, the average number of unique minimums generated, across network sizes of $N \in \{50, 500, 5000\}$ is 88.51 as opposed to 84.02 using 5 bit. Although the ratio of unique minimums to K decreases when using 4 bit, what really matters is the absolute number of unique minimums generated. Table 6.4 shows the average number of generated minimums for different network sizes, using 4 and 5 bit for encoding size.

Effect on Diameter Estimation Accuracy

The accuracy gain that might be obtained by using a geometric distribution in the generation of values for the minimums vector was studied by running

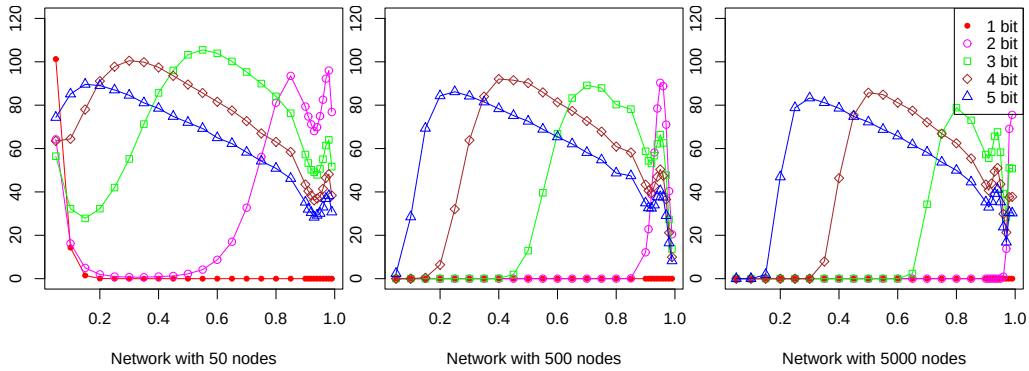

Figure 6.8: Unique minimums for different network sizes and combinations of encoding size and vector length.

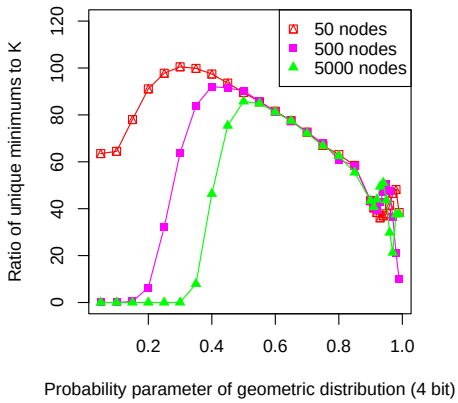

Figure 6.9: Unique minimums generated using 4 bit encoding. Probability parameter $p = 0.5$, originates over 85% of unique minimums across different network sizes.

	$N = 50$	$N = 500$	$N = 5000$
4 bit	89.54	90.22	85.76
5 bit	84.56	84.17	83.32

Table 6.4: Average unique minimums generation using 4 and 5 bit.

the tests described in Section 6.1, but using a geometric distribution with $p = 0.5$ and $K = 125$.[2]

To allow a direct comparison, both the geometric distribution and the standard extrema distribution were applied on the same graphs.

Table 6.5 shows the results, indicating the average relative error and standard deviation for both the standard encoding and the optimized geometric distribution. It also shows the gain (difference column) in the average relative error by using the geometric distribution. Results show a slight de-

		Geometric		Extrema		
Network Type	Average Diameter	Average Relative Error	Sd	Average Relative Error	Sd	Difference
			$N = 50$			
2d	10.22	0.0005	0.07280	0.0010	0.1028	0.0005
Attach	3.0333	0.0003	0.0305	0.0005	0.0400	0.0002
Random	2.3867	0.0005	0.0416	0.0010	0.0553	0.0005
			$N = 500$			
2d	34.8933	0.0156	0.7015	0.0195	0.8133	0.0039
Attach	3.7467	0.0383	0.3699	0.0453	0.3943	0.0070
Random	4	0.0195	0.2730	0.0251	0.3053	0.0056

Table 6.5: Comparison of the use of the standard distribution and the optimized geometric distribution in the diameter estimation error.

crease in the average relative error, in the order of 0.55% for 500 nodes networks and in the standard deviation, for all types of networks.

[2]Due to time constraints, only 50 and 500 nodes networks were tested.

Chapter 7

Evaluation of the Extrema Propagation in NeEM

The evaluation of the integration of the Extrema Propagation technique in NeEM was done using Modelnet as the simulation platform.

7.1 Simulation Setup

The following is a description of the general simulation setup and procedures that were used.

7.1.1 Modelnet

The experiment's network consisted of two computers—one emulator and one application host with the following characteristics:

Emulator:

- Pentium 4 CPU at 3.40 GHz
- 2 Gb RAM
- 3Com Etherlink XL 10/100 PCI TX NIC

Application host:

- Pentium 4 CPU at 3.40 GHz
- 1.5 Gb RAM
- Broadcom NetXtreme Gigabit Ethernet

Modelnet's graph file was created with the following command:

```
inet -n 3037 | inet2xml -l -p 800 among 50 stubs \
min-client-stub 1000 1 0 max-client-stub 1000 50 0
```

Which creates a network of 3037 nodes (the minimum for inet) and 800 clients[1] attached among 50 stubs.[2] The command also specifies the bandwidth of the client-stub link to be 1000 kbps, latency between $[1, 50]$ milliseconds and no packet drops. All the other link types have default parameters, except for latency which is inferred from the node distance.

7.1.2 NeEM Extrema

In order to save CPU, instead of launching several Java virtual machines running one instance of the protocol each, only one Java virtual machine was used and instances of the protocol ran as separate threads. This means that the function performed by Modelnet's `libipaddr` had to be done directly in Java, since the `libipaddr` library would not be able to deal with several IP addresses for different threads in the same process.

NeEM's implementation already allows multicast channels to have two addresses: one local address and one public address that is advertised to peers. So all that was needed was to launch each node with the address of the corresponding virtual node as the local address and turn on bit 23 of that address and assign it to the public address.

7.2 General Experiment Setup

The experiments were controlled by a set of shell scripts that invoked specific JMX functions on a Java class that represented the experiment. This class was responsible for loading all instances of the modified multicast channel, connecting them, starting the Extrema protocol and wait for it to finish.

The process of running an experiment consisted of:

1. Loading the main experiments class;
2. Adding N nodes by randomly picking N virtual node addresses from the set assigned to the machine;
3. Configuring each node according to the experiment settings;
4. Randomly connecting nodes among themselves;
5. Letting overlay network settle for a few seconds;
6. Starting the extrema protocol on one randomly chosen node;
7. Waiting until a preconfigured number of rounds had elapsed;

[1] A client node in Modelnet's terminology is an "edge node in the virtual network corresponding to an [*sic*] computer attached to the wide area internet" [Becker and Yocum, 2003, p. 10].

[2] "A gateway for client nodes to the access the network" [Becker and Yocum, 2003, p. 10].

8. Stopping the experiment;
9. Saving the log files.

The log files recorded the following:

- Message send timestamp;
- Message receive timestamp;
- After each round: round number, current `Minimums` vector, current `Hops` vector, timestamp, number of total sent messages.

7.2.1 Maximum Number of Nodes in an Experiment

Running several instances of the protocol on the same machine has two potential bottlenecks: the CPU and the network interface.

In the case of our experiments, the communication load of each node was fairly small so bandwidth was not a problem. On the other hand, running several threads of the protocol in the Java virtual machine turned out very CPU intensive.

The latency between nodes, for an increasing number of nodes and for different configurations, was measured and the results showed that, for some configurations and number of nodes, the latency between nodes increased beyond that imposed by Modelnet (and the message processing overhead at the machine).

This restriction led to the use the following maximum values in most of the experiments: $K = 10$, $MaximumNumberOfNodes = 75$ and $OverlayFanout = 10$.

7.3 Timeout and F

In order to determine the best combination of $Timeout$ and F to use in the NeEM Extrema protocol, several combinations were tested.

The base values for the tested parameters were $Timeout = 200$, $F = 0.8$ (in the rest of this document, F is the fraction of neighbours to wait for) and the $F\_PLUS\_TIMEOUT$ strategy.

For each of the three parameters, several values were tested, maintaining the other parameters with the base values. This led to the variations enumerated in Table 7.1.

For each configuration, 100 runs were executed and the log files were analyzed to determine the number of rounds, and the total time until convergence of the `Minimums` vector.

Timeout	Strategy	F
200	F_PLUS_TIMEOUT	0.8
400	F_PLUS_TIMEOUT	0.8
600	F_PLUS_TIMEOUT	0.8
400	ONLY_TIMEOUT	0.8[a]
400	TIMEOUT_PLUS_F	0.8
400[b]	ONLY_F	0.8
400	F_PLUS_TIMEOUT	0.5
400	F_PLUS_TIMEOUT	0.9

Table 7.1: Configurations for the NeEM Extrema implementation.

[a]The $ONLY\_TIMEOUT$ strategy does not take F into account.
[b]The $ONLY\_F$ strategy does not take $Timeout$ into account.

7.3.1 Results

The average number of rounds until convergence for the different combinations is shown in Figure 7.1.

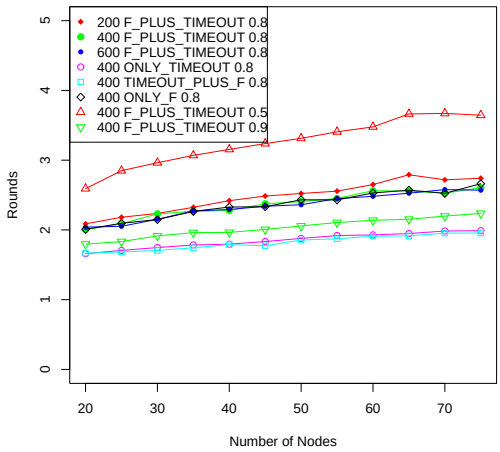

Figure 7.1: Average number of rounds ellapsed until convergence.

The plots on Figure 7.1 indicates, from the number of rounds perspective, that:

- Trying to receive all possible messages before advancing round produces the best results. This is what happens in the $ONLY\_TIMEOUT$ and the $TIMEOUT\_PLUS\_F$ settings.
- Waiting for only half of the neighbours is the worst setting. Clearly, in

that setting, the timeout is not being triggered, meaning that rounds are advancing with only half the messages received increasing the number of needed rounds.

- A timeout of 400 milliseconds seems to be enough for most nodes to communicate, since the $ONLY\_TIMEOUT$ setting is very similar to the $TIMEOUT\_PLUS\_F$ with $F = 0.8$. This means that at least 80% of the neighbours communicate within the 400 milliseconds interval.

Figure 7.2 shows the average time a node needs to achieve convergence. It indicates, from the total time perspective, that:

- Slower rounds, although implying less number of rounds, produce worst results. This can be seen from the top lines, which correspond to the better settings from the number of rounds point of view.

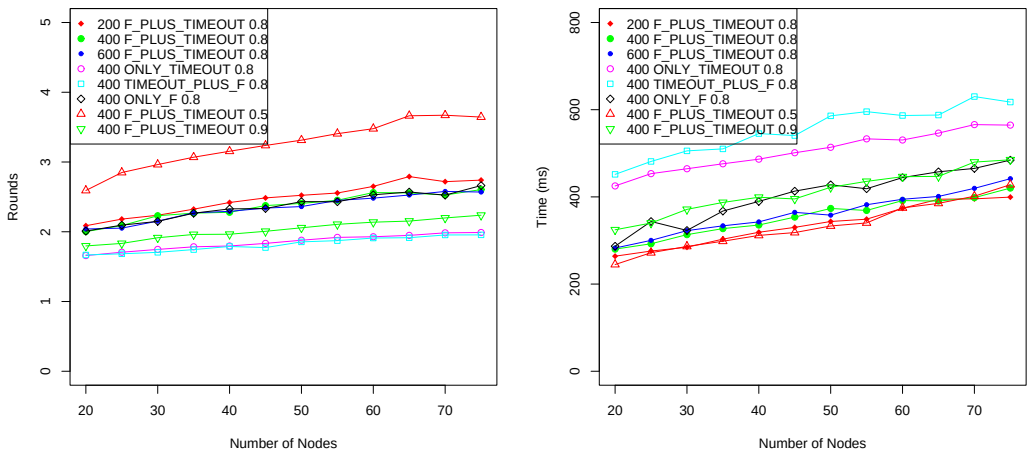

Figure 7.2: Average time until convergence.

Comparing both Figures, its apparent that there is a trade-off between time and communication (the number of rounds is an indirect measure of messages sent): faster convergence means higher communication cost.

7.3.2 No news Rounds

Although convergence of the `Minimums` vector can be determined by examining the log files, it cannot be determined by individual nodes. Nodes have to assume convergence after a predefined number of rounds have elapsed without changes in the `Minimums` vector.

In the previous tests, the maximum number of no news rounds (NN) observed were the following:

Timeout	Strategy	F	NN
200	F_PLUS_TIMEOUT	0.8	5
400	F_PLUS_TIMEOUT	0.8	3
600	F_PLUS_TIMEOUT	0.8	4
400	ONLY_TIMEOUT	0.8	2
400	TIMEOUT_PLUS_F	0.8	2
400	ONLY_F	0.8	4
400	F_PLUS_TIMEOUT	0.5	7
400	F_PLUS_TIMEOUT	0.9	3

Although the values are fairly similar, it appears that faster rounds lead to a larger NN value than slower rounds.

7.4 Diameter Estimation

Convergence of the `Hops` vector cannot be determined by examining the log files, in normal circumstances. This happens because the overlay topology changes over time (NeEM "shuffles" peers periodically).

In order to get a rough idea of how long it might take for the Hops vector to converge in a static topology, a different test was performed. Before starting the Extrema protocol, all nodes were configured to disable shuffling, thus freezing the current overlay topology. A snapshot of the topology was then taken and recorded in a graph file for posterior analysis.

Two configurations were tested: one in which rounds are fast, because only half the neighbours are required in order to advance; and one in which rounds are slow, because nodes wait for every neighbour until a timeout and then proceed after a fraction of neighbours have responded. This corresponded to the following parameters:

- Fast rounds: $TimeOut = 400$, $Strategy = F\_PLUS\_TIMEOUT$, $F = 0.5$.
- Slow rounds: $TimeOut = 400$, $Strategy = TIMEOUT\_PLUS\_F$, $F = 0.8$.

For each value of $K \in \{10, 100\}$, 100 runs were executed and recorded.

The log files were analyzed (specifically the `Hops` vector) to determine the error in the diameter estimation in the rounds after convergence of the `Minimums` vector. The overlay graph file served as reference for the true diameter of the network.

7.4.1 Results

Figure 7.3 shows the average absolute error in the diameter estimation as a function of the number of rounds after convergence of the `Minimums` vector,

for different values of K.

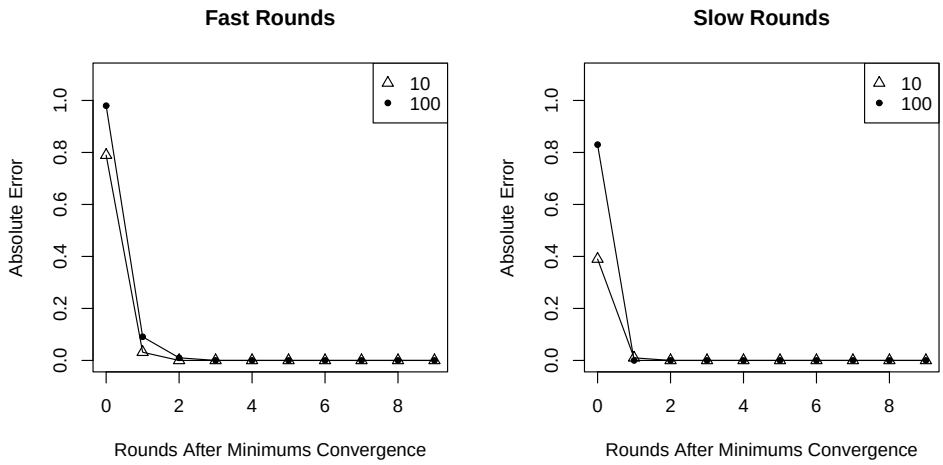

Figure 7.3: Additional rounds needed to estimate the diameter.

In both configurations, it is apparent that the `Hops` vector takes more time to converge than the `Minimums` vector. This happens because some shorter paths are slower than longer paths. So it may happen that a hop count is substituted by a lower count that came from a shorter, slower path. However, the same node may already have the final `Minimums` vector. Slower rounds, in which nodes wait a long time before advancing, tend to mask this effect since nodes on fast paths will still have to wait a long time before advancing.

7.5 Round Number in Messages

In the previous experiments, a message, with a round number of r, received by a node in round r' was only considered in the accounting of neighbours that had already communicated if $r \geq r'$. Meaning that messages from nodes that were behind the current node would not count (the message content would still be integrated in the `Minimums` and `Hops` vector, though). The rationale behind this was that these messages would correspond to old data from slow nodes and thus should not count to the current round.

However, to study the effect of counting all messages despite their round number, the implementation was modified and tested with default parameters: $Timeout = 400$, $Strategy = F\_PLUS\_TIMEOUT$, $F = 0.8$. Only networks of 50 nodes were tested. For each mode (consider round number in messages and do not consider round number in messages) 100 runs were executed and recorded.

7.5.1 Results

Results show that considering round number in messages leads to a slower convergence, i.e., the average total time to converge increases, but the average number of round to converge is lower than when not considering round number in messages. Figure 7.4 shows the box plots of the two modes for average number of rounds and total time to converge.

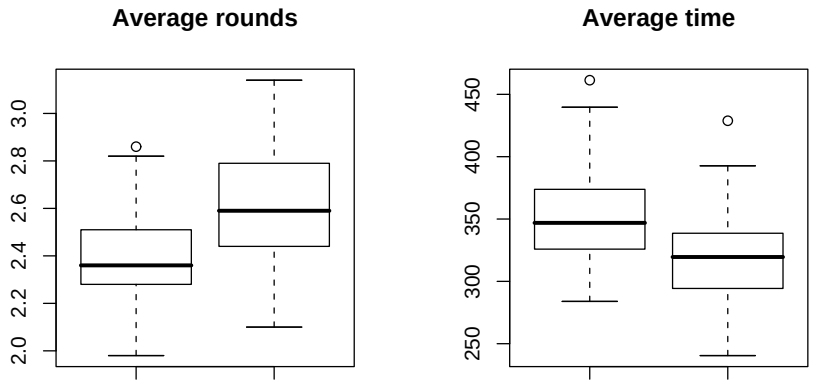

Figure 7.4: Box plots of average number of rounds and average time for convergence. In each chart: Left – Considering round number in message; Right – Not considering round number in message.

7.6 Evaluation of Message Optimization

In order to evaluate the message size improvements presented in Section 3.2, the implementation of Extrema in NeEM was extended in order to log, in each round, the number of values in the `Minimums` vector that needed to be sent to each neighbour, based on the comparison of the current `Minimums` vector and the last vector received from each neighbour. The same was done for the `Hops` vector.

The analysis was done considering two cases separately: messages without the Hops vector (only size estimation would be performed); and messages with the Hops vector (both size and diameter estimation would be performed).

The message size improvement procedure for the message with the Hops vector is analogous to the one presented in Section 3.2.

Five hundred runs were executed with parameters: $Timeout = 400$, $F = 0.5$, $Strategy = F\_PLUS\_TIMEOUT$ and $K = 10$. Only networks

with 65 nodes were tested.

7.6.1 Results

Table 7.2 summarizes the results and shows the average message size reduction in a run of the algorithm, per node, for different values of T, with and without the `Hops` vector included in the message.

Without `Hops` vector

T	Average reduction	Standard deviation	Average vector length
0	3.37%	1.38%	7.94
1	8.62%	1.26%	6.88
2	13.91%	1.03%	5.94
3	18.34%	0.86%	5.19
4	21.85%	0.75%	4.61
5	24.66%	0.67%	4.14
6	26.95%	0.62%	3.77

With `Hops` vector

T	Average reduction	Standard deviation	Average vector length
0	7.97%	2.12%	7.95
1	15.63%	1.95%	6.92
2	23.18%	1.61%	5.98
3	29.44%	1.34%	5.24
4	34.40%	1.15%	4.65
5	38.38%	1.02%	4.18
6	41.62%	0.93%	3.80

Table 7.2: Average message size reduction achieved using the improvements presented in Section 3.2.

As expected, the overall message size reduction increases as T increases. This happens because, near the final rounds of the algorithm, almost all nodes have the final vectors and their neighbours know this. This allows each node to send only a few values to each neighbour instead of the total K values.

When considering the combined techniques for size and diameter estimation (message with `Minimums` and `Hops` vector), the reduction in message size is more pronounced than when considering only the size estimation (only `Minimums` vector). This happens because, in our implementation, each value in the Hops vector was coded using 8 bit, which in most cases would be too conservative a value. This leads to a greater reduction in size when using the improved message format.

The average vector length column in Table 7.2 shows the average number of values sent by each node to each neighbour (K was set to 10). With three safe-guard rounds ($T = 3$) only about five values out of ten have to be sent, on average. For the two cases (only `Minimums` vector and `Minimums` plus `Hops` vector) the average vector length is very similar (which might contradict the difference in the values of the average reduction column, but that was explained in the previous paragraph).

Chapter 8

Conclusions and Future Work

This document presented a study on the behaviour of the Extrema Propagation technique on a more realistic scenario, with different configuration settings. It also presented and evaluated a diameter estimation technique based on the Extrema Propagation technique.

When failures are allowed in the system model, the Extrema Propagation algorithm has to be slightly modified. Instead of waiting for every neighbour, which could lead to deadlocks, each node should only wait for a fraction of its neighbours, wait until a timeout occurs, or a combination of the two.

Results show that there is a clear trade-off that must be considered. Fast rounds—rounds in which each node advances rapidly either because they wait for a small fraction of the neighbours or because a small timeout was configured—lead to a faster convergence of the `Minimums` vector and hence a faster estimation of the network size. However, fast rounds also lead to a larger number of rounds needed, which is an indirect measure of the number of messages exchanged. Slow rounds—in which each node waits for a large number of neighbours—lead to slower estimation but also to a smaller number of messages exchanged. The trade-off between communication cost and time must the considered.

Although the Extrema Propagation technique is already very efficient in terms of number of steps, since it achieves the theoretical minimum of *Diameter* steps (not quite due to the extra safe-guard rounds needed), it is still amenable to improvements in the size of the message that is used. The proposed improvement described in section 7.6 shows that its possible to reduce the total communication cost using a simple approach. Even without considering additional safe-guard rounds, there is an average reduction of 3.4% in message size when considering only the basic Extrema Propagation technique and almost 8% when combining the diameter estimation technique. When aditional safe-guard rounds are used, the savings are even

more substantial, rising to 18% and 29% for 3 aditional safe-guard rounds.

A diameter estimation technique based on the Extrema Propagation was presented and evaluated. The technique takes advantage of the generation of a fixed number of global minimums that act as a node sampling means to calculate the eccentricities of those nodes. If one of the "sampled" nodes is a peripheral node and enough rounds are allowed to pass, then the network diameter can be calculated. The accuracy can be adjusted by the parameter K—the number of minimums to generate. Results from simulations show that even with a small number of minimums ($K = 10$), the relative error in the estimation is only about 10%.

If only an estimate of the diameter is required (and not of the network size) then a better distribution for the generation of minimums can be used instead of the one used by the Extrema Propagation. Using a geometric distribution rises the number of global unique minimums, slightly increasing the accuracy of the estimation.

The Extrema Propagation technique was partly integrated into the NeEM framework. Full integration would simply be a matter of feeding the estimated values back into the gossip and overlay layers which could use the estimated values for a better configuration of the fanout and time-to-live parameters.

8.1 Future Work

The message format presented in section 3.2 suggests yet another way to reduce the message size. Although each node generates K values, only a few will be global minimums. If all minimums were unique, then, on average, each node would produce $\frac{K}{N}$ global minimums, which means that the average ratio of unique minimums to K is $\frac{1}{N}$. Even for small networks, e.g., 50 nodes, this is only 2% of K. Even when considering that duplicate minimums are generated, these values do not change much.

This means that much potentially useless information is sent in each message. The optimal approach would be for each node to send only the final global minimums, but that is obviously not possible to determine beforehand. However, a potentially better way would be to send only a fraction of the K values (the smallest ones). For example, send $\frac{1}{8}$ of K on the first round, $\frac{1}{4}$ in the second, etc., and combine this with the message optimization already proposed, i.e., not sending minimums to a node that already has them.

This approach would probably slow down the convergence time, but decrease the communication cost. Further study would be required to verify this assumption.

Bibliography

[Baquero et al., 2007] Baquero, C., Almeida, P., and Menezes, R. (2007). Extrema propagation: Fast distributed estimation of sums and network sizes. Technical report, DI/CCTC and DMCT, Universidade do Minho. http://gsd.di.uminho.pt/members/cbm/ps/techrepmay2006.pdf.

[Barabási and Albert, 1999] Barabási, A. L. and Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439):509 – 512.

[Bavier et al., 2004] Bavier, A., Bowman, M., Chun, B., Culler, D., Karlin, S., Muir, S., Peterson, L., Roscoe, T., Spalink, T., and Wawrzoniak, M. (2004). Operating system support for planetary-scale network services. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 19–19, Berkeley, CA, USA. USENIX Association.

[Becker and Yocum, 2003] Becker, D. and Yocum, K. (2003). Modelnet howto. Technical report, Duke University. `http://modelnet.ucsd.edu/howto.html`.

[Dolev et al., 2006] Dolev, S., Schiller, E., and Welch, J. L. (2006). Random walk for self-stabilizing group communication in ad hoc networks. *IEEE Transactions on Mobile Computing*, 5(7):893–905.

[Erdös and Rényi, 1959] Erdös, P. and Rényi, A. (1959). On Random Graphs I. *Publicationes Mathematicae*, 6:290–297.

[Eugster et al., 2001] Eugster, P., Handurukande, S., Guerraoui, R., Kermarrec, A., and Kouznetsov, P. (2001). Lightweight probabilistic broadcast.

[Ganesh et al., 2003] Ganesh, A. J., Kermarrec, A.-M., and Massoulie, L. (2003). Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, 52(2):139–149.

[Hagberg et al., 2008] Hagberg, A., Schult, D., and Swart, P. (2008). Networkx. Webpage. `https://networkx.lanl.gov/wiki` (Accessed July 2008).

[Jelasity et al., 2005] Jelasity, M., Montresor, A., and Babaoglu, O. (2005). Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(1):219–252.

[Jesus, 2007] Jesus, P. (2007). Agregação e contagem em redes p2p. Master's thesis, Escola de Engenharia, Universidade do Minho.

[Jesus, 2008] Jesus, P. (2008). Robust distributed data aggregation – thesis planning. Technical report, Universidade do Minho.

[Jesus et al., 2006] Jesus, P., Baquero, C., and Almeida, P. (2006). Id generation in mobile environments. In José, R. and Baquero, C., editors, *CSMU 2006 – Conference on Mobile and Ubiquitous Systems*, pages 159–162. Escola de Engenharia, Universidade do Minho. ISBN: 972-8692-29-3.

[Jin et al., 2000] Jin, C., Chen, Q., and Jamin, S. (2000). Inet: Internet topology generator. Technical report, University of Michigan.

[Kempe et al., 2003] Kempe, D., Dobra, A., and Gehrke, J. (2003). Gossip-based computation of aggregate information. *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 482–491.

[Lynch, 1997] Lynch, N. A. (1997). *Distributed Algorithms.* Morgan Kaufmann.

[Pereira et al., 2003] Pereira, J., Rodrigues, L., Monteiro, M. J., Oliveira, R., and Kermarrec, A.-M. (2003). Neem: Network-friendly epidemic multicast. In *Proc. 22nd International Symposium on Reliable Distributed Systems*, Florence, Italy. IEEE, IEEE Computer Society.

[Psaltoulis et al., 2004] Psaltoulis, D., Kostoulas, D., Gupta, I., Birman, K., and Demers, A. (2004). Practical algorithms for size estimation in large and dynamic groups. Technical report, University of Illinois, Urbana.

[Vahdat et al., 2002] Vahdat, A., Yocum, K., Walsh, K., Mahadevan, P., Kostic, D., Chase, J., and Becker, D. (2002). Scalability and accuracy in a large-scale network emulator. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI).*