ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA

SCUOLA DI SCIENZE

Corso di Laurea Triennale in Scienze e Tecnologie Informatiche

DATA DISSEMINATION OVER COMPLEX NETWORKS THROUGH GOSSIP

Tesi di Laurea in Reti di Calcolatori

Relatore:
GABRIELE D'ANGELO

Presentata da:
GIANMARCO BERTI

Sessione III
Anno Accademico 2013-2014

"*Admitting we are not heroic*

is when we are the most heroic of all."

J.D. - Scrubs

Prefazione

La continua espansione di internet e dei dispositive elettronici utilizzati ha sollevato la necessità di adottare sistemi di diffusione dell'informazioni diversi dal classico paradigma client-server, in favore del paradigma peer-to-peer, che negli ultimi anni ha suscitato notevole interesse anche grazie a software di successo quali Napster, Torrent, Skype e molti altri, che hanno contribuito a cambiare le regole della diffusione delle informazioni. In particolare, questa tesi tratta lo studio di algoritmi per diffusione di informazioni tra un gruppo di agenti caratterizzati da risorse limitate, come una rete di sensori o una di router che necessitino di scambiare informazioni di controllo o di routing.

In particolare, in questa tesi è analizzata una famiglia di algoritmi detta Gossip; questi algoritmi sono stati ispirati dal modo in cui un'informazione, quale ad esempio un pettegolezzo, si diffonde tra la popolazione. Questa famiglia di protocolli, che cerca di ridurre i messaggi inviati sfruttando una logica generalmente probabilistica, ha dimostrato robustezza, flessibilità, semplicità e efficienza, suscitando un notevole interesse tra i ricercatori.

A seguito di una iniziale introduzione, costituita dall'analisi dei protocolli conosciuti in letteratura, viene poi preso spunto per la creazione di nuovi algoritmi per la diffusione di informazioni. Le sezioni successive tratteranno l'analisi e il confronto dei nuovi protocolli implementati con quelli noti in letteratura, da cui verranno tratte le conclusioni in merito.

List of figures

Index

Chapter 1

Introduction

The continuous expansion of Internet and electronic devices has led to a diversification of methodologies to spread information, and after the success of peer to peer protocols as alternative to client-server, many algorithms have been studied and evolved, like Torrent, Skype and Napster, becoming quickly a landmark for new protocols. Some are proprietary, others are open source, but they all have something in common: spreading information efficiently.

Making nodes reach a state of agreement is a fundamental problem in decentralized networked systems [1], mostly for what concerns sensor networks, peer-to-peer (P2P) networks, mobile networks of vehicles and social networks. Those structures strongly differs from the telephone network or the Internet, not only because they are not engineered to provide efficient communication between various entities, but also because they lack infrastructures, exhibit unpredictable dynamics and face stringent resource constraints [2]. Their communication is often intense but light between an unknown number of nodes, where each one only knows its neighbors and has no idea of the topology that may change too frequently to relay on common algorithms like Routing Information Protocol (RIP) and Open Shortest Path First (OPSF) [3]; however, they still require

algorithms for communication or merely spreading information. The unpredictability behavior of those networks combined with their limited resources have excluded client-server approach. Although it is true that the latter paradigm is capable of spreading information efficiently with an appropriate amount of resources, the variable number of nodes in the network would lead to allocate excesses resources without the guarantee of an acceptable scalability. For the reasons listed above, client-server has been sidelined in favor of P2P, which is much more suitable to operate with limited resources in strongly dynamic networks due to its decentralized nature and also because it transforms clients into peers, which does not only consume resources to obtain information they need, but also contributes actively with their own resources to the spreading.

Many different families of protocols have been studied and amongst all the interest has raised for gossip algorithms that have been subject of intense research [4]. In particular, Gossip-Epidemic algorithms have shown robustness, flexibility, simplicity and efficiency in spreading information [5], making them particularly appropriate. Due to their probabilistic nature Gossip-Epidemic algorithms lack reliability: this requires expensive mechanisms to be implemented and give the possibility to detect missing messages and initiate retransmission, and causes the algorithm not to scale over a couple of hundred processes [6].

Many different algorithms have been proposed, unfortunately none of them has shown satisfying results, leaving the debate still open; this is due to the fact that those algorithms are capable of making nodes reach a state of agreement but their efficiency is not satisfying enough.

We are looking for an algorithm capable of spreading information on a network of unknown topology. It should have the highest coverage and the lowest overhead and delay possible. Appreciable characteristics include

topology independence, scalability, adaptability and fault tolerance. Even though an algorithm studied for a specific network may grant better results, topology independence and adaptability are necessary to provide a generic algorithm capable of spreading information in different scenarios. A possible approach to achieve independence is to avoid considering nodes position in the graph, as explained in following sections.

Chapter 2

Gossip algorithms

Gossip algorithms [7], as the name suggests, are built upon a gossip or rumor style and offer many different methods to archive agreement in a group of dynamic agents. They are the evolution of the trivial but inefficient pure flooding protocol, where every node simply forwards every message received to all its neighbors. This solution will lead to broadcast storm problem [8]. Gossip algorithms differs from flooding because after each reception the node evaluates, usually with a probabilistic *threshold*, if forward the message or not. The introduction of random events is a solution adopted by other protocols, as quicksort and many other algorithms do [9], not only because the application context is not always predictable, but also because the random events tend to balance. A notable approach to gossip inspired by the behavior of a spreading disease in epidemiology are epidemic protocols. These protocols set a dynamic state for each node, depending on its current knowledge of a specific message:

- Susceptible (S): the node is not aware of the message;
- Infected (I): the node knows the existence of the message and it is currently spreading the information.

- Removed (R): the node is aware of the existence of the message but is not contributing to its spreading.

The two different models created [10] using these conditions will be briefly explained below.

2.1 SI

In the SI model, only two states are used, indeed a node can only be Infected or Susceptible, but once infected the node cannot vary its state anymore.

Algorithm 1: SI gossip

```
 1: loop                                  11: procedure OnUpdate(m)
 2:    wait(Δ)                            12:    store m.update
 3:    p ← random peer                    13: end procedure
 4:    if push and in state I then        14:
 5:       send update to p                15: procedure OnUpdateRequest(m)
 6:    end if                             16:    if in state I then
 7:    if pull then                       17:       send update to m.sender
 8:       send update-request to p        18:    end if
 9:    end if                             19: end procedure
10: end loop
```

The code is composed by a running thread (lines 1 – 10) executed every Δ time units. In every iteration a random peer is chosen and, depending on the current configuration, the algorithm may be push, pull or push-pull, acting consequently. When an update is received, the message is stored and the node changes its state to Infected (line 12).
If a neighbor has requested an update and if the current state is Infected (line 16), an update is sent (line 17).

In case of push algorithms, infected nodes actively spread information, while susceptible nodes are passive. On the contrary, in the in push-pull all nodes are always active.

2.2 SIR

SI model has no termination condition; to achieve it, a third state has been implemented: Removed.

Algorithm 2: SIR gossip

```
 1: loop
 2:    wait(Δ)
 3:    p ← random peer
 4:    if push and in state I then
 5:       send update to p
 6:    end if
 7:    if pull then
 8:       send update-request to p
 9:    end if
10: end loop
11:
12: procedure OnFeedBack (m)
13:    switch to R with prob. 1/k
14: end procedure
15: procedure OnUpdate(m)
16:    if in state I or R then
17:       send feedback to m.sender
18:    else
19:       store m.update
20:    end if
21: end procedure
22:
23: procedure OnUpdateRequest(m)
24:    if in state I then
25:       send update to m.sender
26:    end if
27: end procedure
```

The protocol differs from SI for few but important lines. At the first reception the node switch state to Infected (line 19): from now on, any new message will trigger a feedback to the sender (line 17). When a node receives a feedback has *1/k* probability to switch to Removed state (line 13). Once in Removed state, the node will not answer to update

requests (line 24) but will continue to send feedbacks. The SIR gossip is strongly influenced by the value *k*.

2.3 Push

The base idea behind push algorithm is that anytime a node has a new piece of information, it forwards to some or all its neighbors, who continue to forward the message if the forwarding condition is satisfied, otherwise the message is dropped. A termination condition is used to prevent the message travelling forever in the network, which may cause higher overhead. Usually the termination condition is the Time-To-Live (TTL) of the message, which is decreased at each hop, but nodes can also be programmed to drop messages that have already been received. In a push protocol, when the first message is created, only the node who generated it know about its existence, so the probability to forward it to nodes that still have to receive it is higher at the beginning, but every time it is forwarded, chances to reach an unconscious node decrease at each hop through the network.

2.4 Pull

Pull protocols have the opposite approach: after a given time, the node asks some or all its neighbors if new information is available, and if the answer is positive, the node will be informed. In contraposition with push protocols, pull tends to be more efficient in later rounds [11], while the chances to find information in the beginning are much lower.

2.5 Hybrid

For the reasons explained above, some protocols try to mix push and pull [11], aiming to collect the advantages of both and avoiding their disadvantages. Unfortunately, this family of protocols tends to be more complex not only because their efficiency is based upon the correct time when switch from push to pull, but they also require nodes to be aware of which messages they have already received. This is not a simple problem, because it requires memory for the node to store the identifier (ID) of each message, which leads to another problem: knowing when to remove an ID from the cache; the complexity is not given only by the correct timing to remove a given message, but also by the unpredictable number of nodes of the graph in a real scenario and their possible dynamicity, hardly emulated in a simulation.

Chapter 3

Simulation environment

The probabilistic nature and the high number of nodes involved in this algorithm do not allow an accurate evaluation and proper adjustment before the effective deploy. As the effective ambit of application of the protocol is not predictable, it is necessary to test in a pseudo-realistic scenario. For this thesis, simulations have been executed using a simulation environment created by Parallel and Distributed Simulation Research Group [12] (Department of Computer Science, Università di Bologna); then the results have been compared to known algorithms. The core simulation environment is Advanced RTI System (ARTÌS) [13], a parallel and distributed simulation middleware, inspired by the High Level Architecture standard [14]. ARTÌS has been integrated with the Generic Adaptive Interaction Architecture (GAIA), a framework responsible of migrating simulation elements in the distributed environment to improve performances. On the top of this architecture, Large Unstructured NEtwork Simulator (LUNES) [15] uses services provided by ARTÌS and GAIA to simulate complex protocols on top of a network graph. Unfortunately, it is not possible to simulate every single aspect of a real scenario because the number of nodes, the width or diameter of the graph, which is the largest number of vertices which must be traversed to trover from one vertex to

another, and the number of edges might vary. Moreover it is not possible to assume a static network and it is not even realistic a graph where all nodes are homogeneous, with equal bandwidth and no nodes failure or packet loss. To soften the distance between a simulation and a real scenario, following tests and evaluations have been executed using the same stable graphs, with no losses and homogeneous nodes. In particular, one hundred graphs has been randomly generated by LUNES and each graph is composed of one hundred nodes, two hundred edges with a diameter of eight nodes.

The width of the graph is a fundamental information for the simulation since the Time To Live (TTL), an integer value which is decreased each hop through the graph and causes the message to be discarded when reaches 0, will be equal to the diameter of the graph.

LUNES also provides an important function: message caching. Since each message will have a unique identifier, the system will be capable of recognizing already received messages simply storing its ID. The default implementation drops messages already received, but different implementations may change its behavior.

Each simulation will return:

- Number of nodes of each graph;
- Average coverage;
- Average delay;
- Average number of messages sent in each dissemination;
- Overhead ratio of the dissemination.

The first and the fourth values are self-explicative, the second represents the percentage of nodes that have been reached by the information, the third the delay of the message since its first sending and the fifth is given by the

ratio between the number of messages sent during the dissemination and the minimum number of messages necessary to obtain a complete coverage [16].

Among those values, overhead and coverage are the most interesting, since they give an idea of how efficient the algorithm is, but the delay is also notable because it shows how quickly the information is propagated through the graph.

Before proceeding, some clarifications are needed about the following figures. Data represented in the figures are the average of 100 run for every algorithm repeated on the same 100 graphs, with 100 nodes and 200 edges. Each algorithm’s result is represented using two figures (e.g. 5.1 and 5.2) that focus respectively on the overhead and the delay referring to the same coverage.

Chapter 4

Analyzed gossip algorithms

In this section, two algorithms will be analyzed: Fixed Probability (FP) and Conditional Broadcast (CB). Both protocols are already built in LUNES and will be used as comparison for new algorithms in the following sections. Despite many other algorithms have been implemented in LUNES, those two have been chosen as metric not only because they represent a base gossip approach to the problem but also proposed algorithms have been inspired by CB and FP behaviors, merging their characteristics and aiming to soften their weakness.

Both algorithms belong to push family, as all the other protocols treated in this thesis. The next two paragraphs will briefly explain the functioning of CB and FP with the aid of pseudo-code. As stated above, LUNES provides caching for messages and drops those who have already been received, so following implementations will only consider the case where the message has not been received yet, since the caching system is transparent for the algorithm. Even if FP and CB do not directly use states, their capacity to recognize new from already received messages, due to caching, implicitly render their behavior dynamic and differs from message to message.

4.1 Conditional Broadcast (CB)

The Conditional Broadcast (CB) is a simple algorithm where messages are sent to all neighbors if the dissemination condition is met. In particular, after the reception of a new message, the node generates a random number between 0 and 1 and, if this number is equal or below the broadcast probability, the node forwards the message to all neighbors, otherwise it does nothing. The protocol has been improved removing the node that generated the message and the sender from the broadcast. All the following algorithms include this feature.

Algorithm 3: CB gossip

```
1:  procedure OnUpdate (s)
2:    if TTL > 1 then
3:      TTL = TTL - 1
4:      if random() ≤ threshold then
5:        for all n ∈ neighbors loop
6:          if n ≠ s then
7:            send update to p
8:          end if
9:        end loop
10:     end if
11:   end if
12: end procedure
```

TTL is used as termination condition; if the check fails (line 2), the message will be dropped, otherwise it will be forwarded to all neighbors provided that the random value is below or equal the *threshold* (line 4). The node excludes itself and the node who generated the message from the broadcast (line 6).

4.2 Fixed Probability (FP)

The Fixed Probability (FP) algorithm differs from CB because the probability is evaluated for each neighbor independently. That choice is the first attempt to better disseminate the information through the graph.

Algorithm 4: FP gossip

```
1:  procedure OnUpdate (s)
2:    if TTL > 1 then
3:      TTL = TTL - 1
4:      for all n ∈ neighbors loop
5:        if random() ≤ threshold and n ≠ s then
7:          send update to p
8:        end if
9:      end loop
10:   end if
11: end procedure
```

TTL is used as termination condition; if the check fails (line 2), the message will be dropped, otherwise it will be forwarded to all neighbors provided that the random value is below or equal the *threshold* (line 4). The node excludes itself and the node who generated the message from the broadcast (line 6).

Chapter 5

Proposed algorithms

Analyzing previous protocols it is noticeable that they are static, since the dissemination probability remains the same for the whole algorithm. This is not a good thing because, as explained in previous sections, push algorithms tends to be efficient in early rounds but may cause much overhead later due to a lower probability to reach an unconscious node every hop through the network. That leads to the conclusion that the algorithm should try to adapt during the dissemination to avoid infecting again the same nodes. This is not a trivial problem, because due to the random and simple nature of Gossip algorithms, nodes are not aware of the current situation, thus a high probability may lead to loops through the graph while a low probability will leave a high number of nodes unconscious.

Ideally, if nodes were somehow able to know how the dissemination is going, they would be able to correct their behavior due to the current situation. Unfortunately, the decentralized nature of Gossip, and more in general of P2P algorithms, does not allow nodes to be conscious of what is happening to others and anyone of them has a global view of the graph. The only way to increase nodes awareness of the dissemination is to increase their communication, which is exactly what we are trying to avoid.

5.1 Descending Probability (DP)

A possible approach to the problem consists in lowering the probability proportionally to the time elapsed since the message has begun its journey through the network. This relies on the idea that the longer a message has travelled, the higher is the number of nodes already informed, so a slowdown would reduce global overhead without compromising too heavily dissemination.

To obtain that behavior, different mechanisms are available:

- Starting the dissemination with a fixed initial value, then reducing it by a constant at each hop;
- Creating a disproportional relation between the time spent by the message in the graph and the dissemination probability;
- Assigning to every TTL a different probability, not necessary related or different from previous or subsequent values.

The last option is pretty easy to implement, since 8 values could be saved into a vector, where at each position corresponds the TTL. Even if this method gives more control for values at each step, it would cause too much effort to study which value would be the best for each position, since combinations are in the order of 100^8 possible combination and may lead to a completely different algorithm.

The first mechanism has the appreciable characteristic to require only two values: the initial and the decrementing one; unfortunately, just like the third mechanism, it has a very high number of combinations but has also the drawback of a static value for the whole dissemination.

I think that it would be more interesting to have a full control of the dissemination instead of a partial one.

The second mechanism is the one which has been chosen for DP, not only because it allows a harmonious and proportional decrease for the probability, but also because it has a reduced pool of possible values and only requires a value that will be proportionated to the time spent by the information through the graph simply multiplying it for the TTL. Adopting the last one as meter for the age of the message and creating a proportional relation between it and the probability, an easier implementation is obtained, since that value is already present and managed in LUNES and nodes are already aware of its existence. Each node only needs to read the TTL of the received message and obtain the current dissemination probability simply multiplying a constant for the TTL. Of course, the node who generated the message will spread it to all its neighbor, since none of them has been informed, so the algorithm will start its evaluation after the first hop. DP algorithm is meant to obtain the dissemination probability as the result of the multiplication between a constant and the TTL, but since the TTL value is decreased immediately after reception, to prevent the constant to be multiplied for zero and provide a better dissemination, the value used for this multiplication will be equal to the TTL + 1, resulting in a range between 1 and 8 due to the initial value. This gives a range for the constant as well, which can be between 1 and 12. Dissemination behavior is similar to FP because each neighbor is evaluated independently.

Note that the algorithm uses indirectly the caching system provided by LUNES, that simply drops messages already received, which is transparent for the algorithm, thus the implementation will not consider the reception of an already received message.

Algorithm 5: DP gossip

```
 1: procedure OnUpdate (s)
 2:   if TTL > 1 then
 3:     TTL = TTL – 1
 4:     threshold  = constant * (TTL + 1)
 5:     for all n ∈ neighbors loop
 6:       if random() ≤ threshold and n ≠ s then
 7:         send update to p
 8:       end if
10:     end loop
12:   end if
13: end procedure
```

If TTL check is successful (line 2), this is decreased (line 3), otherwise the message will be dropped. A new *threshold* is set each update (line 4) and the message will be forwarded to all nodes which pass the random evaluation (line 6). The node excludes itself and the one who generated the message from the dissemination (line 6). As stated above, *constant* is a value between 1 and 12, and of course *threshold* is comprised between 0 and 100.

As shown in figure 5.1, DP can reach a good coverage without exceeding in overhead; despite that, the algorithm is not capable of reaching a full coverage which may become a considerable problem if the network requires continuous information for all nodes. Figure 5.2 shows better results for what concerns delay, which decreases with coverage increase, so the lack of coverage may be softened by a good delay and constant dissemination.

5.2 Double Descending Probability (DDP)

Descending Probability (DP) tries to reduce overhead in later rounds, but its probability drops down significantly after some iterations, down to 12% in the best case. The direct consequence is that after a certain value, the number of messages forwarded through the graph is not sufficient to reach all uninformed nodes left. To soften this gap, the algorithm has been modified resetting the probability to its initial value in the middle of the dissemination. The idea behind the modification is that after some iteration with high dissemination, the message may have reached new portions of the graph which have not been sufficiently informed yet, so reducing the dissemination probability too early and heavily may be the cause of unsatisfying coverage.

Algorithm 6: DDP gossip

```
 1: procedure OnUpdate (s)
 2:   if TTL > 1 then
 3:     TTL = TTL – 1
 4:     if TTL ≥ 4 then
 5:         threshold  = constant * (TTL + 1)
 6:     else
 7:         threshold  = constant * (TTL + 5)
 8:     end if
 9:     for all n ∈ neighbors loop
10:       if random() ≤ threshold and n ≠ s then
11:         send update to p
12:       end if
13:     end loop
14:   end if
15: end procedure
```

TTL is checked (line 2) and if above one, it is decreased (line 3). If the dissemination is in the first half (line 4), the constant will be multiplied for TTL + 1 (line 5), otherwise it will be multiplied for TTL + 4 (line 7). Likewise DP, DDP *threshold* will be reevaluated at each reception and dissemination will be fixed (lines 9-13).

Figure 5.3, shows that even if coverage is increased by almost 10%, DPP is not capable of reaching full coverage and its delay is quite worse compared to DP delay. Despite this, its coverage is slightly better and the overhead is acceptable, so it may be worth.

5.3 Comparison with common algorithms

This section will concern practical tests, comparisons and evaluations.

Figure 5.5 compares the coverage and the overhead of DP, DDP, FP and CB; many things are notable:

- DP does not show striking results, compared to other algorithms, in particular it tends to emulate FP, without granting the complete coverage of the latter;
- DDP instead, is slightly better than FP and can reach a coverage above 95%, but still cannot grant 100%;
- CB, even if it is worse than all the other algorithms, tends to approximate FP when both are about to reach 100% coverage. That is because FP with a high probability tends to have the behavior of a broadcast dissemination.

Figure 5.6 compares delay and coverage of DP, DDP, FP and CB.

- DP performs even worse than CB, offering the slowest dissemination;
- FP has the quickest dissemination;
- DDP seems to act very similarly to FP, despite having worse results.

This first evaluation has shown unsatisfying results: neither DP nor DDP are currently capable of reaching full dissemination, but since the latter has reached an acceptable overhead for an almost acceptable dissemination, algorithms will be analyzed separately.

DP high delay is the result of a slow and highly selective dissemination in later rounds; its efficiency fades when the algorithm is at its ending, likely because it reaches a probability too low to satisfy all unconscious nodes left and those who are informed suffers a long wait.

My first thought was that the probability was getting reduced too much at each hop, but many tests trying to slow its reduction or keep a higher dissemination in later rounds have not shown better results. These tests led to the conclusion that even if DP definitely needs improvements, these improvements cannot be given by values adjustments but only changing the DP behavior, or maybe introducing functionalities to manage more efficiently the dissemination, keeping a good level until later rounds.

DDP values showed in figure 5.5 require a closer look since it is pretty hard to notice its difference from FP. Figure 5.6 consider a smaller section to better clarify the results; and as it is shown, DDP performs slightly better than FP even if can cover narrower values. Despite its not impressive results for what concerns overhead, its delay

is clearly lower (figure 5.7) for the same coverage values, but it is higher for a higher coverage. Those results have been given by the capacity of the algorithm to maintain an appropriate amount of nodes active during the dissemination. That deficiency of DP is a problem that does not afflict CB and FP since their static dissemination allows a constant spreading of the information and has been partially resolved by DDP.

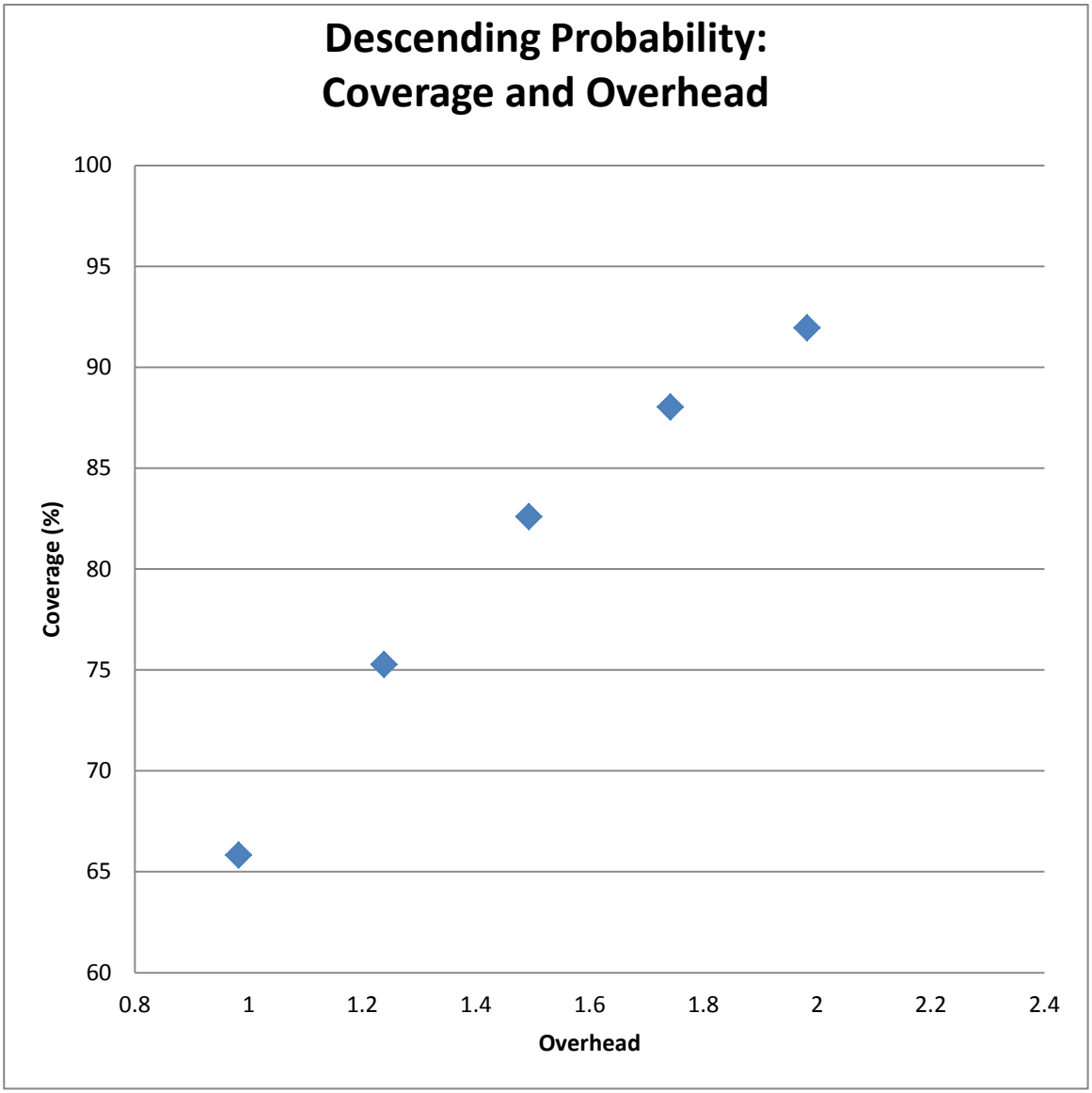

Figure 5.1: Descending Probability: coverage and overhead

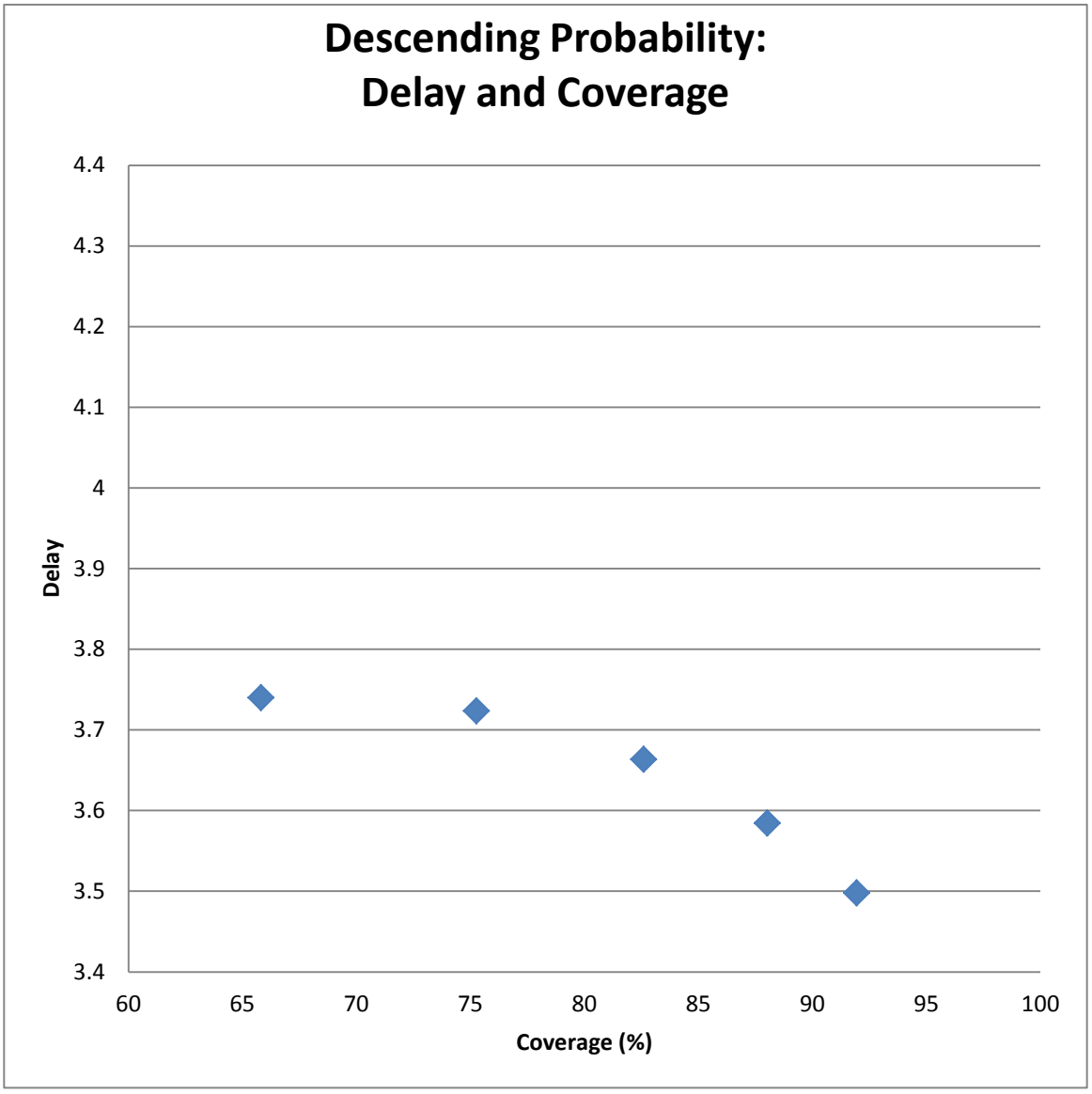

Figure 5.2: Descending Probability: delay and coverage

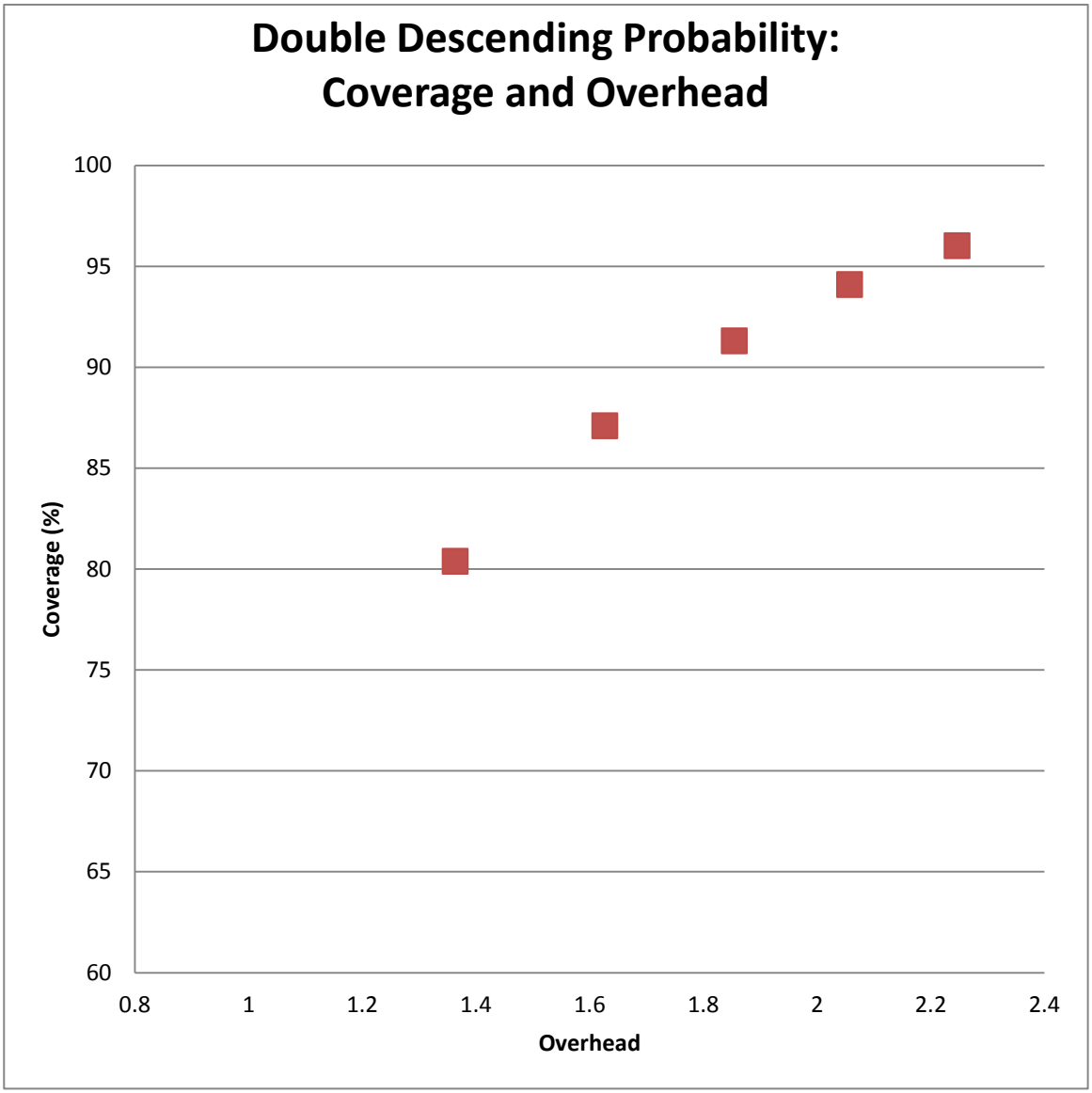

Figure 5.3: Double Descending Probability: coverage and overhead

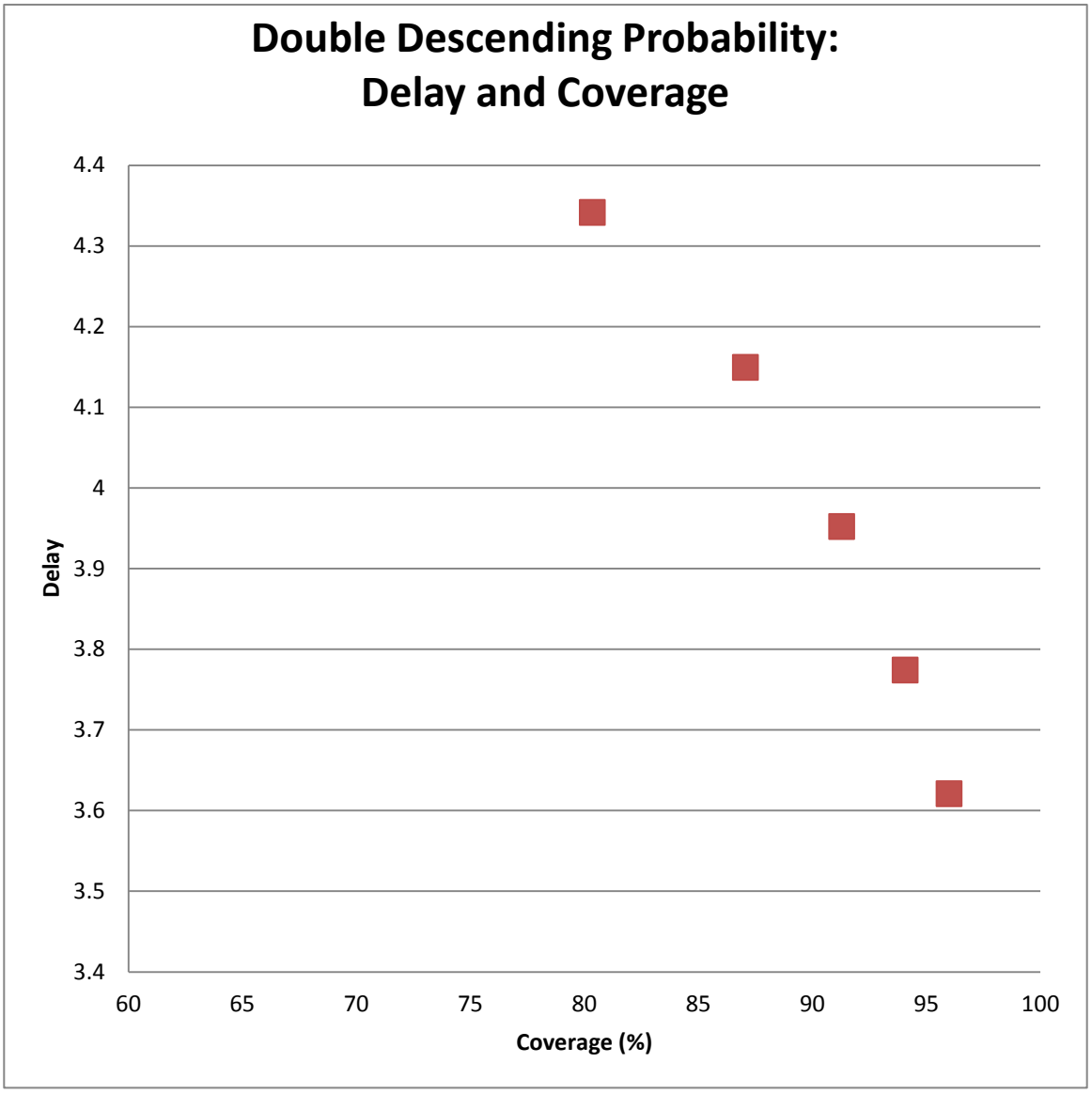

Figure 5.4: Double Descending Probability: delay and coverage

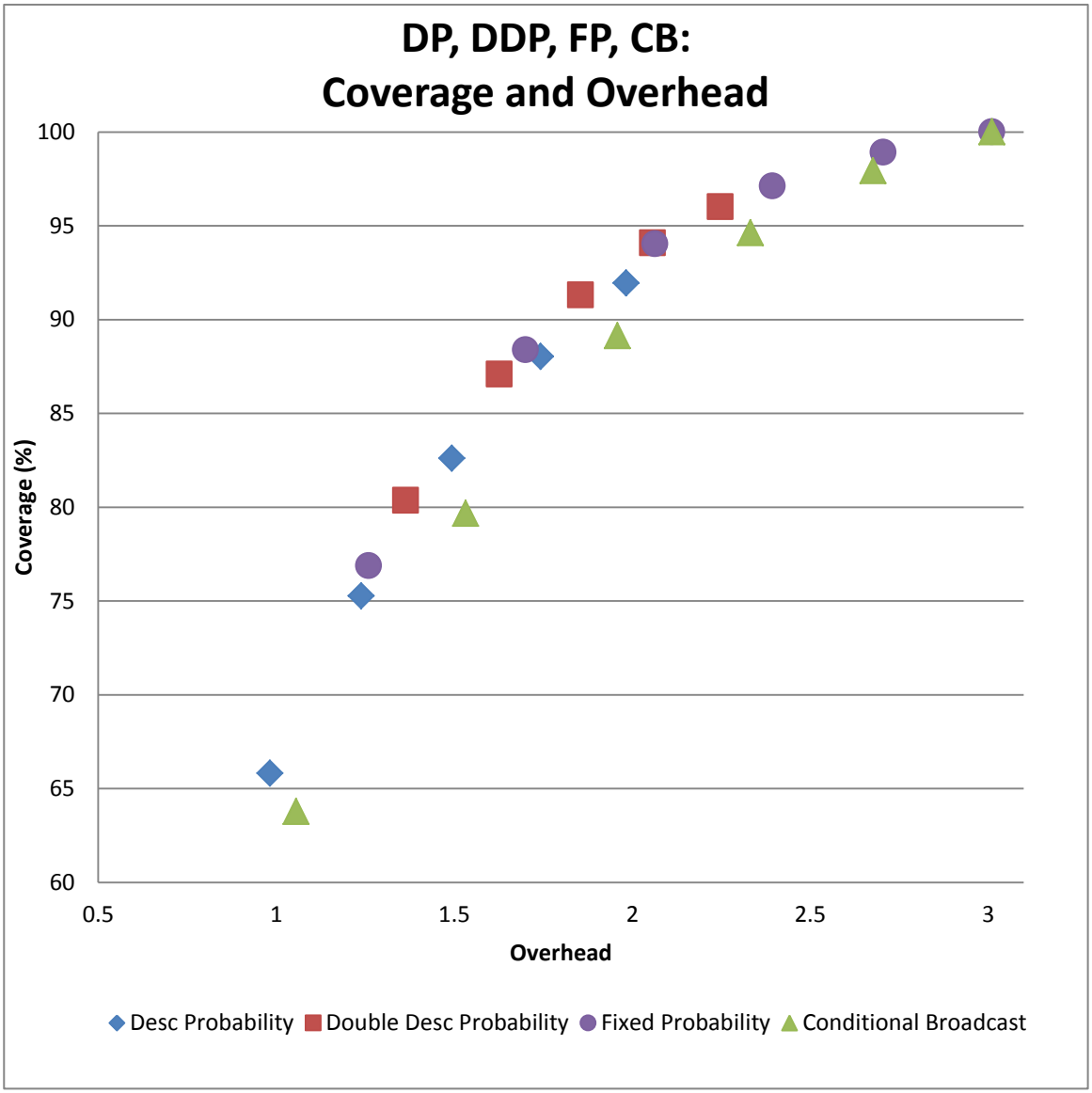

Figure 5.5: DP, DDP, FP, CB: coverage and overhead

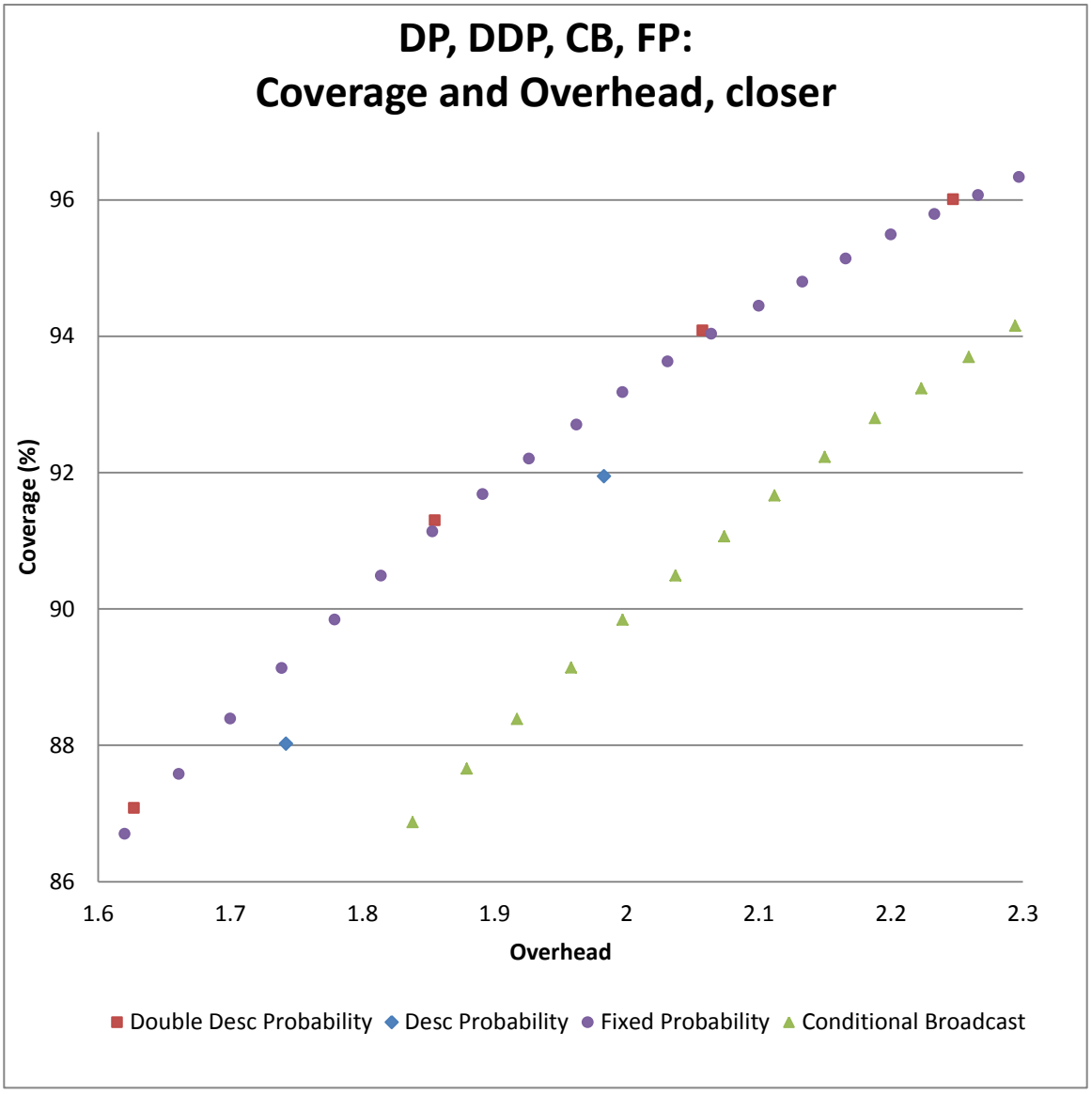

Figure 5.6: DDP, FP, CB: coverage and overhead, close

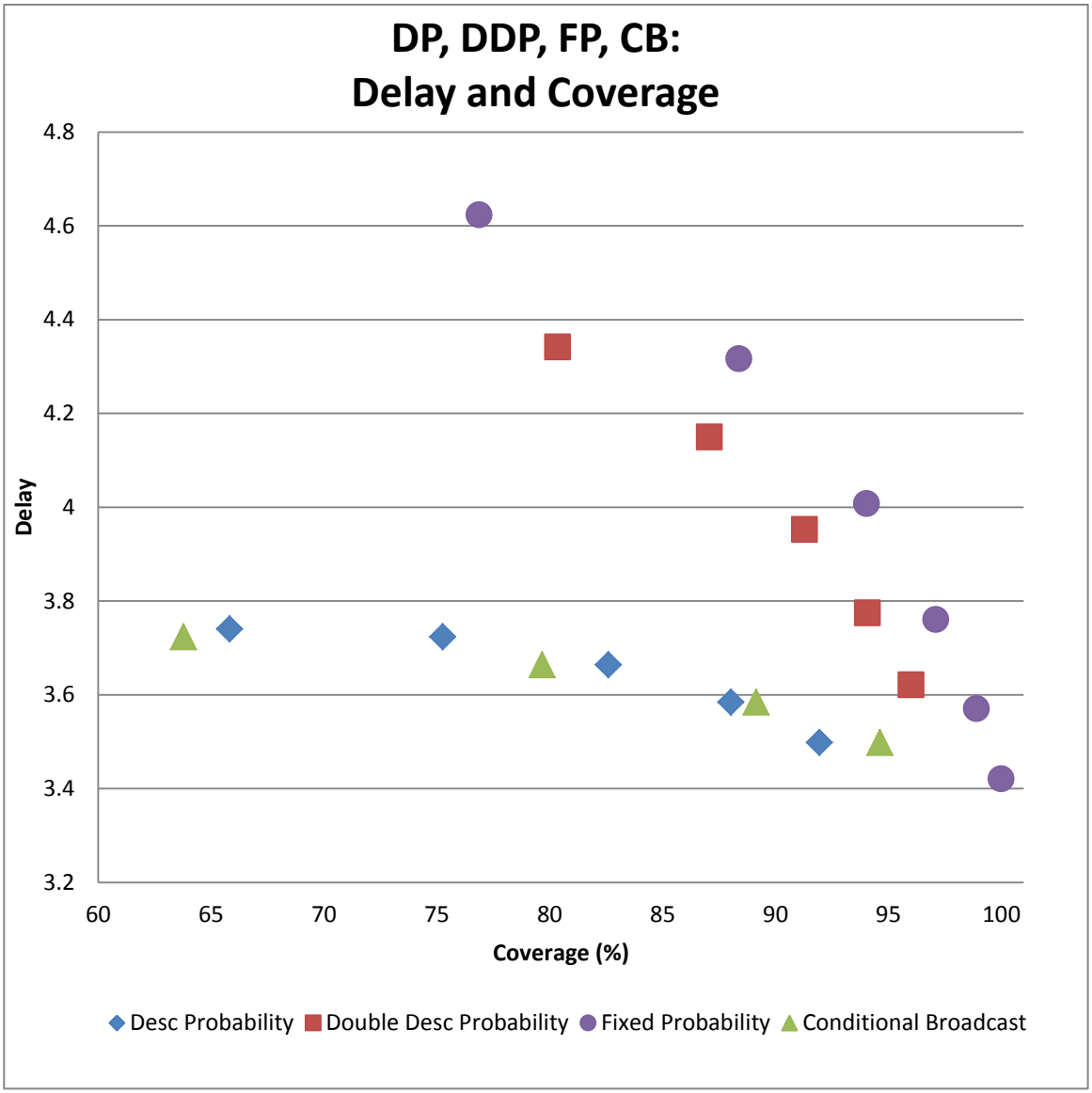

Figure 5.7: DP, DDP, FP, CB: delay and coverage

Chapter 6

Advanced functionalities

Previous simulations have not shown the expected results, anyway a refinement of DP and DDP with the implementation of more functionalities may improve their performance. First it is important to analyze what is wrong with their behavior and what the desired result is. Both DP and DDP cannot reach 100% dissemination, so a mechanism to reach it is necessary, possibly without upsetting the whole algorithm; in addition, their overhead and delay are still too high. Unfortunately, the reasons of their failure are not so easy to interpret: it is not possible to easily know why the algorithm is causing overhead, for example the first part of the dissemination might involve a limited number of nodes who are not sufficient to spread the information to different sections from their own, or the algorithm may have higher dissemination probability that causes redundancy in some part of the dissemination.

FP and CB are capable of reaching high dissemination when their probability strongly increases, approximating a flooding protocol. Since DP and DDP are both derived from FP and CB, increasing their dissemination probability would give the same results.

Even if DP was meant to reduce global overhead, progressively decreasing its probability during the dissemination seems to exceed its

purpose, as explained above. This consideration led to the conclusion that a mechanism which acts oppositely to descending probability may solve or at least soften the problem.

6.1 Conditional incrementing broadcast

The Conditional Incrementing Broadcast (CIB) is one of the tested mechanisms which shows the best results. DP has been slightly modified to include this functionality, as the pseudo-code below shows.

Algorithm 7: DP + CIB (DPCIB)

```
 1: procedure OnUpdate (s)
 2:     if TTL > 1 then
 3:         TTL = TTL – 1
 4:         threshold  = random(0, TTL)
 5:         if (threshold < 1) then
 6:             for all n ∈ neighbors loop
 7:                 if n ≠ s then
 8:                     send update to p
 9:                 end if
10:             end loop
11:         else
12:             threshold  = constant * (TTL + 1)
13:             for all n ∈ neighbors loop
14:                 if random() ≤ threshold and n ≠ s then
15:                     send update to p
16:                 end if
17:             end loop
18:         end if
19:     end if
20: end procedure
```

TTL is checked (line 2) and decreased (line 3) as usual, but *threshold* is used for two random evaluations (lines 4 and 12), in particular, lines 4-11 implement CIB: a random number between 0 and TTL (both included) will be generated (line 4), and if this value is lower than 1 (line 5) this node will broadcast to all its neighbor (lines 6-10) instead of using fixed dissemination (lines 11-18). With this modification, the message will have 1/TTL chances to be broadcast; probability is 1/8 the first hop, up to 100% when the TTL is 0. This behavior can be changed adding to the TTL a fixed value, for example adding 1 will result in a 1/2 broadcast probability when TTL is 0. Different cases have been evaluated but, due to the reduced number of messages that continue their travel through the graph, omitting the value does not seem to rely on the overall performance, on the contrary it provides a better coverage.

Figure 6.1 compares DP coverage and overhead with its advanced version: DPCIB. The latter algorithm seems to continue the behavior of its original version, with a better coverage for the same overhead, reaching over 95% graph coverage. Including CIB seems to have soften DP deficiencies, and despite not being able to solve them completely, it still improved algorithm performances.

Figure 6.2 compares DPCIB and DP coverage and delay. DPCIB behavior may confuse, because its values are much more concentrated than DP, and the delay seems to drop down quickly, but their density is just given by the higher proximity among coverage values of the algorithm. In fact, DPCIB has much higher delay for the same coverage, but reduces it quickly when its coverage increases.

DDP has been modified to include CIB but, since it can be considered the result of two DP algorithms applied two different parts of the dissemination, CIB probability will be proportionated to DDP probability.

Algorithm 8: DDP + CIB (DDPCIB)

```
1: procedure OnUpdate (s)
2:     if TTL > 1 then
3:         TTL = TTL – 1
4:         if TTL ≥ 4 then
5:             threshold = random(0, TTL)
6:         else
7:             threshold = random(0, TTL + 4)
8:         end if
9:         if (threshold < 1) then
10:            for all n ∈ neighbors loop
11:                if n ≠ s then
12:                    send update to p
13:                end if
14:            end loop
15:        else
16:            if TTL ≥ 4 then
17:                threshold = constant * (TTL + 1)
18:            else
19:                threshold = constant * (TTL + 5)
20:            end if
21:            for all n ∈ neighbors loop
22:                if random() ≤ threshold and n ≠ s then
23:                    send update to p
24:                end if
25:            end loop
26:        end if
27:    end if
28: end procedure
```

Line 4 checks if the dissemination is in the first or second half, then adapts the *threshold* (lines 5 and 7). The rest of the code does not vary from previous examples.

Figure 6.3 compares DPPCIB coverage and overhead with its previous version. This algorithm's results make it appear as an extension of DPP, but unlike DPCIB, it does not seem to provide notable improvements except a couple of points of coverage, that is slightly above 97% in the best case. These results could have been expected considering that CIB is a functionality meant to increase the dissemination when the probability is getting too low, proportionating the chances to broadcast disproportionally to the dissemination probability, and since DDP characteristic is to restore initial values in the middle of the algorithm, round 5 will be equal to round 1, round 6 to 2, and so on. Thus instead of having a broadcast probability from 1/8 to 1/1, it will be 1/8-1/4 for the first half and 1/8-1/4 for the second half. Although it could have been implemented without considering the probability, keeping this probability from 1/8 to 1/1 just like DPCIB, simulations have shown negative effects on the algorithm, causing more overhead without increasing the coverage, likely because of an excessive dissemination in later rounds. Another possibility is that DDP performs well enough in its later rounds that the presence of CIB is almost unnoticed.

As figure 6.4 shows, while the relation between overhead and dissemination has remained the same, CIB has reduced the delay of the algorithm.

6.2 Final observations

As conclusion to this chapter, the two algorithms explained in the previous section have been compared to FP and CB. Again, figure 6.5 does not clarify enough differences between FP, DPCIB and DDPCIB, but certainly it gives the idea of how limited the range of new algorithms is confronted to canonicals.

DPCIB performs better than FP when the coverage is between 89 and 94 (figure 6.6), but then it is overturned when reaching its higher coverage. Its delay (figure 6.7) is considerably lower for the coverage that DPCIB can cover; unfortunately FP reduces its delay even more when the coverage increases, nullifying DPCIB initial good results.

As figure 6.6 shows, DDPCIB performances are slightly above FP, for all its range, between 89 and 98, since for the same coverage the overhead of FP is higher. The problem of the coverage is not yet resolved and is not mitigated by a lower delay (figure 6.7), because despite it offers a lower delay for the same coverage, the increase of the coverage of FP corresponds to a reduction of the delay.

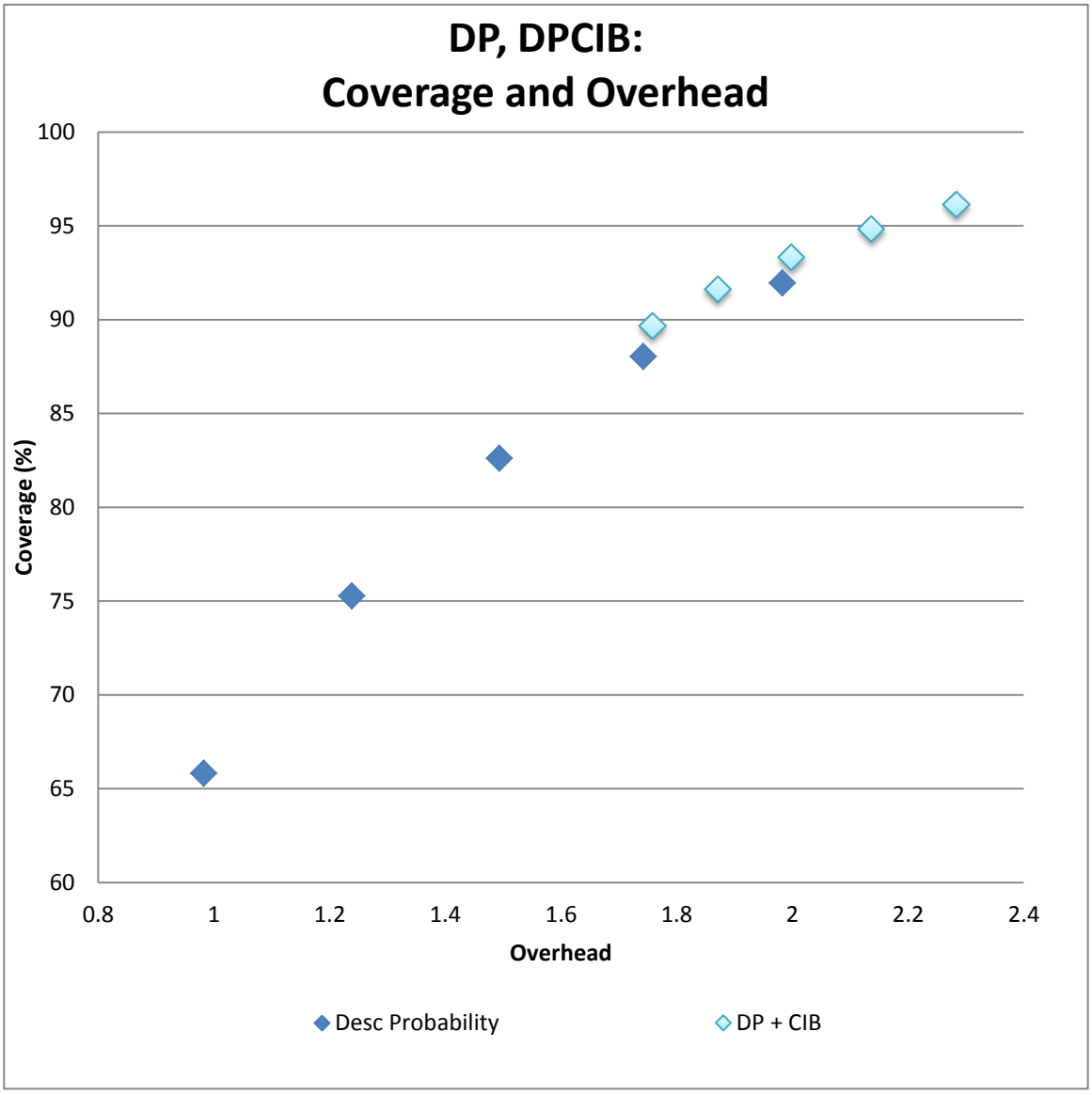

Figure 6.1: DP, DP + CIB: coverage and overhead-

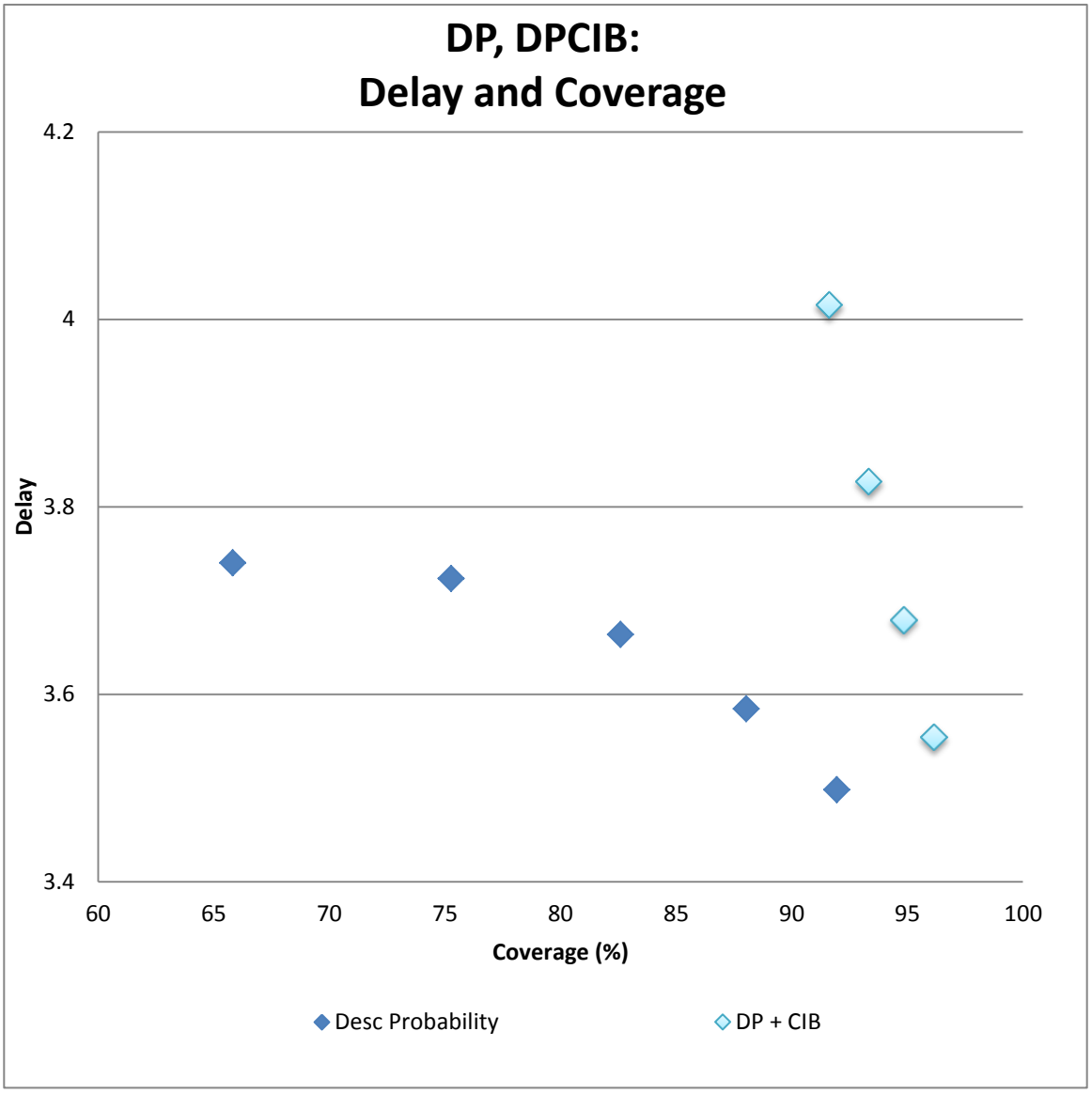

Figure 6.2: DP, DP + CIB: delay and coverage

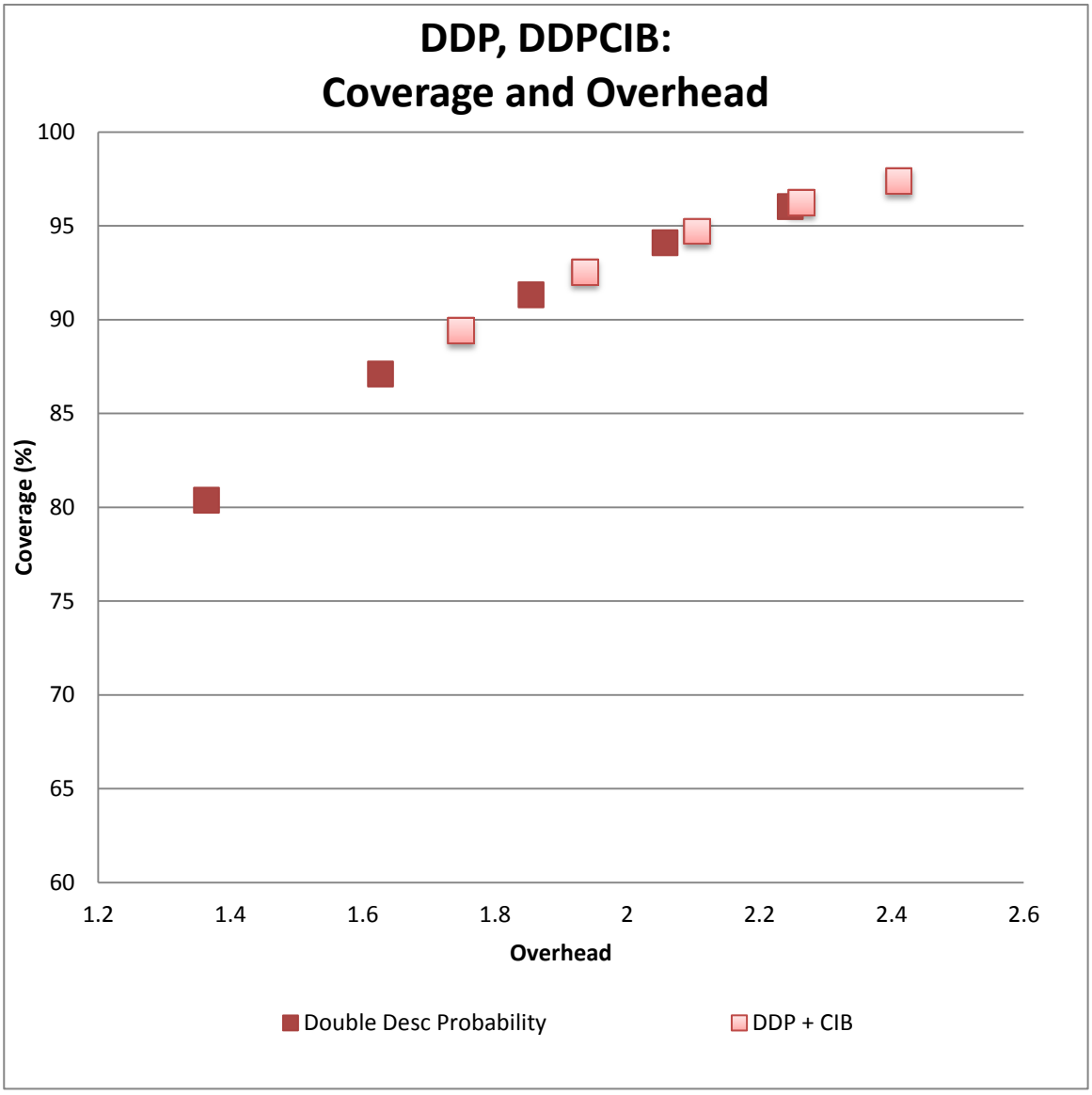

Figure 6.3: DDP, DDP + CIB: coverage and overhead

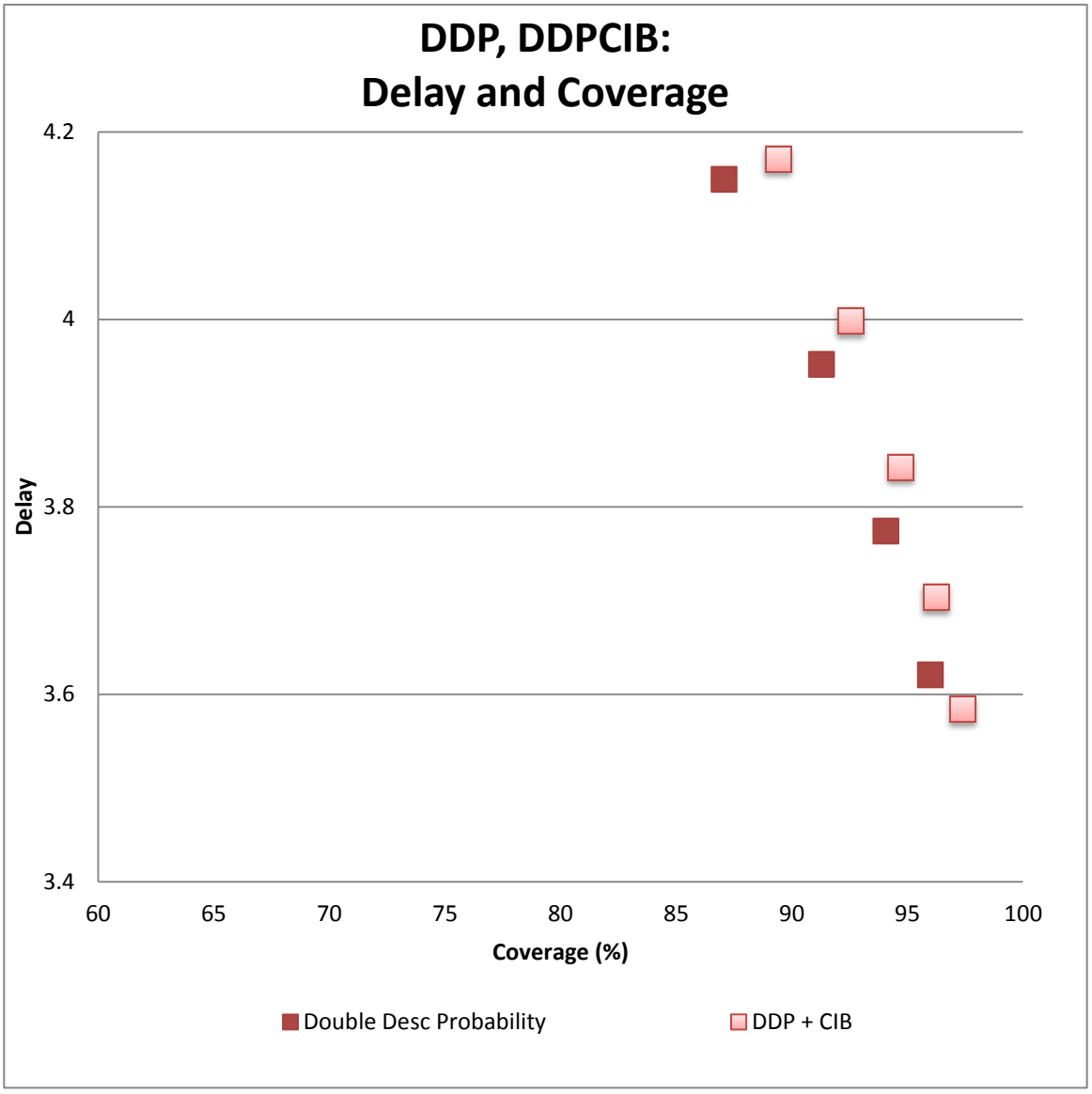

Figure 6.4: DDP, DDP + CIB: delay and coverage

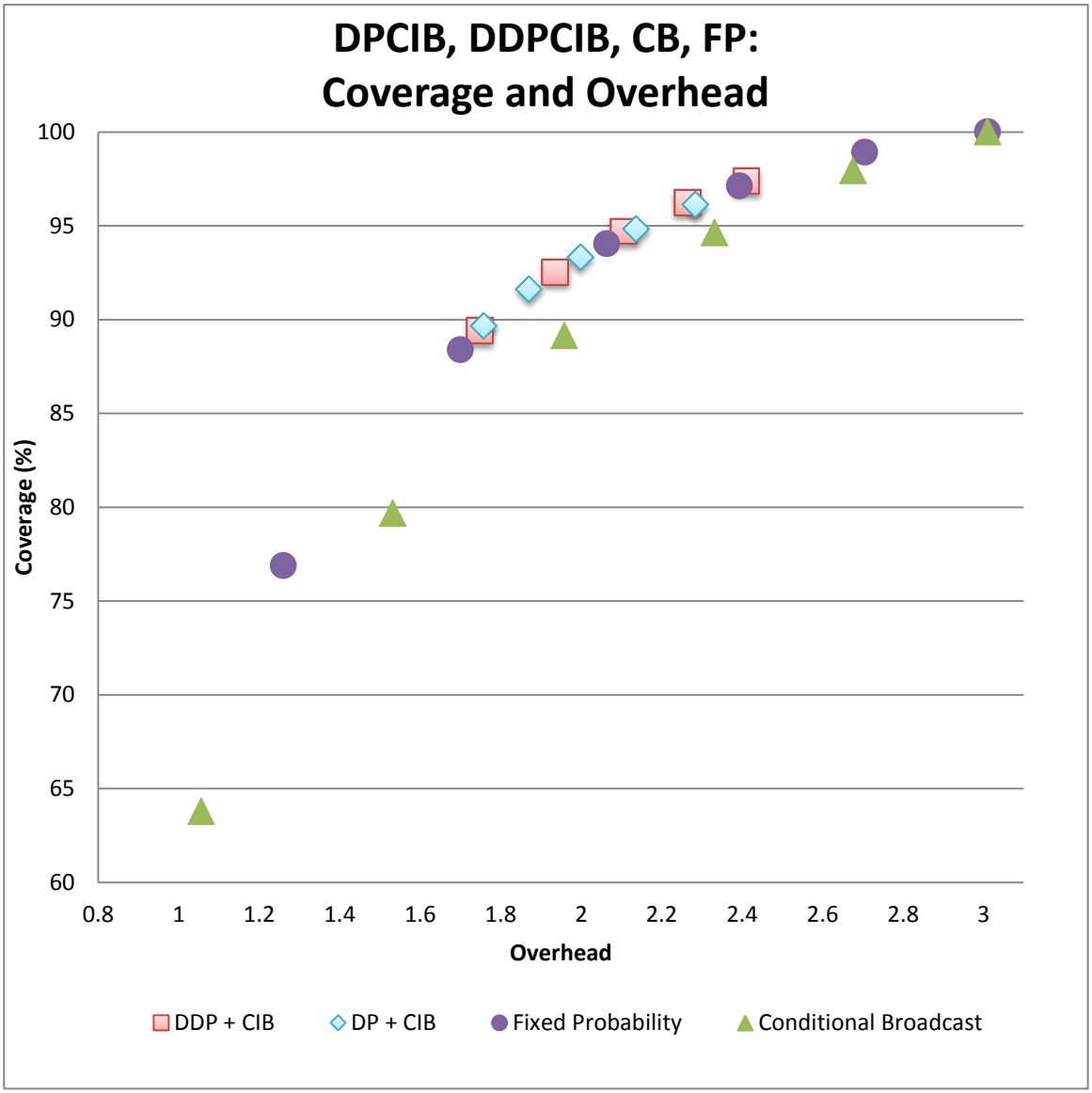

Figure 6.5: DP + CIB, DDP + CIB, FP, CB: coverage and overhead

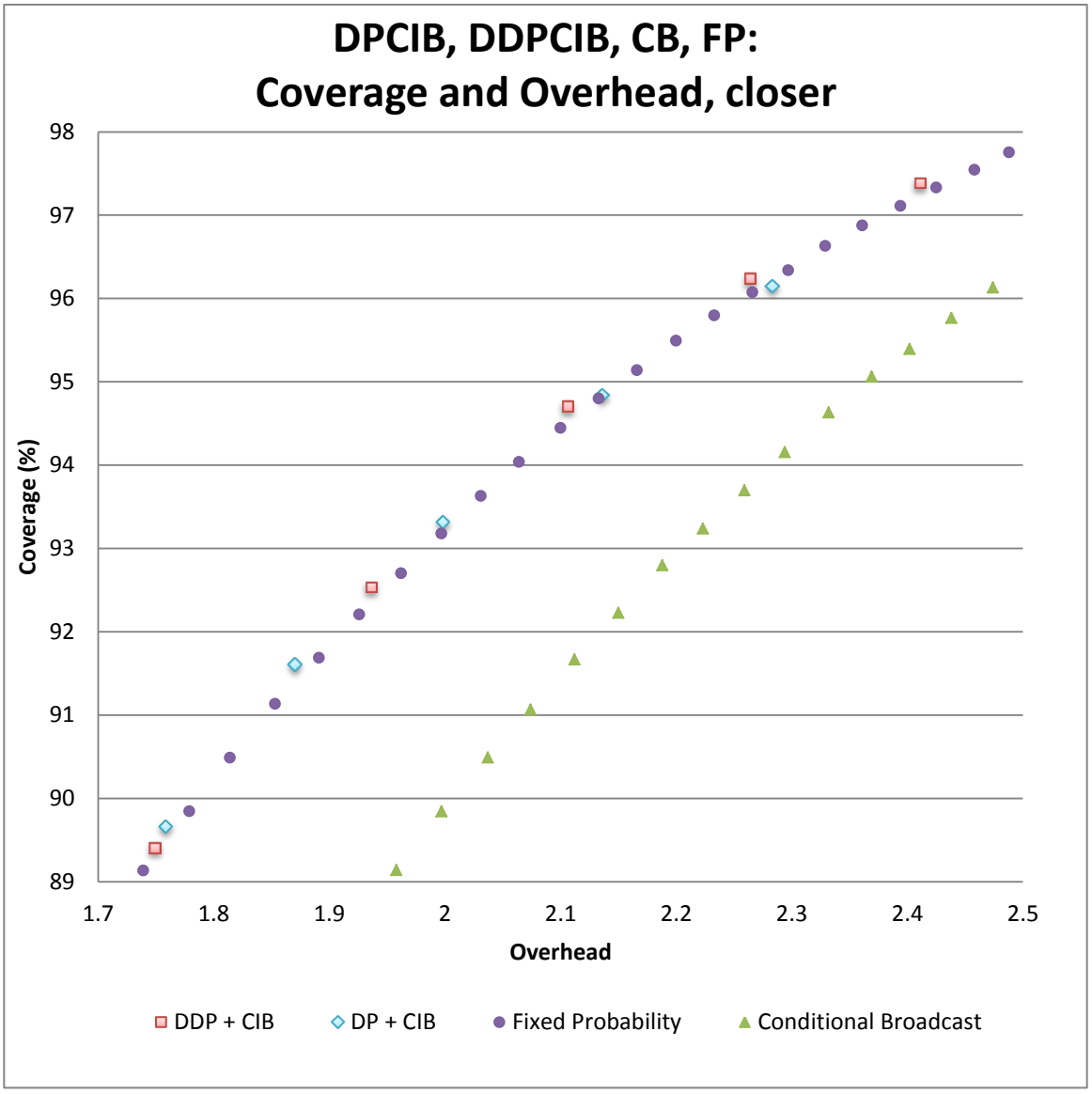

Figure 6.6: DP + CIB, DDP + CIB, FP, CB: coverage and overhead, closer

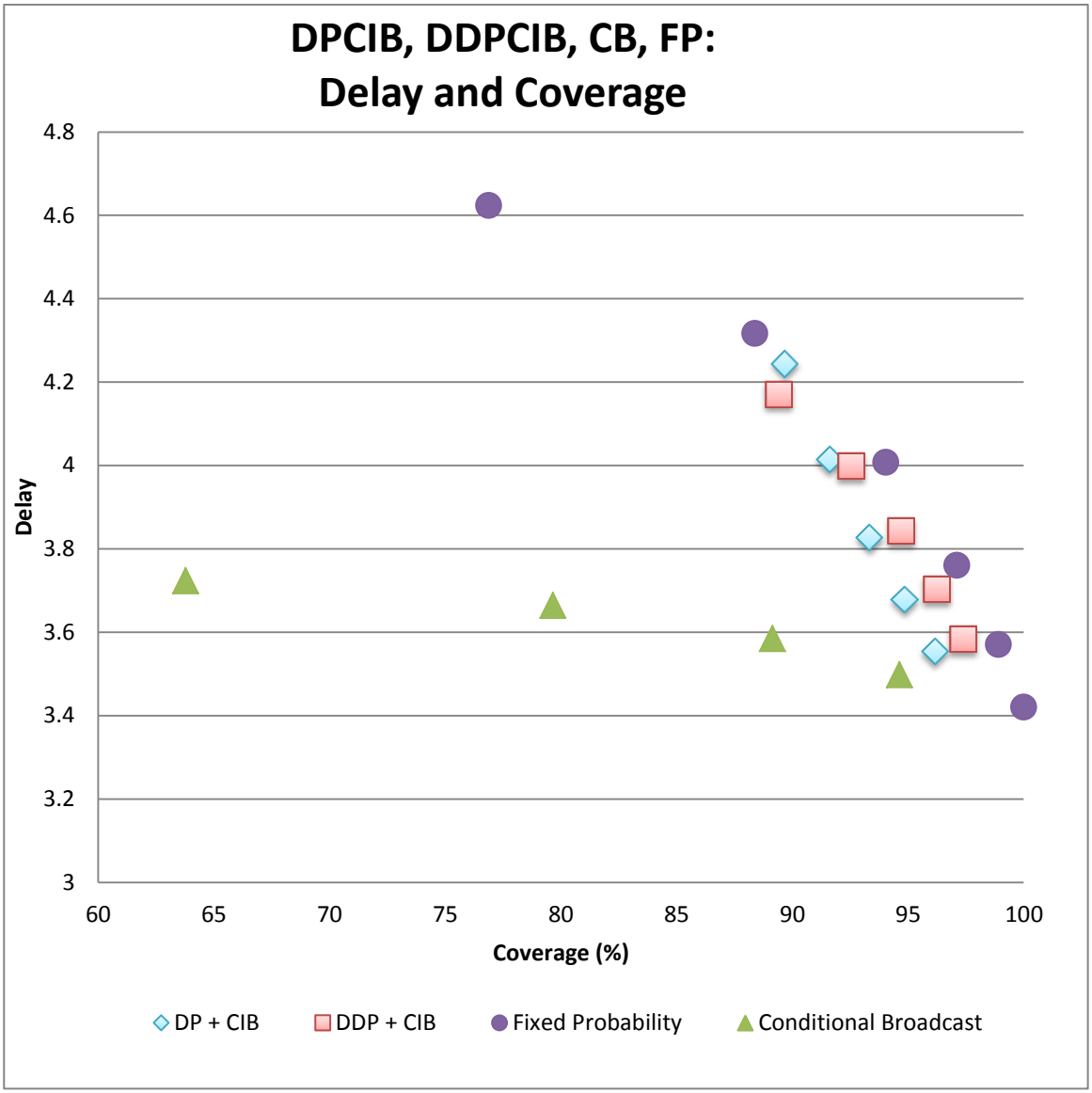

Figure 6.7: DP + CIB, DDP + CIB, FP, CB: delay and coverage

Chapter 7

Conclusions

In this thesis, we have considered practical and theoretical evaluations about gossip-epidemic algorithms, comparing them with proposed algorithms and analyzing their behavior. Unfortunately, results are not impressive because, as widely explained above, even though their overhead and delay are lower for certain coverage values they cannot grant full coverage. The incapacity of all proposed algorithms to reach full coverage is a notable gap that excludes them from a wide number of scenarios where leaving part of the network uninformed is not acceptable; this aspect is more negative than what may appear because is the result of the failure of algorithms to adapt themselves to different scenarios.

Despite unsatisfying results, this thesis has focused on studying these results regardless their success to better understand reason of their success or, in this case, their failure hoping to contribute to the study of other algorithms with ideals and qualities to emulate and defects to avoid.

In addition, literature algorithms have been explained and treated at a high level to better allow a clear understanding of their ideals and functionalities even for those who approach this topic for the first time.

Another important aspect observed is the huge distance between the expected results of an idea and its effective implementation and its strong

dynamicity; indeed DP was not originally meant to operate as it has been explained, but has been adapted before and during the writing of this thesis to refine its behavior to better fit the studied scenario to finally be evolved in DDP, while CIB was originally part of a more complex algorithm discarded after the implementation phase.

Efficient data dissemination is an interesting field of research due to its wide application to different sectors and to the increasing interest of companies to improve their services with the minimum effort.

In particular, last years have seen a crescent interest for mobile networks since nowadays a high percentage of population has an electronic device capable of interface itself with others thanks to various applications. These networks are highly dynamic and since they follow human behaviors they can be interpreted with the aid of sciences like sociology or psychology to better adapt the algorithm to the current scenario.

These years have been full of changes and technology's evolution seems to proceed at high speed, offering solutions to unresolved problems and improving performances. Gossip-epidemic algorithms have the appreciable characteristic to adapt themselves to ambits where much in required but few information are available and their characteristics seems to fit perfectly the current technology scenario. Indeed, in computer science everything comes to a price, and gossip algorithms are no exception; in fact, they offer high flexibility, simplicity, robustness and efficiency, but their simplicity often hides an intense and accurate study which is not always rewarded with satisfactory results.

For all these reasons, I think that gossip-epidemic algorithms have the potential to maintain a significant role for the coming years.

7.1 Future works

Test and evaluations has been conducted using static graphs, with nodes immune to failure or packet losses. Even if a network with a stable topology for the current technology is possible to obtain, gossip algorithms have raised interest for their notable results in highly dynamic networks, thus studying algorithms behavior in a real or pseudo real scenario is much more interesting. This has the notable drawback to require a deep knowledge of the application context, or at least the behavior. In fact, different ambits may have different characteristics for what concerns nodes failures: P2P networks are highly dynamic, but peer are much more likely to leave once they have completed the download or soon after, while nodes' failures in a sensor networks could be simulated statistically estimating their living time or the probability to have power loss. I think that would be interesting studying gossip algorithms applied to a specific ambit, possibly in a pseudo real scenario like a cluster or a small network.

7.2 Acknowledgements

I would like to thank my parents for giving me the opportunity to study one of the few fields that really fascinates me and all members of my family for supporting me during my school and university career.

Thanks also to my friends and whoever are part of my everyday life despite my nature not always easy to deal with.

Last but not least I would like to thank my teacher and supervisor Gabriele D'Angelo for giving me the possibility to write this thesis and for his constant support, availability and scrupulosity.

Bibliography

[1] Tuncer C. Aysal, Mehmet E. Yildiz, and Anna Scaglione. *Broadcast gossip algorithms*, Information Theory Workshop, 2008, pages 343 – 347.

[2] Devavrat Shah. *Network gossip algorithms*. Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on, pages 3673 – 3676.

[3] James F. Kurose, Keith W. Ross. *Computer Networking: A Top-Down Approach 6th edition*, Prentice Hall, 2012.

[4] Walid Ben-Ameur, Pascal Bianchi Jeremie Jakubowicz. *Robust Average Consensus using Total Variation Gossip Algorithm*, 2012 6th International ICST Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS), pages 99 – 106.

[5] Bogdan Ghit, Florin Pop, Valentin Cristea. *Epidemic-Style Global Load Monitoring in Large-Scale Overlay Networks*, 2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, pages 393 – 398.

[6] Rico Piantoni, Constantin Stancescu. *Implementing the Swiss Exchange Trading System*. Fault-Tolerant Computing, 1997. FTCS – 27. Digest of Papers., Twemty-Senventh Annual International Symposium onn. Pages 309 – 313.

[7] D. Shah. *Gossip algorithms*, vol. 3 of Foundations and Trends in Networking, Now Publishers Inc., 2009.

[8] Isabelle Demeure, Ruijing Hu, Julien Sopena, Luciana Arantes, Pierre Sens. *Fair Comparison of Gossip Algorithms over Large-Scale Random Topologies*, 2012 31st International Symposium on Reliable Distributed Systems, pages 331 – 340.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, McGraw-Hill. *Introduction to Algorithms, 3rd edition*, The MIT Press, 2009.
[10] Márk Jelasity. *Gossip*. Self-organising software, 2011. Pages 139 – 162.
[11] Ali Saidi, Mojdeh Mohtashemi. *Minimum-Cost First-Push-Then-Pull Gossip Algorithm*, 2012 IEEE Wireless Communications and Networking Conference: Mobile and Wireless Networks, pages 2554 – 2559.
[12] http://pads.cs.unibo.it/doku.php?id=
[13] Luciano Bononi, Michele Bracuto, Gabriele D'Angelo, Lorenzo Donatiello. *Scalable and efficient parallel and distributed simulation of complex, dynamic and mobile systems*. Techniques, Methodologies and Tools for Performance Evaluation of Complex Systems, 2005. (FIRB - Perf 2005). 2005 Workshop on, pages 136 – 145.
[14] IEEE Computer Society. *1516 – 2000 – IEEE Standard for Modeling and Simulation* (M&S) *High Level Architecture* (HLA) – *Framework and Rules*.
[15] Gabriele D'Angelo, Stefano Ferretti. *LUNES: Agent-based Simulation of P2P Systems.* Proceedings of the International Workshop on Modeling and Simulation of Peer-to-Peer Architectures and Systems (MOSPAS 2011). Istanbul (Turkey), IEEE, July 2011. ISBN 978-1-61284-382-7.
[16] Gabriele D'Angelo, Stefano Ferretti, Moreno Marzolla. *Adaptive event dissemination for peer-to-peer multiplayer online games*. Proceedings of 2nd ICST/CREATE-NET Workshop on DIstributed SImulation and Online gaming (DISIO 2011). In conjunction with SIMUTools 2011. Barcelona, Spain, March 2011. ISBN 978-1-936968-00-8.