Master Thesis Report

A Robust Protocol for Aggregation in Dynamic Peer-to-Peer Networks

By

Vidya Lakshmi Rajagopalan
Supervisor: Dr. Spyros Voulgaris

August 2012

This thesis is submitted in partial fulfillment of the requirement of
Master of Science in Parallel and Distributed Computer Systems
in the
Department of Computer Science, Faculty of Sciences
Vrije Universiteit, Amsterdam

Abstract

Aggregation in peer-to-peer networks provides a global view of system properties, such as average computational load and total free disk space, which can be used for administering the system. Among the various aggregation protocols, gossip-based aggregation protocols provide faster aggregation of the node values in a scalable and adaptive manner. This is particularly important in peer-to-peer networks which are essentially dynamic, due to node failures, node churn, communication delays and communication failures. These dynamics affect the correctness of the system aggregate, which is a function of the individual node values (node contributions).

We first propose a protocol hierarchy for gossip-based aggregation that provides the capability to perform accurate estimation of the system aggregate, even in the presence of node failures. In this protocol hierarchy, we develop a robustness protocol for aggregation in dynamic peer-to-peer networks that ensures the correctness of the system aggregate amidst node failures. Our protocol detects node failures and performs the necessary cleanup operations. The evaluation of our protocol shows that the error of the converged aggregate from the system aggregate is very low. We observe that the failure detection time is independent of the network size and the number of nodes that fail. Additionally, we investigate the factors affecting the accuracy of a node's estimate of the system aggregate (node estimate) and introduce a metric called the 'confidence value' which gives an indication of this accuracy at runtime. Hence, our robustness protocol enables accurate computation of system aggregate even in the presence of node failures.

Acknowledgements

I would like to thank all the people without whose guidance and help, this thesis would not have been possible. First and foremost, my utmost gratitude to my supervisor, Dr. Spyros Voulgaris for his guidance and encouragement throughout the work. Regular discussions with him have helped to shape the idea behind the thesis and have helped me greatly to progress.

I am very grateful to VU Amsterdam and Nuffic for supporting me with scholarship, without which my master studies at VU Amsterdam would not have been possible.

Next, I would like to thank my friends who have been a constant source of motivation throughout my studies. I thank them for proof reading my thesis and for the useful comments which helped to shape my thesis.

Last but not least, I thank my family for their continuous support.

Contents

List of Figures

CHAPTER 1

Introduction

Peer-to-peer networks are being extensively used in many commercial and scientific applications [**10**]. For effective monitoring, control and management of the peer-to-peer networks, a global view of the properties of the system is required. The various properties of the nodes, e.g., the computational load and free disk space, are represented by a set of node values, which we refer to as *contribution* of the nodes. The aggregation of these node contributions, given by an aggregation function, provides a globalized view of the system, which can be used for administering the system. The aggregation function can be the sum, average, maximum, and minimum among others.

Over the past decade, a significant amount of research work has focussed on developing aggregation protocols for peer-to-peer networks [**6**][**3**]. Among them, the aggregation protocols based on gossiping enable faster computation of the system aggregate in addition to being scalable and adaptive [**6**] [**8**] [**9**]. The nodes gossip with each other their current estimate of the system aggregate (node estimate) and recompute their node estimate based on the value they receive from the node with which it gossiped. Hence, the node estimates change with time and converge towards the system aggregate.

Peer-to-peer networks are essentially dynamic in nature, owing to the failure of nodes, arrival and departure of nodes (node churn), message losses, communication delays, communication link failures etc. Consequently, the aggregation protocol should include the necessary features that ensure the correctness of the system aggregate under these conditions. There are works that handle the arrival of nodes, communication delays and communication link failures during aggregation [**6**][**4**]. Our work focuses on maintaining the correctness of the system aggregate and thereby, the accuracy of the node estimates during aggregation in peer-to-peer networks when nodes fail. By node failures, we refer to the failure of nodes or ungraceful departure of the nodes from the system. Maintaining the correctness of system aggregate means conserving the system mass, which is the sum of the contributions of the nodes in the network. In our work, we focus on the aggregation function that computes the system average.

1.1. Problem Statement

We aim to address the following questions in our work :

(1) How to detect node failures during aggregation?
(2) What are the effects of node failures and node churn on aggregation and what are the cleanup steps to be taken to maintain the correctness of the system aggregate and the accuracy of the node estimates?
(3) Is it possible to determine the accuracy of the node estimates during aggregation?
   - What are the factors that determine the accuracy of the node estimate?
   - How can the accuracy of the node estimate be determined?

1.2. Our Contributions

We study the effect of node failures and node churn on aggregation in peer-to-peer networks. We introduce a protocol hierarchy for gossip-based aggregation that provides the capability to perform accurate estimation of the system aggregate inspite of node failures. In this protocol hierarchy, we develop a robustness protocol for aggregation in dynamic peer-to-peer networks that detects node failures during aggregation. The protocol also performs the necessary cleanup steps that maintain the correctness of the system aggregate and accuracy of the node estimates. Through simulations and evaluations, we show that our protocol is successful in maintaining the correctness of the system aggregate in case of node failures. Through experimental analysis, we investigate the factors which affect the accuracy of node estimates during aggregation. We introduce a confidence metric that indicates the accuracy of the node estimate under various dynamic conditions.

1.3. Thesis Outline

In chapter 2, we describe the problem of aggregation in peer-to-peer networks and the research work that has been done in this area. We explain the motivation for our work. We introduce our robustness protocol in chapter 3, that includes a scheme for detection of node failures during aggregation and a cleanup algorithm. We compare the existing failure detection schemes and explain why we chose the proposed detection mechanism. Finally, we study the cleanup steps to be performed after detecting node failures and propose an algorithm that maintains the correctness of the system aggregate even after node failures. In chapter 4, the experimental setup of our simulations is explained and the evaluations done in static and dynamic peer-to-peer networks are discussed along with the inferences drawn from it. The performance of our protocol is also shown in the chapter. We study the factors affecting the accuracy of node estimate in chapter 5. We introduce a metric called the confidence value, which gives an indication of the accuracy of the node estimate and show evaluations based on it. We conclude in chapter 6, mentioning the future work which can be done in this direction.

CHAPTER 2

Aggregation in Peer-to-Peer Networks

For the effective monitoring and control of peer-to-peer networks, a global view of a system property, such as the average computational load or the total free disk space, is required. Each node in the system has an associated numerical value, which we refer to as the contribution of the node, which corresponds to a property of the node, such as the computational load and the free disk space. The aggregation of the contributions from all the nodes, given by an aggregation function, provides a global view of the system. The computed aggregate can be the sum, average, maximum, minimum, standard deviation, variance etc, of the node contributions. The aggregation should be carried out in a decentralized manner, so that each of the node in the system obtains the system aggregate.

In this chapter, we discuss the various aggregation protocols for peer-to-peer networks. Next, the dynamicities in peer-to-peer networks are explained, followed by the existing works that deal with aggregation in dynamic peer-to-peer networks. We focus on protocols that compute the system average.

2.1. An Overview of Decentralized Aggregation protocols

There have been many works that describe aggregation protocols for peer-to-peer networks. Decentralized aggregation protocols can be classified into tree-based aggregation protocols [**3**] and gossip-based aggregation protocols[**6**][**7**][**8**][**9**]. Tree-based aggregation protocols rely on constructing and maintaining a hierarchical structure like a tree, which is used to compute the aggregate[**3**]. While this method has a low message overhead, they require the runtime maintenance of a complex tree structure. Also, they are not scalable and are reactive protocols, which means that the computation is performed for a specific query and it doesn't reflect the changes due to the dynamism of the system. In gossip-based aggregation protocols, the nodes compute aggregate through gossiping. Nodes periodically interact (gossip) with each other, and modify their node estimates. After some rounds of gossiping, the node estimates converge to the system average. Gossip-based aggregation protocols are better than tree-based aggregation protocols because they are adaptive, that is, the computation of aggregate value is continuous and considers the changes in the contributions of the nodes over time. Also, gossip-based aggregation protocols are scalable and doesn't require the need for the maintenance of structures like trees required by tree-based aggregation protocols. In our work, we focus on gossip-based aggregation protocols.

The protocols described in [**6**], [**8**] and [**9**] are gossip-based protocols for aggregation. In [**8**], the authors describe a push-based protocol which is asymmetric, where the gossiping sessions affect the local estimate of the initiating node only. Protocols described in [**6**] and [**9**] are push-pull based protocols, in which the local estimates of the initiating node as well as receiving node is affected. Hence, push-pull style protocols converge faster than push-based protocols. The protocol in [**6**] (Jelasity protocol) is a lightweight, adaptive, decentralized aggregation protocol aimed at rapid convergence and lesser cost. Compared to [**9**], the averaging operation in [**6**] is simpler and cleaner, helping in faster convergence in stable peer-to-peer environments.

The basic averaging protocol that we use in our work is an adapted version of the Jelasity protocol and will be described in the later sections.

2.2. Aggregation Protocols in Dynamic Peer-to-Peer Networks

Peer-to-Peer networks are highly dynamic, subject to sudden departure of nodes, failure of nodes, arrival of new nodes to the system, message delays, message losses, failure of communication links etc. Aggregation in a dynamic peer-to-peer network is susceptible to these dynamicities of the system, as a result of which the correctness of the system aggregate and thereby, the accuracy of the node estimates is affected. Maintaining the correctness of system aggregate means conserving the system mass, which is the sum of the contributions of the nodes in the network. In our work, we focus on maintaining the correctness of the system aggregate and accuracy of node estimates, during node failures.

Now we describe the aggregation protocols that deal with node failures and cleanup and discuss their drawbacks. The aggregation protocol described in [**8**] deals only with node failures during initialization stages. It doesn't deal with node failures and node churn during the aggregation. In [**12**], the authors extend the aggregation protocol from [**8**] to make it robust against node failures. But the extended protocol is robust for failures only if nodes who are neighbours do not fail in short time of each other[**8**]. In [**6**], the aggregation protocol is based on epochs, which are periods during which the gossiping algorithm is executed. In order to deal with node failures, the protocol is restarted periodically. In [**9**], failure detection is not described, but mechanism for removing a failed node's mass from the system is described. The drawback being that, it relies on an acyclic graph topology. In [**4**], the author describes schemes for failure detection and cleanup, but the performance of the scheme is observed to be not good.

We introduce a failure detection mechanism and recovery algorithm that doesn't depend upon the above constraints and assumptions. Our failure detection mechanism detects node failures at any time during aggregation and is adaptive, so that the changes in the network is adapted quickly to the computation of the system aggregate.

CHAPTER 3

A Robust Protocol for Aggregation

In this chapter, we introduce a robustness protocol for aggregation that detects node failures during aggregation and performs the necessary cleanup steps.

3.1. System Model

We consider a peer-to-peer network of size N that consists of a large number of nodes with unique identifiers [**6**]. The nodes of the network communicate with each other through messages. The nodes are arranged in an unstructured peer-to-peer overlay. Each node knows only a subset of the nodes in the system. This subset of nodes that are known to each node forms the neighbourhood of that node. We assume that the overlay network can change dynamically.

The network is highly dynamic, subject to sudden departure of nodes (failures) or voluntary departure of nodes, arrival of new nodes to the network, message delays, message losses and communication link failures. Each node has a local clock associated with it. Also, each node has a value associated with it called the contribution of the node, which is denoted by x_i. The estimate of node i at an instance is denoted by $\overline{x_i}$. The contributions of the nodes may change at runtime.

3.2. Protocol Hierarchy for Aggregation

One of our contributions is the introduction of a protocol hierarchy for gossip-based aggregation in peer-to-peer networks. The protocol hierarchy consists of the following protocols :

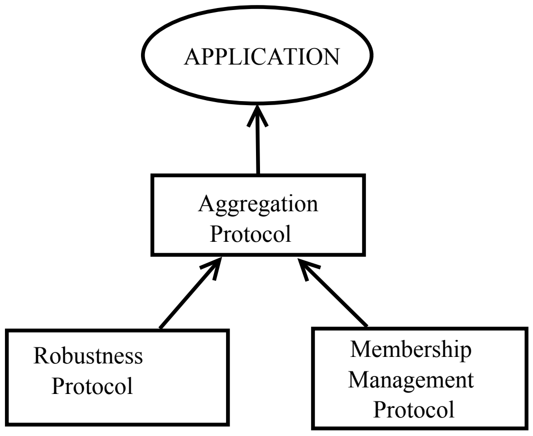

Figure 3.1. Protocol hierarchy for gossip-based aggregation

(1) Aggregation Protocol : The basic protocol executed by each node in the network resulting in the decentralized aggregation of node contributions.
(2) Membership Management Protocol : The protocol responsible for constructing the unstructured peer-to-peer network overlay required by the aggregation protocol.
(3) Robustness Protocol : The protocol responsible for detecting node failures and taking corrective actions to repair the corresponding aggregation errors.

3.2.1. Aggregation Protocol

The basic aggregation protocol that we have used for our work is from [**4**], which is a modified version of the protocol proposed in [**6**]. It is designed to adapt to asynchronous settings and deal with graceful departure of nodes and contribution updates. The algorithm is described in Figure 3.2. The aggregation protocol is based on a push-pull scheme. The protocol proceeds in rounds, which are periods of time when the information exchange is performed. In each round, the nodes invoke the function NextCycle() and choose a random peer from their neighbourhood set using the function getNeighbour(). The node gossips its local aggregate with the neighbour and recomputes its local aggregate when it receives a value from its peer.

3.2.2. Membership Management Protocol

Membership management protocols are responsible for constructing and maintaining the overlay network. It provides the service of providing neighbours to the aggregation protocol layer. For our experiments, we used Cyclon which is a gossip-based membership management protocol that is highly resilient to node failures[**11**]. Through a gossip-based enhanced shuffling method, Cyclon provides a continuosly changing set of peers for each node in the network. This results in faster propogation of the node contributions throughout the network, resulting in faster aggregation of node contributions towards the system aggregate.

```
function NextCycle($i$)
    $p = $ GetNeighbour($i$)
    REQUEST($p, \overline{x_i}, i$)
end function

function REQUEST($i, val, p$)
    RESPONSE($p, \overline{x_i}, val, i$)
    $\overline{x_i} = \frac{\overline{x_i}+val}{2}$
end function

function RESPONSE($i, val, sval, p$)
    if $val = sval$ then
        $\overline{x_i} = \frac{\overline{x_i}+val}{2}$
    else
        $\overline{x_i} = \frac{val-sval}{2}$
    end if
end function
```

Figure 3.2. A gossip-based aggregation protocol [**4**]

3.2.3. Robustness Protocol

Aggregation in dynamic networks has to deal with node churn and node failures. The robustness protocol aims at maintaining the correctness of the system aggregate under these dynamicities. Our work aims at developing a robustness protocol that can detect node failures and perform necessary cleanup operations during aggregation in a peer-to-peer network so as to ensure the correctness of the system aggregate and thereby maintain the accuracy of the node estimates.

3.3. Overview of the Robustness protocol

3.3.1. The Protocol

Our robustness protocol includes

(1) Failure detection
(2) Failure cleanup algorithm

3.3.2. Characteristics

Since we aim at maintaining the correctness of the system aggregate, the robustness protocol for aggregation should satisfy the following requirements.

(1) The failure or departure of a node should be detected by at least one node.
(2) The removal of the failed node's contribution should be performed only once.
(3) False positives in failure detection should be avoided.

The robustness protocol is explained in the following sections.

3.4. Failure Detection

We have used K-failure detectors [**5**] for failure detection during aggregation in dynamic peer-to-peer networks. The basic working of K-failure detectors is as follows. Each node in the network is monitored by one or more other nodes. Every monitored node periodically sends a message called heartbeat message to the monitoring nodes. Based on the delay in the arrival of the heartbeat messages from the monitored node, the monitoring node computes a real value called the 'suspicion value'. The suspicion value of a monitoring node q about a monitored node p expresses the confidence that q has about whether p has failed.

We chose K-failure detectors due to the following reasons. Firstly, since we focus on maintaining the correctness of the system aggregate, we aim at conservative failure detection that aims at reducing wrong suspicions at the cost of an increase in the failure detection time. K-failure detectors are adapted for conservative failure detection since they take into account the correlation of messages. Secondly, K-failure detectors are adaptive as the failure detection mechanism considers the changing network conditions. Thirdly, K-failure detectors are a type of accrual failure detectors. While traditional failure detectors give a binary output, which outputs if the node is trusted or suspected, accrual failure detectors output a real value for each node, which provides an estimate of the level of suspicion about the failure of a node. Hence, in case of accrual failure detectors, the interpretation of data is done by the applications. This enables different applications to interpret the output from failure detector according to their quality of service requirements.

In [**2**], the authors have extended the K-failure detectors to include clustering of nodes where the suspicion values are gossiped within nodes in a cluster, so as to improve the confidence that a node has about another node's failure. We combine the ideas from [**5**] and [**2**] to introduce a failure detector for aggregation in dynamic peer-to-peer networks.

In the following sections, we discuss the different phases involved in failure detection.

3.4.1. Grouping of nodes

The nodes in the network are arranged into different groups of size g. The nodes within a group are responsible for monitoring each other. The group size should be big enough so that node failures do not cause the group to collapse thereby affecting the detection and small enough such that the message complexity (number of messages and size of data in a message) is low. The strategy for grouping can be decided by the peer-to-peer application. Possible strategies are grouping those nodes that are close to each other, or those which are within the same network.

3.4.2. Sending heartbeat messages

Every $t_{heartbeat}$ time units, every node i sends heartbeat messages to all other nodes in its group. The heartbeat message consists of the following fields:

- Node id - Identifier of node i
- Node contribution - The contribution of node i
- Node estimate - The current estimate of the node i
- Version - The version of the node's estimate
- Suspicion value - The suspicion value that the node i has computed for other nodes of the group. The computation of suspicion value is explained in the next section.

3.4.3. Computing suspicion value

When a node receives a heartbeat message from a node, it stores the information received from the message and also stores the time of arrival of the heartbeat. This is used to compute and store the interarrival times of heartbeats from every other node in the group in a window of size w. Table 3.1 shows the information stored by each node about other nodes in the group.

Every $t_{compute}$ time units, each node computes the suspicion value for every other node in the group. For this, the mean interarrival time of heartbeats for every node in the group is calculated using the recent w interarrival times of heartbeats of that node. Since we take the recent w interarrival times, the changing network conditions are considered in failure detection. Using the mean interarrival time of heartbeats of a node, the number of expected but not-yet-received heartbeats from that node by the current time is calculated. This is done for every node by adding the mean interarrival time of heartbeats of the node to the arrival time of the latest heartbeat from that node.

Each of these not-yet-received heartbeats contributes to the suspicion value. The contribution of the these not-yet-received heartbeats is calculated using a contribution function. The contribution function is a function of the delay in the arrival of the heartbeat. It provides a real value between 0 and 1.

Variable	**Description**	**Fields**
Group	The list of nodes in the group	{ *nodeid, nodeestimate, nodecontribution, version, state, interarrivaltime, lasthbeat, suspvalList* } *nodeid* - Identifier of the node *nodeestimate* - Current estimate of the node *nodecontribution* - Contribution of the node *version* - Version of estimate of the node *state* - State of the node *(ALIVE, DEADFIRST, DEAD)* *interarrivaltime* - Interarrival time of heartbeats from the node *lasthbeat* - Arrival time of the last heartbeat from the node *suspvalList* - List of suspicion values computed by the node and consists of $\{nodeid, suspicionvalue\}$ where *nodeid* - Identifier of the node and *suspicionvalue* - Suspicion value computed about the node

TABLE 3.1. Information stored by each node

The contribution function that we use is :

$$f(t) = \frac{t}{t+2} \tag{3.1}$$

where t = (current time - expected time of arrival of the heartbeat).
Figure 3.4 depicts the contribution function. The suspicion value of a node is calculated as the sum of contributions of all expected heartbeats. The suspicion value calculated using the contribution function is gossiped to all the other nodes in the group by including it in the heartbeat messages. Hence, each node would have the suspicion value that it computed about every other node in the group and also the suspicion values computed by other nodes about every other node in the group.

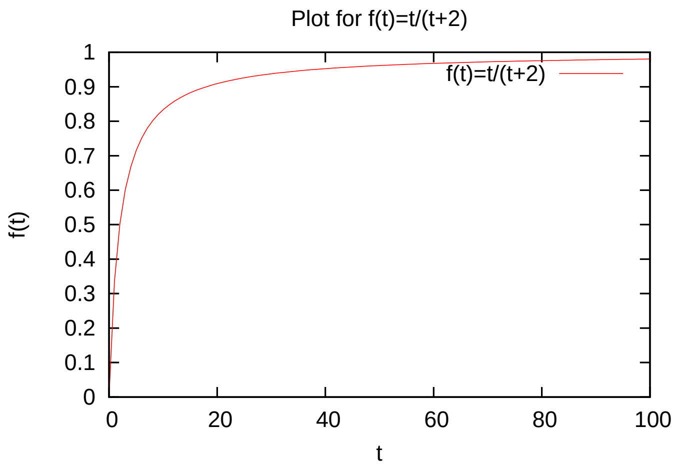

FIGURE 3.3. Contribution function for expected but unarrived heartbeats

For example, consider that node A is computing the suspicion value for node B. Suppose,

The mean of the interarrival times	=	30 seconds
The time when the last heartbeat from B arrived	=	200^{th} second
Current time	=	280^{th} second

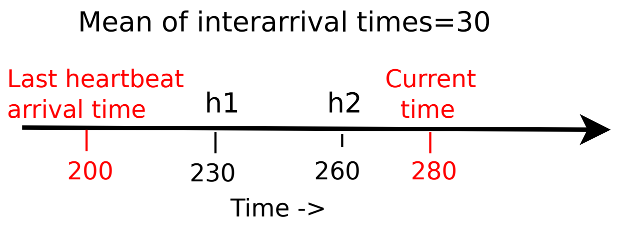

FIGURE 3.4. Example of suspicion value computation

The heartbeats are expected to arrive at time	=	230, 260
Delay in the arrival of the heartbeats	=	(280-230), (280-260)
	=	50, 20
Suspicion value	=	f(50) + f(20)
	=	0.961 + 0.909
	=	1.87

3.4.4. Detecting failures

Applications set a threshold for the suspicion value. Every $t_{cleanup}$ interval, every node i checks the suspicion value that it has computed for each of the other node j and also the suspicion value that it has obtained from other nodes about node j. If all these values has exceeded the threshold set by the application, then node j is declared dead. More about setting the suspicion threshold is discussed in chapter 4.

3.5. Failure Cleanup Algorithm

Once a node failure has been detected, the contribution of that node has to be removed. Also, as a result of gossiping with other nodes, the node would have accumulated some mass from the system. Hence, the cleanup after detection of a node failure would involve the removal of the contribution of that node and restoring the mass that it has taken from the system.

Maintaining the correctness of the system aggregate means conserving the system mass, which is the sum of the contributions of all the nodes.

Let S_a be the sum of the contributions of the nodes 1,2,...,N.

$$S_a = x_1 + x_2 + ... + x_N \tag{3.2}$$

After some rounds of gossiping, the node estimates change to $\overline{x_i}$. Since the system mass is conserved,

$$\overline{x_1} + \overline{x_2} + .. + \overline{x_i} + .. + \overline{x_N} = S_a \tag{3.3}$$

Suppose that node i has failed. So the sum of the estimates of the nodes in the sytem would be S_b given by,

$$S_b = \overline{x_1} + \overline{x_2} + .. + \overline{x_{i-1}} + \overline{x_{i+1}} + .. + \overline{x_N} = S_a - \overline{x_i} \tag{3.4}$$

Since node i has failed, the new sum should be S_c given by,

$$S_c = S_a - x_i = x_1 + x_2 + .. + x_{i-1} + x_{i+1} + .. + x_N \tag{3.5}$$

We need to obtain S_c from S_b. Subtracting 3.4 from 3.5,

$$S_c - S_b = -x_i + \overline{x_i}$$

$$S_c = S_b + \overline{x_i} - x_i \tag{3.6}$$

Hence, if a node i has failed, then the cleanup would involve adding $(\overline{x_i} - x_i)$ to the system mass. In order to maintain the accuracy of the aggregate values :

- The failure cleanup for a node should be performed exactly once.
- The failure cleanup should be done with latest version of that node's estimate.

It may happen that the failure of a node is detected by more than one node. Inorder to remove the conflict, we assume that all nodes have unique identifiers and the algorithm makes sure that among the nodes which detect the failure, the node with the least id is responsible for performing the cleanup steps.

The failure cleanup algorithm is described in Algorithm 3.5. Initially, the state of all the nodes monitored by a node is set to $ALIVE$. When a node i detects that node j has failed, it invokes the function *Cleanup()*. It performs the cleanup steps specified above. It changes the state of the node j to $DEADFIRST$, which means that node j has failed and node i has performed the cleanup steps for it. It then informs all nodes k in the group about the death of node j by sending message $DEAD(k, j, version(j), i)$.

When a node i receives a $DEAD$ message from node k about the death of node j, it checks if the cleanup performed by node k is with the latest version of estimate of node j. If i has a later version, then it sends that to node k through an $UNDO$ message. Next, if this is the first information that node i gets regarding the death of node j, it changes the state of node j to $DEAD$. Else, if node i already had known about the death of node j and if it has already performed the cleanup, then it compares its id with the id of node k. If node k has a lesser id than itself, it undoes the cleanup steps and changes the state of node j to $DEAD$.

When a node i receives an $UNDO$ message, it checks if the version with which it performed cleanup is an earlier version than the version it has received through the $UNDO$ message. If so, it undoes the cleanup steps with earlier version and performs the cleanup with the newer version. The worst-case message complexity for detecting a node failure, with a group size 'g' is $O(g^2)$.

```
function CleanUp($i, j$)
    Choose node $deadnode$ from $Group$ such that $nodeid(deadnode) = j$
    $\overline{x_i} = \overline{x_i} + nodeestimate(deadnode) - nodecontribution(deadnode)$
    $state(deadnode) = DEADFIRST$
    for all node $k \in Group$ do
        DEAD($k, j, version(j), i$)
    end for
end function

function DEAD($i, j, version, k$)
    Choose node $deadnode$ from $Group$ such that $nodeid(deadnode) = j$
    if $version(deadnode) > version$ then
        UNDO($k, j, version(deadnode), nodeestimate(deadnode), i$)
    end if
    if $state(deadnode) = ALIVE$ then
        $state(deadnode) = DEAD$
    else if $state(deadnode) = DEADFIRST$ then
        if $i > k$ then
            $\overline{x_i} = \overline{x_i} - nodeestimate(deadnode) + nodecontribution(deadnode)$
            $state(deadnode) = DEAD$
        end if
    end if
end function

function UNDO($i, j, version, value, k$)
    if $state = DEADFIRST$ then
        Choose node $deadnode$ from $Group$ such that $nodeid(deadnode) = j$
        if $version(deadnode) < version$ then
            $\overline{x_i} = \overline{x_i} - nodeestimate(deadnode) + value$
            $version(deadnode) = version$
            $nodeestimate(deadnode) = value$
        end if
    end if
end function
```

Figure 3.5. Failure Cleanup Algorithm

CHAPTER 4

Evaluation

We performed simulations using PeerEmu, which is a customized version of PeerSim [**1**], that is exclusively event-driven.

We used the Cyclon protocol [**11**] for the creation and maintenance of the overlay network. The default contributions of the nodes used for the simulations are chosen from a uniform distribution, where an integer value is chosen randomly from the range 0 to $(2^{20} - 1)$. For simulations that compare different initial value distributions we have used the following distributions:

- Uniform distribution, where an integer value is chosen randomly from the range 0 to $(2^{20} - 1)$
- Poisson distribution with λ=1000
- Non-uniform distribution, where a value is chosen from the range 0 to 5000 with 0.2 probability and from the range 100000 and 105000 with 0.8 probability

In order to simulate the delay in messages, a trace of message delays from the real world was used. For each pair of nodes, a value from this trace is chosen and used to simulate the delay in the delivery of messages between them.

The default values for parameters are :

- Group size of nodes = 8
- Suspicion threshold = 1
- 1 aggregation round = 1 second
- Robustness protocol execution round = $0.5 \times$ aggregation round = 500 ms
- Percentage of message loss = 20

- Period of sending heartbeats ($t_{heartbeat}$) = 0.5 × aggregation round = 500 ms
- Period of calculating suspicion values ($t_{compute}$) = 0.5 × aggregation round = 500 ms
- Period for checking for failed nodes ($t_{cleanup}$) = 0.5 × aggregation round = 500 ms

Grouping of nodes was performed using a simple round-robin strategy. Node with identifier i was assigned the group ($i\%N$) where N is the network size. To simulate node churn, $x\%$ of nodes were added and removed from the system every aggregation round.

4.1. Evaluation Metrics

The following are the metrics that we have used for the evaluation.

(1) The standard deviation of the node estimates are plotted against time/rounds. The standard deviation describes the spread of the node estimates from their average. Hence, the lesser the standard deviation, the lesser are the spread of the node estimates from their average, which means they are closer to their average.
(2) Convergence ratio is defined as the ratio of the variance of node estimates in the current aggregation round to the variance of node estimates in the previous aggregation round. Convergence ratio is plotted against time/rounds. Since, the variance of the node estimates gives an indication of the spread of the node estimates from their average, the ratio of consecutive variances of node estimates gives an indication of the rate of convergence, which is the rate at which the node estimates reach the system average.
(3) We define the span of a set of values as the difference between the maximum and minimum of the values. The converging round is the round when the value span of the node estimates reach a threshold times the span of the initial node contributions. The threshold used for our experiments is (1/million). The converging round gives an indication about how long it takes for the nodes to converge towards the average.

4.2. Stable Peer-to-Peer Networks

Here we describe the performance of the aggregation protocol in stable peer-to-peer networks.

In Figure 4.1, we have plotted the standard deviation of the node estimates against rounds, for varying network sizes. We observe that the standard deviation of the node estimates decreases uniformly with time, indicating that the node estimates become closer to the average with gossiping.

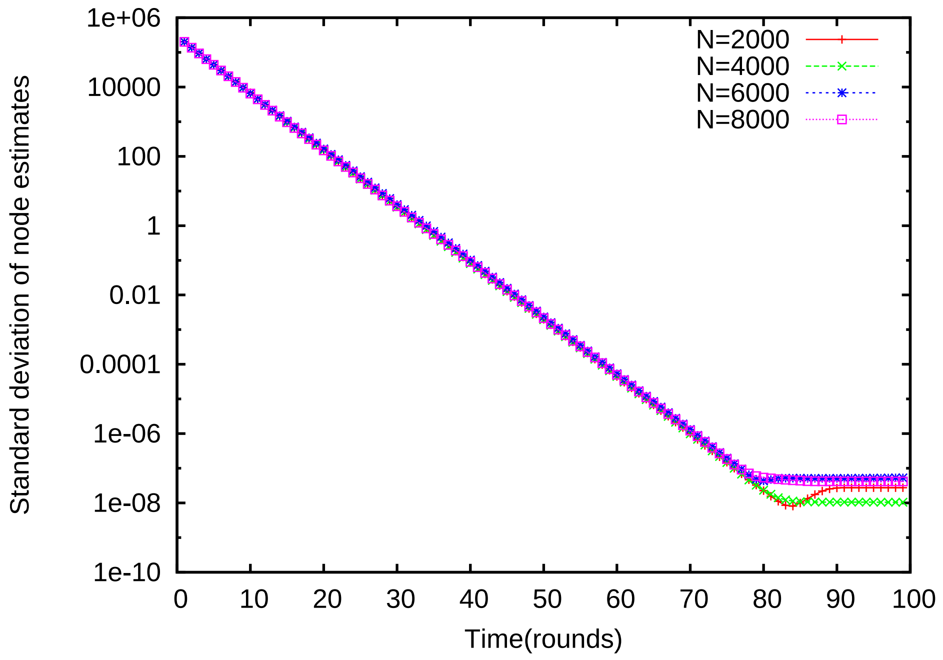

FIGURE 4.1. Standard deviation of node estimates vs time for varying network sizes

In Figure 4.2, we have plotted the convergence ratio against rounds for varying network sizes. We observe that the rate of convergence of node estimates is almost constant and doesn't depend on the network size.

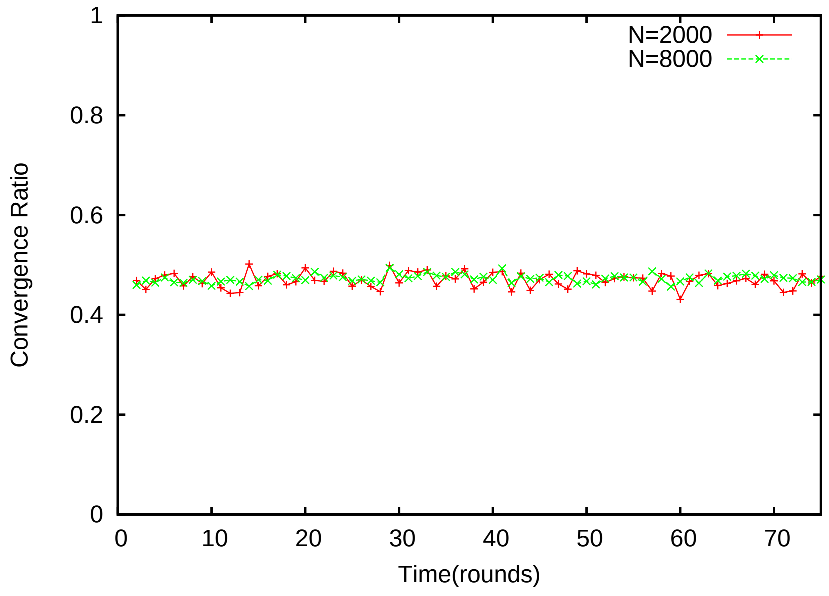

FIGURE 4.2. Convergence ratio vs time for varying network sizes

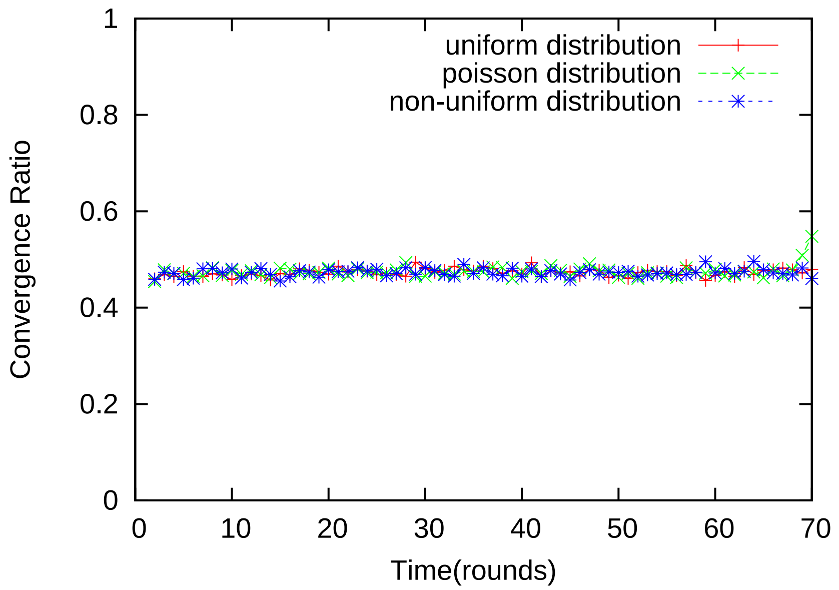

FIGURE 4.3. Convergence ratio vs time for varying initial value distributions

In Figure 4.3, we have plotted the convergence ratio for varying initial value distributions. We observe that the rate of convergence of node estimates is similar for various initial value distributions.

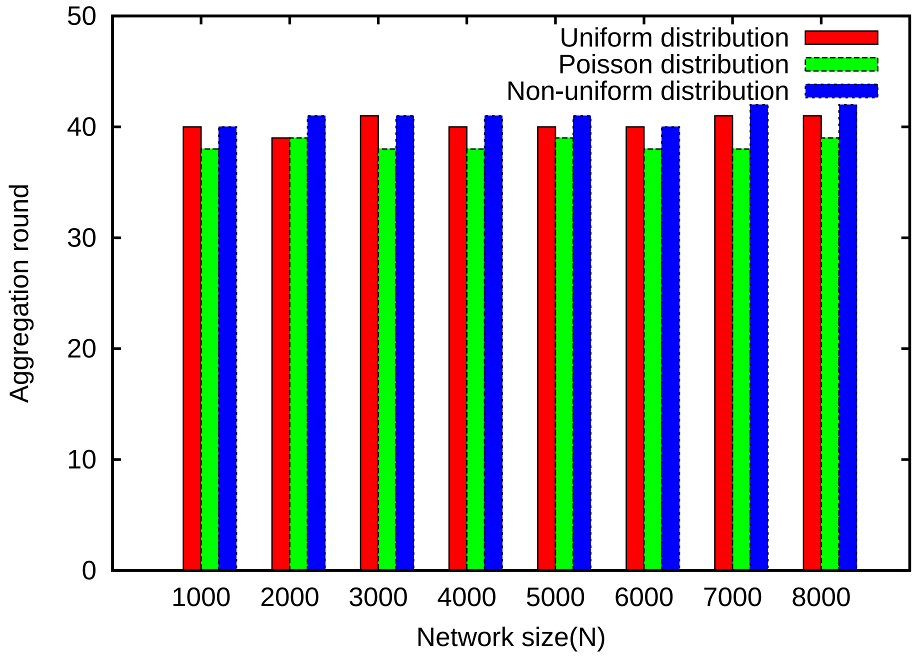

FIGURE 4.4. Converging round for varying network sizes and varying initial value distributions

In Figure 4.4, the converging round for various network sizes and initial value distributions is plotted, and we see that the convergence is independent of network size and initial value distributions.

From our evaluations, we conclude that the convergence of node estimates towards the system average in a stable peer-to-peer network is independent of the network size and the initial value distributions.

4.3. Dynamic Peer-to-Peer Networks

In order to study the effect of node failures and node churn on aggregation, we performed two sets of experiments.

- A percentage of nodes are removed from the network after the nodes have converged to the system average.
- In each aggregation round, x% of node churn is simulated.

In Figure 4.5, the convergence ratio for a stable network of size 8000 and for a network with churn of 0.001% and 0.0005% is shown. We observe that the convergence ratio constantly changes during churn.

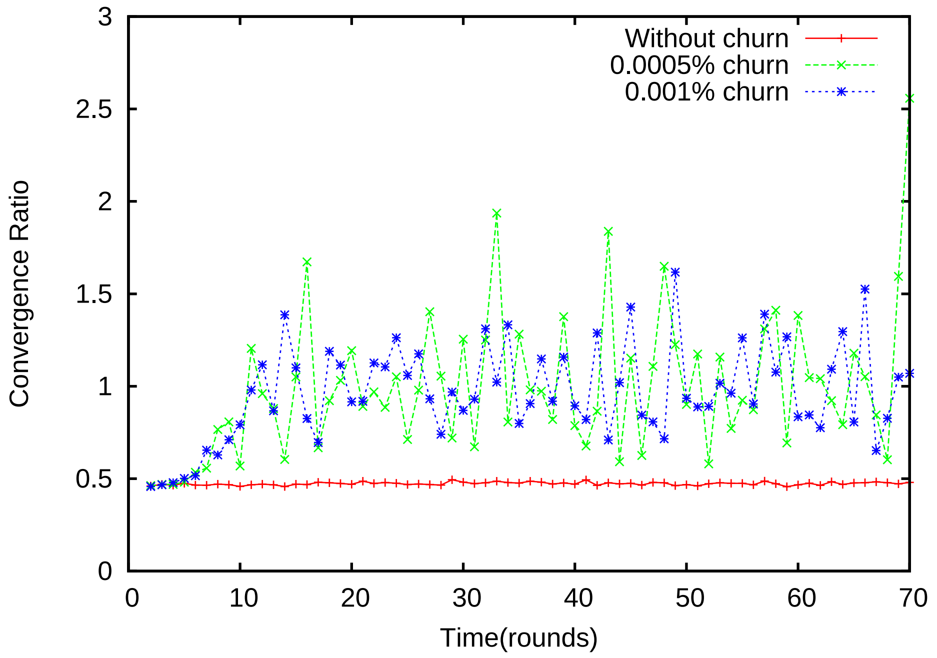

FIGURE 4.5. Convergence ratio with and without churn for a network size of 8000

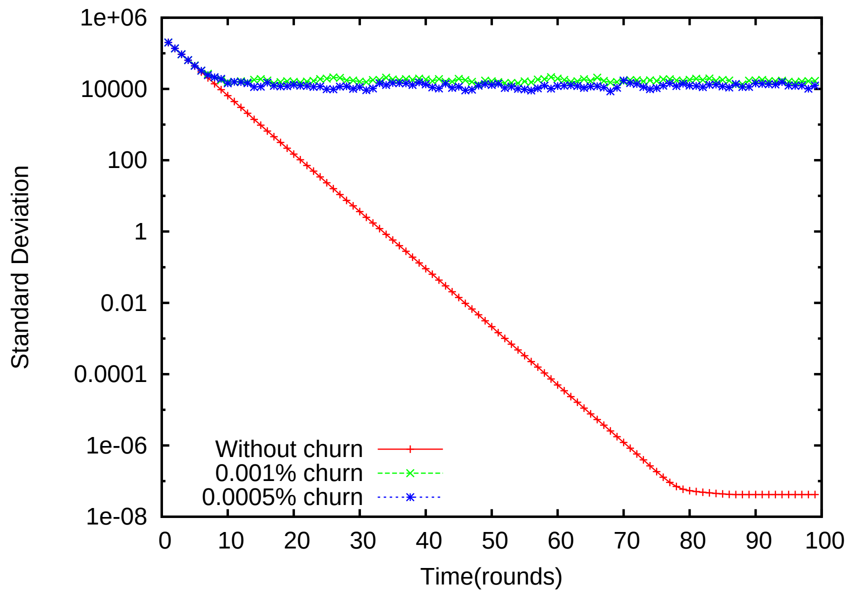

FIGURE 4.6. Standard deviation of node estimates with and without churn for a network size of 8000

In Figure 4.6, the standard deviation of node estimates for a stable network and for a network with churn is shown. The standard deviation of node estimates for a stable network decreases with time and stays constant, showing the convergence of the node estimates towards the system aggregate. The standard deviation of node estimates in a network with churn, doesn't decrease, but remains almost the same throughout, showing the existence of error of the node estimates from the system average and hence, the lack of convergence of the node estimates to the system average.

4.4. Failure Detection

In this section, we analyze three characteristics of our robustness protocol.

- Failure detection time which is the time taken to detect the node failures.
- Number of false positives, which is the number of wrong failure detections.
- Performance of the failure detection and failure cleanup algorithm given by the error of the converged average from the true system average.

4.4.1. Failure Detection Time

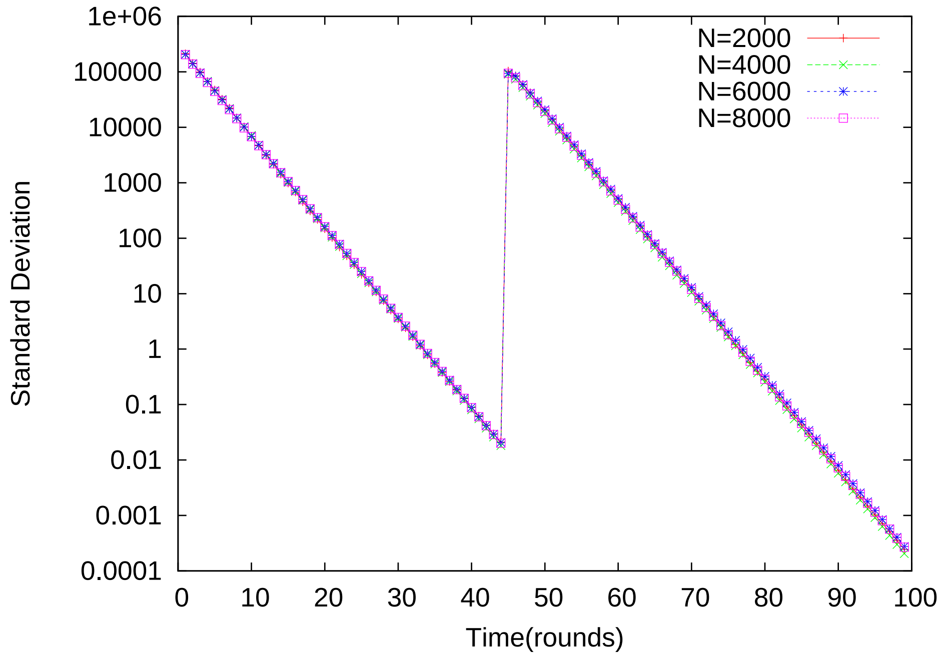

FIGURE 4.7. Standard deviation of node estimates for varying network sizes, when 10% of nodes are removed after convergence

In Figure 4.7, the standard deviation of node estimates are plotted, when 10% of nodes are removed after convergence. The plot for different network sizes is shown. The transition of the standard deviation to a higher value shows the time when the failure of nodes is detected. In Figure 4.7, the transition happens at the same time irrespectively of the network size, which means that the failure detection time doesn't depend upon the network size.

In Figure 4.8, the standard deviation of node estimates for a network of size 8000 is plotted, when varying percentage of nodes are removed after convergence. From Figure 4.8, we observe that the transitions of the standard deviations from lower to higher value occurs at the same time, irrespectively of the percentage of nodes removed. This shows that the failure detection time is unaffected by the number of nodes removed.

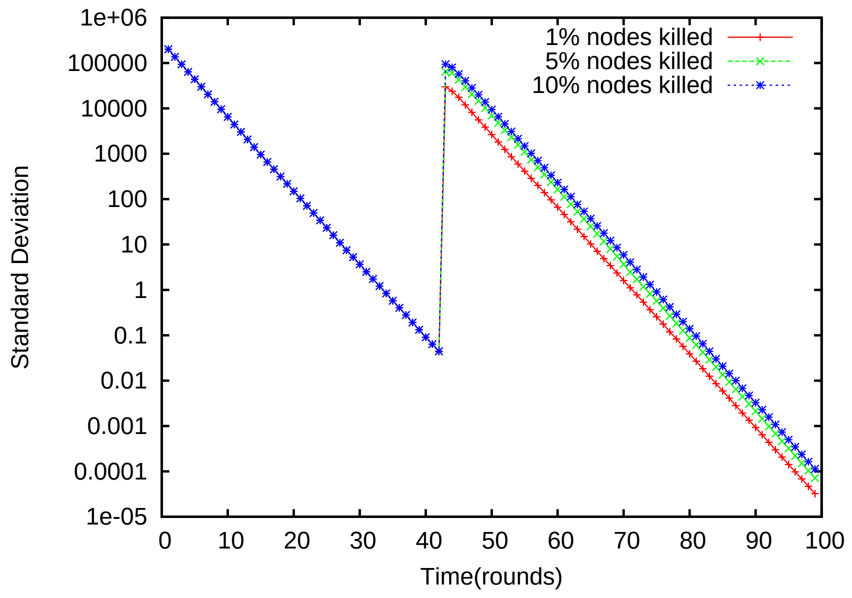

FIGURE 4.8. Standard deviation of node estimates when varying percentage of nodes are removed after convergence for a network size of 8000

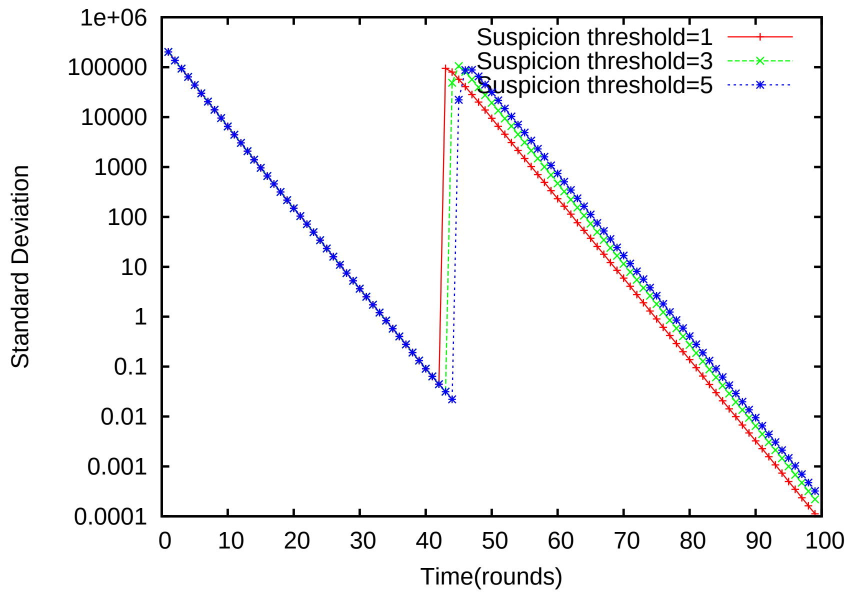

FIGURE 4.9. Standard deviation of node estimates for varying suspicion thresholds, when 10% of nodes are removed after convergence

In Figure 4.9, the standard deviation of node estimates are plotted for a network of size 8000, for varying suspicion thresholds, when 10% of nodes are removed after convergence. From Figure 4.9, we observe that the lower the suspicion threshold, the quicker the failure detection. Hence, we conclude that the failure detection time depends upon the suspicion threshold.

4.4.2. False positives

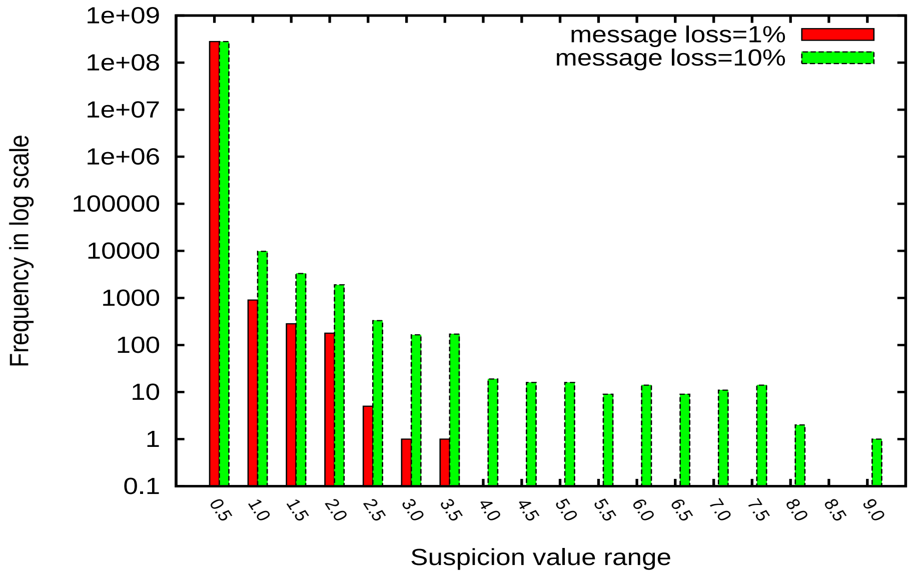

FIGURE 4.10. Suspicion value distribution for varying message losses, group size=8

In Figure 4.10, the suspicion value distribution for a network of size 8000 is plotted, with a group size of 8. The suspicion value of all the nodes for a period of 5000 rounds is used for the plot. Here, we observe that higher the suspicion threshold, lesser the number of false positives. For example, if the suspicion threshold is set to 1 for a network with message loss 1%, then the percentage of false positives is 1.6×10^{-6}. While when the suspicion threshold is set to 0.5, the percentage of false positives is 4.8×10^{-6}. Also, we observe that the suspicion value for a network with higher message loss spans a broader range compared to a network with lower message loss. For instance, the percentage of false positives when the suspicion threshold is set to 1 is 1.6×10^{-6} for a network with message loss 1% while compared to 2.1×10^{-5}

for a network with message loss 10%. Hence, the suspicion threshold for a network with lower message loss can be lower than that of a network with higher message loss.

4.4.3. Performance of failure detection and cleanup

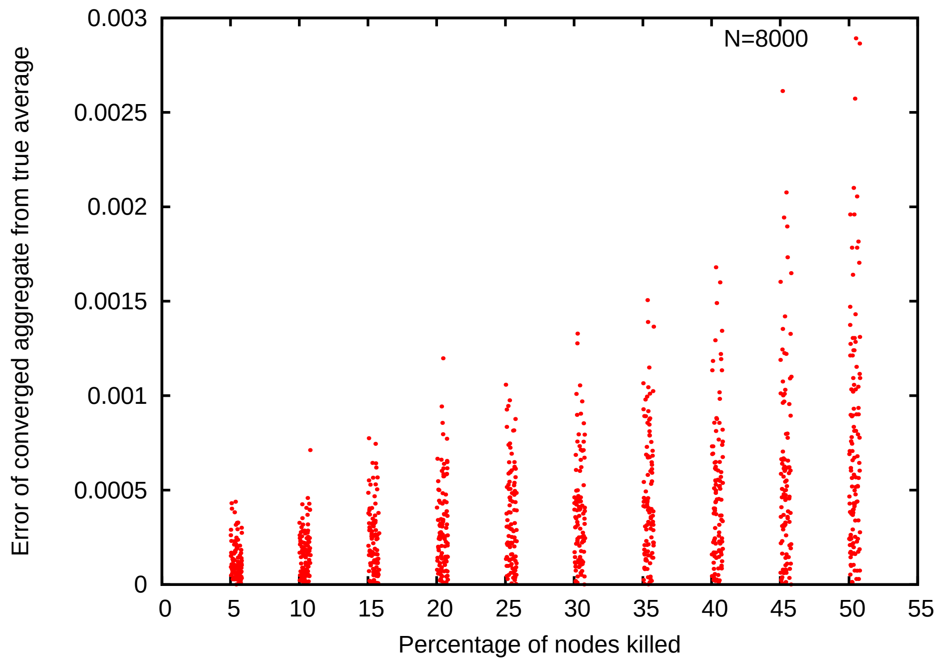

FIGURE 4.11. The error of converged average from true system average, when varying percentage of nodes are killed after convergence, group size=8

In Figure 4.11, we compute the difference of the converged average from true system average, when varying percentage of nodes are killed after convergence for a network size of 8000 and group size 8. For each set, we performed 100 experiments. Firstly, we observe that, in the worst case when half of the nodes are killed, the error observed is as low as 0.0028, while when lesser number of nodes are killed, the error is even lower, for 10% node failures, the error is 0.0007. Here, we observe that when the percentage of nodes killed are more, the error is more. This is because when more number of nodes are killed, a group itself can collapse and hence the failure cleanup for those nodes is not performed, leading to errors.

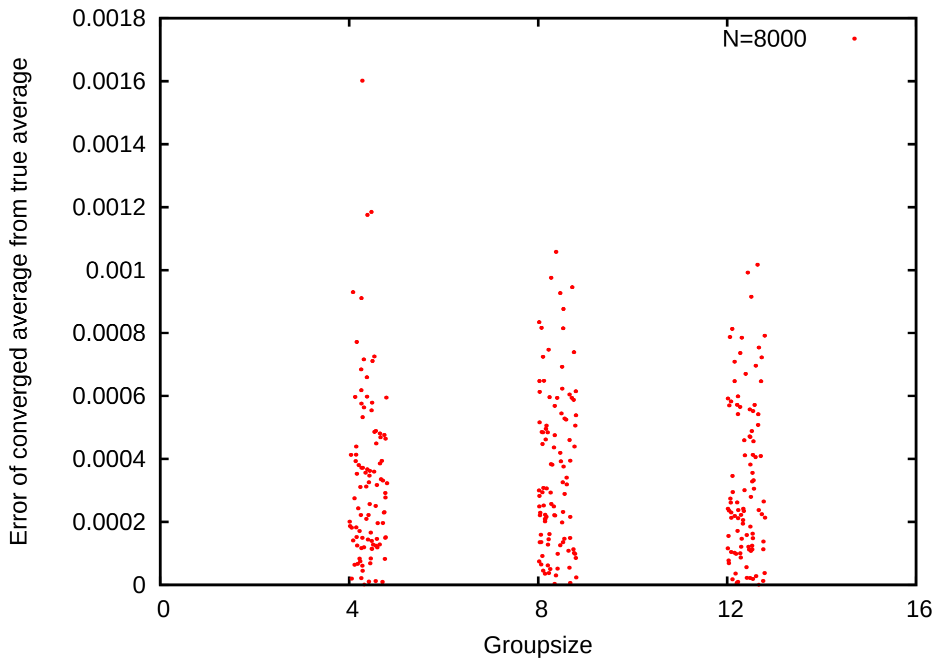

Figure 4.12. The error of converged average from true system average, when 25% of nodes are killed after convergence, for different group sizes

In Figure 4.12, the difference of the converged average from true system average is plotted for varying group sizes, when 25% of nodes are killed after convergence. For each group size, we performed 100 experiments. We observe that for group size 3, the error can reach a higher value than for group sizes 8 and 12. This is because when a large number of nodes of the network, it may happen that all the nodes of a group of smaller size is killed. This leads to lack of failure cleanup and error in converged average.

4.5. Inferences

In case of stable peer-to-peer networks, the convergence of the node estimates towards the system average is unaffected by the network size or the initial value distribution. For dynamic peer-to-peer networks with node failures and node churn, the convergence is affected by the percentage of node churn, which is the percentage of nodes added and removed from the system. There is a percentage of error that exists in the system in case of node churn and hence the convergence of the node estimates towards the system average is hindered due to node churn.

Regarding the performance of the failure detection algorithm, we saw that the failure detection time is unaffected by the network size and the number of nodes failed. The failure detection time depends upon the suspicion threshold. The lesser the suspicion threshold, faster the node failures can be detected. But, in that case, the number of false positives can increase. Hence, there is a tradeoff between the failure detection time and the number of false positives. The application should decide the suspicion threshold, according to its quality of service requirements like, the failure detection time and the percentage of false positives acceptable. Also, the suspicion value distribution depends upon the message loss in the network and the group size, and hence the suspicion threshold should be determined according to the nature of the network.

In addition, we saw that the correctness of the system average is affected when the number of nodes failed is more. A smaller group leads to more errors, due to the chance of the collapse of the group as a whole.

CHAPTER 5

Confidence Value

In this chapter, we answer the following questions :

- What are the factors that determine the accuracy of the node estimate?
- How can the accuracy of the node estimate be determined?

As aggregation progresses, the node estimates of the different nodes change. The node estimates also change when interacting with newly arrived nodes and when failure cleanup is performed. The changes in node estimates affect the accuracy of the node estimate, which is defined as the difference of the node estimate from the system average. Our goal was to explore the factors that affect the accuracy of node estimates and to determine the accuracy of the node estimates during aggregation.

5.1. Observations and Inferences

From the evaluations described in the previous chapter, we have seen that in stable peer-to-peer networks, the rate of convergence of node estimates is unaffected by the network size and initial value distribution. The decrease in the standard deviation shows the decrease in the error of the node estimates from the average. Hence, the more the nodes gossip, the lesser their error from average and hence, more accurate the values become. But in cases of node churn, we have observed that the standard deviation of values doesn't decrease uniformly. These observations lead us to the inference that the accuracy of node estimate depends upon how long the node has gossiped and also on whom it has gossiped to.

5.2. Confidence Value

The above inferences led us to define a metric which we call the ‘confidence value’ which gives an indication of the accuracy of the node estimate. The confidence value is calculated using the algorithm presented in 5.1. A node i which has just joined the system has a confidence value of 0 (*Join*(i)). When a node i gossips with another node j (*Gossip*(i, j)) having a confidence value that is greater than or equal to its confidence value, then the confidence value of the node increments by 1. When a node i gossips with node j with lesser confidence value, then the confidence value of node i becomes confidence value of node j incremented by one. During failure cleanups (*Cleanup*(i)) and contribution updates (*ContributionUpdate*(i)), the confidence value of node i becomes zero.

```
function JOIN($i$)
    $confidence(i) = 0$
end function

function GOSSIP($i, j$)
    if $confidence(i) \leq confidence(j)$ then
        $confidence(i) = confidence(i) + 1$
    else
        $confidence(i) = confidence(j) + 1$
    end if
end function

function CONTRIBUTIONUPDATE($i$)
    $confidence(i) = 0$
end function

function CLEANUP($i$)
    $confidence(i) = 0$
end function
```

FIGURE 5.1. Algorithm to compute confidence value

5.3. Evaluations

Here we show the evaluations performed using confidence value. In Figure 5.2, the percentage error of the node estimates from the system average is plotted against confidence value for network size of 1000 and 8000. We observe that with each confidence value, a range of percentage error of node estimates is associated. The higher the confidence value, the lesser the percentage error of node estimates. In addition, the relationship between percentage error and confidence value is independent of the network size.

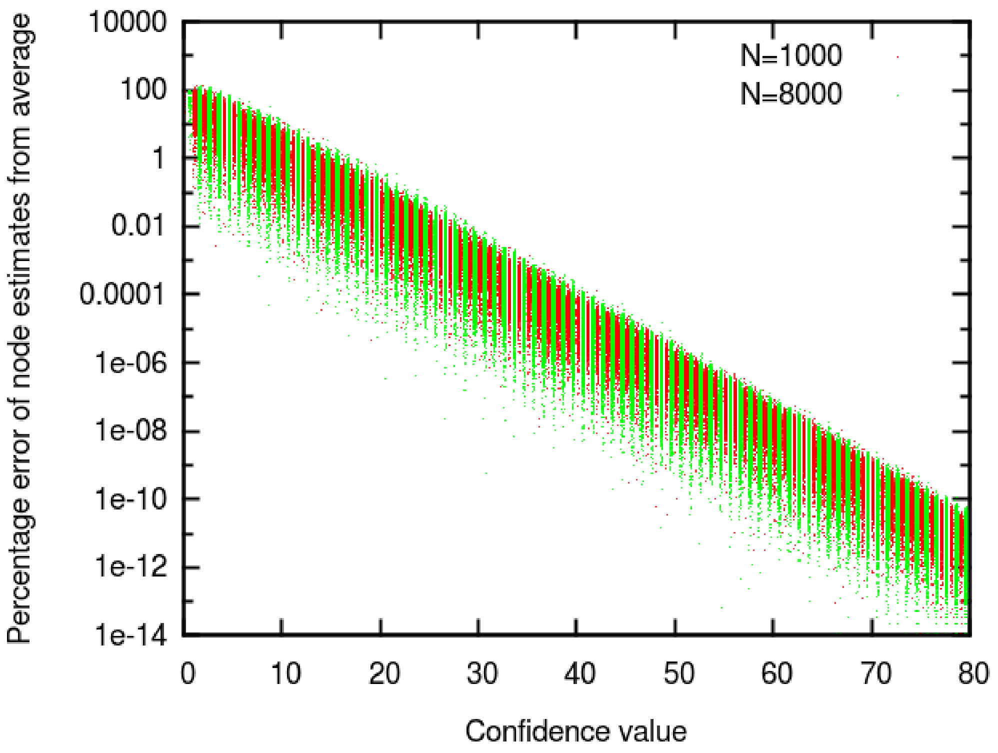

Figure 5.2. Percentage error of node estimates from system average vs confidence value for varying network sizes

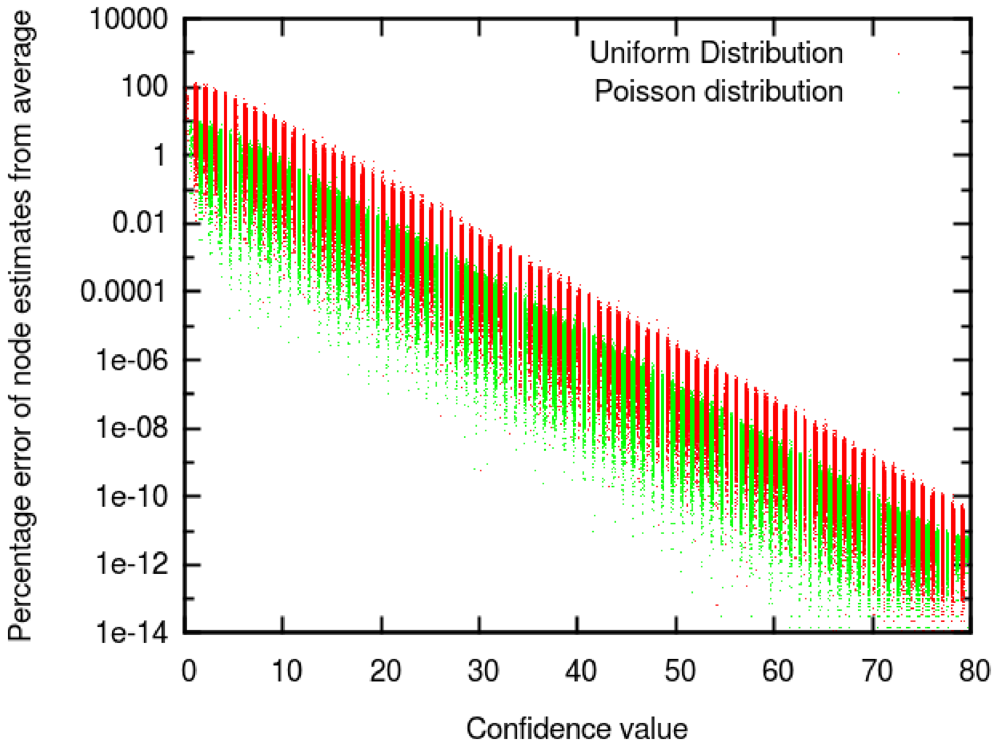

FIGURE 5.3. Percentage error of node estimates from system average vs confidence value for varying initial value distributions

In Figure 5.3, the percentage error of the node estimates from the system average is plotted against confidence value for a network size of 1000, for uniform distribution and poisson distribution. We observe that the percentage error of node estimates corresponding to a confidence value is different for different initial value distributions. Hence, the relationship between percentage error and confidence value is dependent on the initial value distribution.

From Figure 5.4 and Figure 5.5, we observe that the relationship between confidence value and the percentage error of node estimates is different for networks with different churn. Also, we can see that there is an upper limit of confidence value in each case, which shows the lower bound of percentage error of node estimates. This shows the error that would exist in a network with a particular value of churn. Convergence of node estimates beyond this error is not possible.

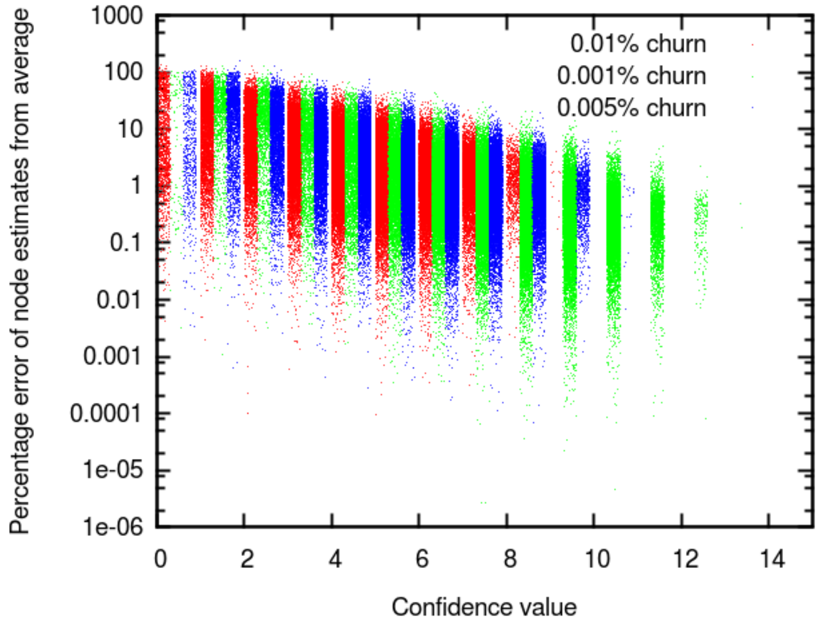

FIGURE 5.4. Percentage error of node estimates from system average vs confidence value for varying node churn

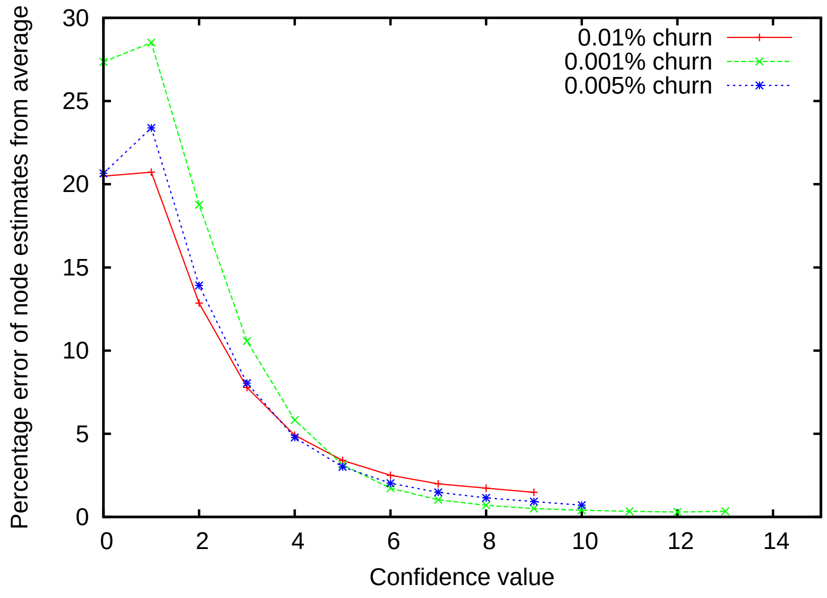

FIGURE 5.5. Average of Percentage error of node estimates from system average vs confidence value for varying node churn

5.4. Inferences

For a given network condition, with a specific percentage of node churn and a specific initial value distribution, the confidence value of a node indicates the accuracy of the node estimate, that is, the error of the node estimate from the system average. Also, given a network with a specific percentage of node churn, the maximum attainable confidence value provides the maximum possible accuracy of node estimate that can be achieved. This gives a lower bound of the error of node estimates that exists in the network.

CHAPTER 6

Conclusion

We studied the effect of node failures and node churn during aggregation in peer-to-peer networks. We introduced a robustness protocol that is scalable and adaptive. Our protocol detects node failures and performs the cleanup steps after node failures, such that the correctness of the system aggregate is maintained. The failure detection performs quite well producing an error of converged average as low as 0.0028 for a network size of 8000. Our failure detection mechanism detects the failure of nodes at any time during aggregation and reflects the changes in the networks as quickly as possible, enabling in faster convergence of node contributions towards the system average. The failure detection mechanism can ensure very low false suspicion rate, with a tradeoff in the failure detection time. The failure detection time is unaffected by the network size and the number of nodes failed. The application can set the suspicion threshold according to its quality of service requirements, such as the failure detection time and the number of false suspicions that it allows.

We introduced the confidence value which gives an indication of the accuracy of the node estimates. The relationship between the confidence value and the accuracy of the node estimates depends on the node churn in the system and the initial value distribution. The maximum confidence value that is possible in a particular network setting, gives an indication of the maximum achievable accuracy of the node estimates in the system.

The future work in this direction would be to introduce a grouping strategy for grouping the nodes. The strategy should take care of the collapse of node groups due to failures and create more groups when a group size exceeds the maximum allowed size. The strategy should provide for the creation, maintenance and restructuring of node groups. We saw that the error of converged average increases with the number of nodes that fail. The more the failed nodes, the more the error. It is expected that such a strategy would help to keep the error of converged average constant, irrespectively of the number of nodes that fail and hence improve the performance of failure detection and cleanup.

Bibliography

[1] Peersim simulator. `http://peersim.sourceforge.net/`.

[2] L. Andrei, C. Dobre, F. Pop, and V. Cristea. A failure detection system for large scale distributed systems. In *2010 International Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2010)*, pages 482–489, February 2010.

[3] Mayank Bawa, Hector Garcia-Molina, Aristides Gionis, and Rajeev Motwani. Estimating aggregates on a peer-to-peer network. Technical Report 2003-24, Stanford InfoLab, April 2003.

[4] Adriana Cristina Draghici. Peer to peer aggregation on dynamic networks. Master's thesis, Vrije Universiteit, Amsterdam, 2011.

[5] Naohiro Hayashibara, Xavier Defago, and Takuya Katayama. Flexible failure detection with k-fd. Technical Report IS-RR-2004-006, School of Information Science, Japan Advanced Institute of Science and Technology, February 2004.

[6] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(3):219–252, August 2005.

[7] Srinivas Kashyap, Supratim Deb, K. V. M. Naidu, Rajeev Rastogi, and Anand Srinivasan. Efficient gossip-based aggregate computation. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '06, pages 308–317, June 2006.

[8] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '03, page 482, October 2003.

[9] Mortada Mehyar, Demetri Spanos, John Pongsajapan, Steven H. Low, and Richard M. Murray. Asynchronous distributed averaging on communication networks. *IEEE/ACM Trans. Netw.*, 15(3):512–520, June 2007.

[10] Dinesh C. Verma. *Legitimate Applications of Peer-to-Peer Networks.* Wiley-Interscience, 2004.

[11] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Springer Journal of Network and Systems Management*, 13(2):197–217, June 2005.

[12] Fetahi Wuhib, Mads Dam, and Rolf Stadler. Robust monitoring of network-wide aggregates through gossiping. In *Proc. Tenth IFIP/IEEE International Symposium on Integrated Management (IM 2007)*, pages 21–25, May 2007.