ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Scienze e Tecnologie Informatiche

DATA DISSEMINATION ON OVERLAY NETWORKS THROUGH GOSSIP

Tesi di Laurea in Reti di Calcolatori

Relatore:
GABRIELE D'ANGELO

Presentata da:
GIULIO CIRNIGLIARO

Sessione I
Anno Accademico 2012-2013

Dimidium facti, qui coepit, habet: sapere aude;
incipe. Vivendi qui recte prorogat horam,
rusticus expectat, dum defluat amnis; at ille
labitur et labetur in omne volubilis aevum.

Orazio (Epistulae I, 2)

Sommario

Nell'ultimo decennio si è assistito ad un aumento esponenziale dei dispositivi connessi alla Rete, che siano essi personal computer, tablet o smartphone. Parallelemente vi è stato anche un notevole sviluppo delle connessioni ad alta velocità, ma la sempre crescente necessità di connessioni veloci per muovere quantità di dati sempre maggiori ha portato alla luce la necessità di trovare nuove forme di trasmissione dati alternative al ben noto paradigma *client-server*. Sullo scenario globale è emerso come miglior soluzione alternativa il paradigma *peer-to-peer*, in cui non vi è un singolo punto di centralizzazione e di smistamento dati, ma ogni nodo della rete (*peer*) contribuisce alla diffusione delle informazioni.

La distribuzione del carico di lavoro sui nodi mostra, tuttavia, alcune problematiche relative all'eterogeneità delle infrastrutture di rete dei singoli *peer*, che possono generare colli di bottiglia nella trasmissione, rallentando la diffusione di alcuni segmenti di dati. Pertanto, non è pensabile applicare gli stessi criteri di carico sulla banda di tutti i nodi, rendendo necessario un meccanismo che risulti adattivo e, possibilmente, dinamico, nel senso che si possa non solo valutare staticamente le performance di un nodo al momento del suo arrivo nella rete, ma anche modulare il contributo che esso deve dare sulla base di parametri dinamici, come ad esempio la congestione della rete in un dato istante temporale o la distanza dei peer in termini di routing.

Una delle soluzioni alle problematiche sopra elencate e su cui la ricerca sta lavorando negli ultimi anni è l'avvento dei cosiddetti 'protocolli di gossip', la cui caratteristica peculiare è quella di cercare di ridurre l'overhead sui nodi ritrasmettendo i pacchetti ricevuti secondo una logica parzialmente o totalmente probabilistica. In questa tesi sono stati presi in esame alcuni dei principali algoritmi di gossip presenti in letteratura, evidenziandone pro e contro. Sulla scorta delle idee da essi veicolate è stato progettato, implementato e simulato un nuovo algoritmo, di cui sono qui riportati i risultati sperimentali.

Introduction

The last few years have seen an exponential increase [1] in the number of devices connected to the Internet using wired or wireless reliable high-speed connections. This led to the evolution of different types of large networks: peer-to-peer (P2P) networks, social networks etc. The main goal of these networks is spreading information, usually under several constraints (e.g., real-time applications) within huge groups of nodes. Focusing on applications like video streaming or Massively Multiplayer Online Games (MMOGs), there are mainly two architectures used to disseminate the data within the network: client-server and the aforementioned peer-to-peer.

In a client-server architecture, the content is stored in a single machine or in a cluster: a client requests some content to the server, which then answers with the data stream. This solution shows several weaknesses: for example, if an Internet content provider streams a TV show to a large number of clients, it's required a server farm with high bandwidth, which means considerable maintenance costs. This architecture is also prone to failures and malicious attacks like Distributed Denial of Service (DDoS), even with machine redundancy. On the other side, a peer-to-peer architecture can overcome these limits with its naturally decentralized attitude: every client is a 'peer' and has to make a contribution to the spread of information. In the previous example, the broadcaster should only stream the content to a small part of the clients, which would spread it all over the network. Such an architecture can drastically reduce the cost of content dissemination for a content provider and it turns to be more resilient to node failures and malicious attacks.

Peer-to-peer networks are highly dynamic, in the sense that nodes can join or leave the network at any time, and lean usually on heterogeneous network infrastructures, which are often not reliable and resource-constrained in terms of communication, computation and sometimes energy resources. All these reasons focus one of the main aspects of efficient information dissemination: the algorithm behind the network.

A straightforward but inefficient way to disseminate information network wide is pure flooding protocol in which upon the first reception of a message, every site of the network relays it once to its respective neighbors. In this case a very large number of messages may be generated, which entails broadcast storm problems [3]. Therefore, the algorithm must possess certain properties to be implementable in such networks. First of all, it should be *topology-independent*, i.e. it should perform well on almost every topology (random graph, scale-free etc). Second, it should be *scalable*, in order to be applied to large-scale networks. Third, it should be robust against network dynamics and should not require synchronous message exchange. Finally, it should utilize minimal computational and communication resources. These constraints naturally lead to 'Gossip' algorithms: the key idea is that every node forwards the message received to its neighbors with a given probability. For a complete and detailed introduction to this topic, see [2].

The remainder of this document is organized as follows: chapter 1 describes the data dissemination problem and how gossip addresses it; chapter 2 analyzes the main gossip approaches and introduces Degree-dependent Dynamic Gossip, a novel gossip protocol; chapter 3 presents testbed and experimental evaluation results of Degree-dependent Dynamic Gossip; chapter 4 describes future work and possible improvements. Finally, concluding remarks are reported in the last chapter.

Contents

List of Figures

List of Algorithms

Chapter 1

Data dissemination and gossip

Data dissemination on peer-to-peer networks has been analyzed through different perspectives. This thesis will address *gossip* (or epidemic) *algorithms*, which are one of the feasible solutions for this problem, by showing existing alternatives and proposing improvements.

1.1 The data dissemination problem

The spread of large-scale complex networks brought to light the need for efficient information spread. Client-server architecture is often too expensive, thus inappropriate to bear a massive network load and a highly dynamic number of clients. In this scenario a decentralized network topology (i.e. P2P networks) is often needed.

Data dissemination (or information dissemination) problem involves the delivery of one or more messages to *all* the nodes within the network. The communication could be *one-to-many* or *many-to-many* (messages are generated by a single node or many nodes), and it's important that the transmission has low latency and as little network overhead as possible. The most trivial way to achieve this task is for every node to recursively *broadcast* each message to its neighbors. This solution leads to heavy network load and very high latency, making it unfeasible for almost any application (e.g., online games or video streaming).

Gossip protocols are one of the possbile ways described in literature, aimed at minimizing communications on the network while maximizing message reception rate of the clients (possibly 100%).

1.2 Data dissemination through gossip

Gossip algorithms have recently gained popularity as a potentially effective solution for disseminating information in large-scale systems, especially P2P systems deployed on Internet or ad-hoc networks [4].

The spread of a message in a network through gossip follows the same pattern of contagious diseases in human populations: nodes pass data to randomly chosen neighbors the same way as infected individuals spread a virus to those with whom they come into contact.

Most gossip algorithms are extremely simple, the implementation involves a few lines of code and every node runs the same algorithm. Furthermore, gossip protocols show inherent scalability and resilience to *node churn* (node failures). This feature comes from gossip's natural behavior: single or multiple node failures have no impact on actions taken by other nodes while spreading messages (i.e., if a node is unable to receive a message, other nodes won't bother to send it again). Flexibility and simplicity are the most interesting aspects of these algorithms and the reasons which led to the analysis performed in this work.

When designing an epidemic algorithm is very important to consider practical constraints such as limited network or node resources, in order to contemplate realistic scenarios. The following lines summarize the main issues presented in [4]:

- *Membership*: every node that receives a message can forward it only to other *known* nodes. This knowledge means storing and updating data structures in every node. Because of this, the criteria behind membership information impact the performance of data dissemination and the scalability of the algorithm itself. Broadcast algorithms imply a complete knowledge of the network (every node owns membership information of the whole network). This is possible as long as the size of the network is limited, because the storage needed grows linearly with it. A solution to improve performances is to provide the nodes with a partial view on the system, a limited subset of the whole network. It's also possible to include membership information to the dissemination (*piggybacking*), so a node can update its membership list while receiving messages.

- *Network awareness*: the concept of membership doesn't involve any kind of knowledge about network *topology*, so every node can freely communicate with every other known one. This lack of communication constraints can impose a high load on the network, limiting the application of gossip on Internet-wide settings. A feasible solution is to create some kind of *hierarchy* to reflect the topology of the network and then to ensure that messages are mostly forwarded to nodes within the same branch of the hierarchy.

- *Buffer management*: in a realistic scenario, the dimension of the node buffer is limited. Depending on broadcast rate, it might be insufficient to forward every message enough times to achieve acceptable reliability. Dropping new messages when buffer is full prevents the data from being correctly disseminated, while dropping old messages in favor of new ones could result in some old messages not being forwarded a sufficient number of times. One solution is to assign priorities to messages, for example based on the number of times a message has already been forwarded (*age-based prioritization*).

- *Message filtering*: it might be possible that not all nodes are interested in receiving all messages. In this case, the algorithm should partition nodes in groups with the same 'interest' and then disseminate messages only to interested nodes. An alternative is to allow nodes to express a preference towards messages and make sure they receive the appropriate ones, increasing the probability to receive a message they are interested in and simultaneously decrease the probability to receive a not interesting one. Implementing this, however, is not trivial.

Because of their abstraction level, the algorithms presented and proposed here often don't have to deal with some of the points above. The key ideas, however, make them implementable and applicable in real scenarios. More details will be presented in the following chapters.

1.3 Introduction to gossip algorithms

The essence behind gossip is *probabilistic dissemination*: the source of the message sends it to all its neighbors, then every receiving node retransmits it recursively to some of its neighbors, according to a probability distribution. This idea laid the foundation for a large part of the algorithms presented in literature.

The work in [3] evaluates the performances on different network topologies of three gossip strategies, which can be considered the most simple examples of this data dissemination approach:

- *Fixed Fanout Gossip* (*GossipFF*): this algorithm takes as input the number n (*fanout*) of neighbors to send the message to. Upon the reception of a message, the node randomly chooses n neighbors and sends them the message. Notice that, if n (being fixed) is greater than or equal to the biggest number of neighbors of every node, GossipFF becomes a pure flooding algorithm.

- *Probabilistic Edge Gossip* (*GossipPE*): here the input is a probability parameter p. The receiving node randomly chooses those edges over which the message should be transmitted with regard to the value of p. Even in this case it's worth noticing that if $p = 1$ for all nodes, it becomes a flooding algorithm.

- *Probabilistic Broadcast Gossip* (*GossipPB*): as in GossipPE, the input parameter of this algorithm is a probability p. In this case, p represents the probability that the receiving node broadcasts the message to *all* its neighbors. In particular, when $p = 1$ GossipPB becomes the flooding algorithm.

The algorithms above are at the basis of more complex solutions, for example adaptive protocols, which will be described in the next chapter.

1.3.1 Gossiping strategies

The algorithms introduced in the previous paragraph differ because of message retransmission technique, but in all cases the nodes react just after the reception of a message. However, nodes may behave in four different ways [7]:

- *Eager push*: nodes send messages to random selected peers as soon as they receive them for the first time.

- *Pull*: periodically, nodes query random selected peers for information about recently received messages. When they gain knowledge of a message they haven't received yet, they explicitly request to that neighbor the message.

- *Lazy push*: when a node receives a message for the first time, it gossips only the message identifier and not the full payload. If peers receive an identifier of a message they have not received, they make an explicit pull request.

- *Hybrid*: gossip is executed in two distinct phases. A first phase uses push gossip to disseminate a message in a best-effort manner. A second phase of pull gossip is used in order to recover from omissions produced in the first one.

These approaches have pros and cons, depending on the context which they have to be applied to. For example, eager push strategies produce more redundant traffic but they also achieve lower latency than pull strategies, as pull strategies require at least an extra round trip time to produce a delivery.

1.3.2 Hidden assumptions

An important aspect to consider when implementing every kind of algorithm is to clarify and describe all the assumptions under which it is developed. In particular, communication algorithms have to cope with the infrastructure on which they will run. This means possible bandwidth restrictions, node churn, limited buffer size, asynchronous message exchange and so on. It has been chosen to report the main hidden assumptions behind most gossip protocols, highlighted in [8].

1. *In a gossip protocol, participants gossip with one or more partners at fixed time intervals*: this is a synchrony assumption, in which the clocks of all nodes have the same rate of progress and real-time communications can't be arbitrarily slow. This is obviously unrealistic because in real situations there could be several bottlenecks, such as network congestion or difference between CPUs processing time. Moreover, real networks could be targeted by malicious attacks like DDoS, which would increase network latency and invalidate synchrony assumption.

2. *There is a bound on how many updates are concurrently propagated*: this assumption relates to the amount of resources that nodes need to provide and it may impact gossip's scalability. It can be expressed as an assumption of *unlimited cache*. Most protocols assume that nodes have enough resources to compute and store the data and gossiping doesn't exceed this limit. As a consequence, network overload caused by dynamic message production rate or issues like DDoS is not taken into account.

3. *Every gossip interaction is independent of concurrent gossiping between other processes*: this assumption involves an implied node communication independency, i.e. message losses are unrelated to each other. This is not completely realistic, because gossip algorithms run on a physical infrastructure, so a single link failure can result in a large amount of dependent message loss. Even in the absence of link failures, losses can happen for other reasons: overflows at router, difference between hosts' computation time, and external traffic.

4. *Any two processes can discover each other independently of the gossip mechanism*: this assumption hides the bootstrap problem, i.e. nodes joining the network have to discover one another in order to start the communication. Hence, a 'discovery service' is needed and it has to be continuously available. Its task is not only to help a newly joining node, but also to recover communication in case of failures: in fact, if a group of nodes is cut off from the rest of the membership due to a network failure, the network graph becomes partitioned and each partition can make progress independently. When partition resolves, gossip can repair inconsistencies, but the discovery service must be available to let nodes rediscover each other and restore regular communication.

5. *Processes select gossip partners within a round in an unpredictable random-like fashion*: as stated in the previous sections, this is the key point of gossip itself. Nodes must select their partners completely randomly. If this doesn't happen, data dissemination could be negatively affected: the mixing time could be slowed, the graph may become partitioned and the transmission could lose its inherent reliability. Therefore, it's very important to choose a proper pseudo-random number generator.

This enumeration is not intended to be exhaustive, but its aim is to focus on the main set of problems which may undermine gossip robustness in a real scenario. For a detailed discussion of limits and suitable solutions, see [8].

1.3.3 Metrics

Literature offers a wide choice of metrics to sift graph properties and evaluate performances of network algorithms. With regard to graph properties, some of the most used are the following (from [6]):

- *Degree distribution*: the *degree* of a node in an undirected graph is defined as the number of its neighbors. In a directed graph it's possible to distinguish between *out-degree* and *in-degree*, the number of outgoing and incoming edges. The *degree distribution* is the probability distribution of these degrees over the whole graph.

- *Average path length*: the shortest path length between two nodes is the minimal number of edges that are necessary to reach one node from the other in the graph. The *average path length* is the average of shortest path lengths over all pairs of nodes in the graph.

- *Clustering coefficient*: the *clustering coefficient* of a node a is defined as the number of edges between the neighbors of a divided by the number of all possible edges between those neighbors. Intuitively, this coefficient indicates the extent to which the neighbors of a are also neighbors of each other. The clustering coefficient of the graph is the average of the clustering coefficients of the nodes, and always lies between 0 and 1. For a complete graph, it is 1, for a tree it is 0.

With regard to regards performance evaluation metrics, the points below will briefly introduce some (from [3] and [5]):

- *Coverage*: denotes the fraction of nodes which actually received the messages (higher is better).

- *Delay*: represents the average number of hops that a message traverses before reaching a node (lower is better).

- *Fraction of total infected sites*: is defined as the percentage of all nodes in the system that delivered a message generated by a source in the end of the dissemination.

- *Latency*: measures the number of hops required to deliver a message to all recipients, i.e. the number of hops of the longest path among all the shortest paths from the source to all other nodes that received the message.

- *Message complexity*: measures the mean number of messages received (or sent, if no message loss is taken into account) by each node.

- *Reliability*: is defined as the percentage of messages generated by a source that are delivered by all nodes. A reliability value of 100% is indicative that the algorithm was successful in delivering any given message to all sites.

1.4 Other uses of gossip

The idea behind gossip is that of a 'distributed' and almost random message exchange. This work focuses on gossip as a way to disseminate data within a network, either physical or virtual. However, this approach is applied to other problems both in computer science and mathematics, especially for its inherent scalability. For example, gossip algorithms are used to solve the distributed averaging problem [9, 10, 11, 12] or distributed computation of separable functions [13]. On the other side, some works analyzed gossip as a suitable protocol for failure detection [14], ad-hoc routing [15], network size estimation [16], load balancing [17, 18], and communication optimization in wireless [19] and sensor networks [20, 21].

Chapter 2

Gossip algorithms on overlay networks

The results of research on gossip algorithms offer a wide variety of solutions for efficient and reliable information dissemination. As stated before, algorithms differ because of gossiping technique and assumptions about message exchange, nodes' properties or network graph knowledge. Most algorithms described here rely on few assumptions, such as limited or absent *membership management services* and partial *network awareness*. Some of them are also the main source of inspiration for Degree-dependent Dynamic Gossip (DDG), an evolution of the basic *eager push* approach, enriched by a degree-dependent adaptivity. DDG has been designed to address the problem of efficient and reliable message delivery on overlay networks, thus neglecting low-level issues like routing, which is assumed to happen only along shortest paths. More details as well as testbed and simulation results of DDG will be presented later.

2.1 Theoretical notions and basic schemes

There are several gossiping strategies, but all hark back to three main ones: *push*, *pull* and *hybrid* (push-pull). In order to better understand the behavior of each strategy, it's worth introducing the algorithmic approach of gossip by means of epidemiology, as carried out in [23]. According to this terminology, when a single message is created, each node can be in one of these three states:

- *Susceptible* (S): the node doesn't know about the message.
- *Infected* (I): the node knows the message and is actively spreading it.
- *Removed* (R): the node has seen the message, but is not participating in the spreading process (in epidemiology, this corresponds to *death* or *immunity*).

These states are related to the behavior of a node with regard to a single message. In the presence of multiple concurrent messages, each node is in a different state for each message. Two known models are SI and SIR, which differ in the number of possible states.

2.1.1 The SI model

In the SI model, each node can be susceptible or infected. Once infected, a node cannot change its state anymore.

Algorithm 1 SI gossip

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

Algorithm 1 (taken from [23]) represents the generic SI procedure. The active thread (lines 1 - 10) is executed every Δ time units. The parameter p gets the value retrieved by a *random peer sampling* procedure. Statements at line 5 and 8 trigger two other procedures: in the first case, the message sent to node p is stored in its cache (line 12), thus making p switch to state I; in the second case, an update-request executed by a node q makes p, if in state I, send updates to q (line 17). The two boolean parameters *push* and *pull* describe the dynamics of the algorithm. Depending on these parameters, we can talk about push, pull, and push-pull gossip. In push gossip, susceptible nodes are passive and infective nodes actively infect the population. In pull and push-pull gossip each node is active [23].

2.1.2 The SIR model

According to the SI model, nodes continue forwarding useless updates endlessly, because they don't have memory of already forwarded messages.

Algorithm 2 SIR gossip

```
 1: loop
 2:     wait($\Delta$)
 3:     $p \leftarrow$ random peer
 4:     if push and in state I then
 5:         send message 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 state 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 message to $m.sender$
26:     end if
27: end procedure
```

The SIR model faces the termination problem considering the *age* of each message and basically discarding too 'old' messages, thus stopping their propagation. In other words, each node switches to the *removed* state in relation to a message when its age crosses a threshold value. Lifespan of messages represents a tradeoff between complete coverage and redundancy. In fact, the expected result of a gossip protocol is a 100% coverage with as little network overhead as possible, that is absence of redundant messages.

Algorithm 2 shows the SIR gossip variant. It's the same as SI gossip, but it shows a different behaviour in procedure *onUpdate* (line 15). The first reception of a message makes the node switch to state I. A new reception of the same message has a probability of $1/k$ to make the node switch from state I to state R, which means *immunity.* If this happens, the node is not able to forward that message anymore, neither by pushing (line 4) nor answering to an update request (line 24). Thus, a correct tuning of the value k is crucial for good performances.

SI and SIR are the basic propagation models and they can be implemented using push, pull or push-pull approach.

2.2 Push algorithms

Push protocols are based on the recursive forwarding of messages among peers. A node receiving a message actively passes it on to a few random other nodes, which recursively do the same until some termination condition is met. The termination condition ensures that the recursion does not go on forever. For instance, messages could be augmented by a Time-to-Live (TTL) field to limit the number of hops they can take. Alternatively, nodes could be programmed to forward messages only upon their first reception and ignore subsequent copies [22].

In this simple approach, nodes aren't aware of messages received by their neighbors from other nodes. It may result in a retransmission of 'old' messages, which have already spread on a specific portion of the network, thus fruitlessly increasing overhead. Furthermore, this ‘massive’ propagation doesn’t ensure a quick complete coverage. In the first steps of propagation, one or two random forwards are enough to reach a not-yet-informed node, but this number increases considerably for the last few nodes. Assume a generic push model, in which nodes are selected uniformly at random and one at a time. At each iteration, the message is forwarded to the selected node, whether it is already informed or not. Under these assumptions, the expected number of times a message should be forwarded to reach all n nodes is in the order of $O(n \ln n)$ [22]. However, this is an approximated evaluation, because gossip itself isn’t able to ensure a complete coverage as well as a restrained redundancy, especially in real scenarios.

2.2.1 Fixed Fanout Gossip

This is probably the simplest eager push gossip algorithm. After the reception of a message, each node forwards it to a fixed number (*fanout*) of neighbors. If the fanout value is greater than or equal to the number V_i of

Algorithm 3 GossipFF

```
Require: $message$, $fanout$
 1: if $fanout \geq V_i$ then
 2:     $toSend \leftarrow \Lambda_i$
 3: else
 4:     $toSend \leftarrow \emptyset$
 5:     for $f = 1$ to $fanout$ do
 6:         random select $s_j \in \Lambda_i / toSend$
 7:         $toSend \leftarrow toSend \bigcup s_j$
 8:     end for
 9: end if
10: for all $s_j \in toSend$ do
11:     Send($msg$, $s_j$)
12: end for
```

neighbors of node s_i (denoted Λ_i), the message will be obviously forwarded to all of Λ_i elements. Otherwise, the list *toSend* is filled with $fanout$ random nodes taken from Λ_i (lines 5-8). If the condition of line 1 is true for each node, GossipFF becomes a pure broadcast algorithm.

The analysis performed in [3] on Bernoulli (Erdös-Rényi), scale-free, and random geometric graphs shows that network topology strongly influences performances of gossip algorithms. GossipFF turns to be the best choice in terms of infected sites, reliability and latency on random geometric graphs, whereas it performs badly on scale-free graphs.

2.2.2 Probabilistic Edge Gossip

In this case, node s_i randomly chooses the edges over which the message will be forwarded. Unlike GossipFF, the second input parameter is a fixed

Algorithm 4 GossipPE

```
Require: $message$, $p_e$
1: for all $s_j \in \Lambda_i$ do
2:     if Random() $\leq p_e$ then
3:         Send($msg$, $s_j$)
4:     end if
5: end for
```

probability. The function *Random*() generates a random number in $[0, 1]$. If this number is less than or equal to p_e for a given edge, the message is forwarded on that edge.

The performances of GossipPE on random geometric graphs are not as good as GossipFF, whereas it performs well on graphs with high degree variance and low edge dependency such as scale-free networks. Instead, the behavior of GossipFF and GossipPE on Bernoulli graphs is similar.

2.2.3 Probabilistic Broadcast Gossip

This is a variation of pure broadcast, in which the node randomly chooses whether broadcasting the message to its neighbors. Even in this case, the

Algorithm 5 GossipPB

```
Require: $message$, $p_v$
1: if Random() $\leq p_v$ then
2:     for all $s_j \in \Lambda_i$ do
3:         Send($msg$, $s_j$)
4:     end for
5: end if
```

function *Random*() generates a random number in $[0, 1]$. If this number is less than or equal to p_e, the node broadcasts the message to all its neighbors.

GossipPB shows the same performances of GossipPE on scale-free and random geometric graphs. On Bernoulli graphs, it behaves the same way as both GossipFF and GossipPE.

2.2.4 RingCast

A viable solution to disseminate data within networks other than gossip is *deterministic dissemination*. This technique doesn't focus on optimization of message forwarding, but on network topology. In fact, deterministic algorithms build an *overlay network* and spread messages on it by means of flooding. The requirement to ensure a complete dissemination starting from any node is to form a strongly connected graph including all nodes. Many topologies have been proposed, showing different results in relation to the metrics described in the previous sections. For example, spanning trees are optimal with respect to message overhead, but a single link failure in a non-leaf node disconnects the tree.

The drawback of deterministic approaches is that reliability is achieved by imposing a fixed structure on overlays, which is unfeasible in massive-scale dynamic networks. In the end, probabilistic protocols are good at spreading messages very quickly, but they don't ensure reliability, whereas deterministic algorithms are reliable, but they don't scale well.

RingCast [24] addresses data dissemination problem by mixing probabilistic and deterministic strategies. It establishes two types of links among nodes, namely random links (*r-links*) and deterministic links (*d-links*). The set of d-links forms an overlay network with a global bidirectional ring structure, which constitutes a strongly connected graph. This topology is compliant to the deterministic protocols' requirement, thus a complete dissemination is guaranteed. Instead, r-links are links randomly selected by a membership management protocol.

After generating or receiving a new message, the node forwards it to its two ring neighbors (across two outgoing d-links) and to other peers across $F-2$ randomly selected r-links, being F the system-wide fanout parameter. If the message has been received through a ring neighbor, the node relays it across the other d-link and selects other $F-1$ random r-links. The overlays are built using epidemic protocols too: r-links are sampled by means of CYCLON [25], an epidemic protocol that is an instance of the Peer Sampling Service [26]; d-links are maintained using a proximity-based topology construction epidemic protocol, Vicinity [27].

2.3 Pull algorithms

In a pull protocol, each node periodically probes random peers in the network hoping to reach an already informed peer, and retrieves new messages when available. Typically, during a pull round, random pairs of peers exchange information about the messages they have recently received and request missing messages from each other [22]. However, this technique isn't often very effective in terms of latency, because messages are forwarded only during periodic pull rounds.

2.3.1 CREW

CREW [29] is a pull-based algorithm which tries to address the problem of *flash dissemination*, that is rapid dissemination of varying amounts of information to a large number of recipients in a very short period of time, for example a service which provides accurate and timely information about seismic events. In this kind of situations, the events to be monitored are unpredictable and communication has to be efficient and it can't be scheduled. Moreover, the number of entities which have to get the information is not fixed and the underlying network may be heterogeneous in bandwidth and latency, especially if end receivers are geographically distributed.

In the basic version of CREW, every message is divided into chunks, each of them having a unique chunk-ID. The list of chunk IDs is called *metadata*. Metadata about the chunks are known by all nodes before they start gossiping. Instead of being randomly pushed, chunks undergo a pull logic: a pull-initiator node sends out the list of the IDs of already received chunks to a target node, selected uniformly at random. The target node then sends one chunk that the initiator does not have, chosen randomly. If the target node has no 'missing' chunks, it sends an error message. Once a node receives all chunks listed in the metadata, it immediately stops gossiping. When all nodes stop gossiping, each node has all chunks, thus achieving a complete deterministic dissemination. For all the extensions of basic CREW algorithm and for further details, see [29].

2.4 Hybrid algorithms

Push protocols allow an exponential spread of data in the first steps of dissemination, but after some time the rate of the dissemination diminishes and the cost of reaching uninformed nodes increases dramatically, due to higher amount of redundant messages. On the other side, pull protocols show a slow initial progress of message dissemination. However, once a message has reached a sufficient number of nodes, it quickly spreads to all the remaining ones. Therefore, push is an eccellent candidate for the early stages of dissemination, when a fast dissemination is needed. Pull, on the other side, appears good for the final stages, because it succeeds in delivering messages to all remaining nodes without overloading the network with useless redundant messages. Hybrid protocols try to combine the best of both push and pull worlds.

2.4.1 PULP

PULP [22] blends push and pull, trying to pursue three main objectives: first, limit push to the first stages of dissemination, in order to have a good bootstrap mechanism without overloading the network with redundant messages; second, avoid useless redundant pulls, probing only when a message is known to be missing; third, adapt the pull probing frequency to match current message rate.

It assumes fully decentralized operations, asynchronous message exchanges and multiple concurrent generated messages. With regard to supporting mechanisms and technologies, PULP uses CYCLON as a peer sampling service and each node maintains a partial view of the network, periodically updated by exchanging some links with other peers. Nodes need to have also a rough estimate of the network size, provided by the interval density algorithm [28].

The dissemination provided by the first push phase is strongly influenced by TTL and fanout parameters. In PULP, these two values are strictly dependent, one of them is fixed and the other is derived accordingly. Push stage is used to reduce probing requests of pull one. In fact, forwarded messages by the push component carry information about which other messages are available, helping the pull phase.

Each node maintains a history of recently received messages and a trading window, a list of messages available to other nodes on request. When a node N generates a message or receives it for the first time, it registers the message in its history and, if TTL threshold has not been reached, forwards it to *fanout* random peers. Furthermore, the message is piggybacked by the IDs saved into N's trading window. Each receiving peer checks for messages not contained in its own history. If it discovers some messages it has missed, it inserts them in the missing set. These messages will be asked for by the periodical pull thread, which simply selects a random peer and sends it a pull request.

The adaptation of the pulling frequency is performed by a separate thread, which periodically monitors the number of useful and useless pulls that were performed during that period and the dimension of the set of messages which a node has heard of, but it has not received (*missing* set), in the same period. If the size of the missing set is too large, the adaptation thread lowers the period of pull thread. On the other hand, if the size of missing set shrinks, the evolution depends on useless and useful pull operations ratio.

2.5 Adaptive algorithms

It's possible to characterize a fourth additional family of algorithms, namely *adaptive gossip.* The concept of adaptivity can be applied both to push and pull schemes and involves the response of the system to the dynamics of real networks, such as churn, network topology evolution or communication rate variability.

The adaptive algorithm presented in [5] focuses on employing gossip to improve performances in MOGs, where responsiveness and scalability are the main aspects to be taken into account. It exploits the typical behavior of MOG players, who commonly generate game events according to some inter-generation probability distribution between successive moves, thus making an optimization of message distribution possible. It considers a MOG system in which peers communicate through an arbitrary overlay. Messages are distributed through the overlay and peers which are not directly connected must exploit multi-hop communication. Moreover, each peer knows the list of peers interacting in a given area of the virtual world and maintains statistical information on received messages for each other peer.

The algorithm is a basic push scheme, in which each node forwards new information (either received or generated) to a subset of its neighbors according to a dissemination probability. The adaptivity lies in the variation of this probability: in fact, as soon as a node p observes that it is receiving messages from another peer q at a rate lower than expected, it asks the neighbor n, from which it usually receives messages originated from q, to increase its dissemination probability of game events. This request from p to n (*stimulus*) remains active at n for a limited period of time, then the dissemination probability returns to the original value. This main idea has been converted into three variants of the same algorithm.

2.5.1 Stimuli associated to receivers

The gossiping procedure executed by each node (algorithm 6) resembles basic push: when a node p receives or generates a new message, forwards it to each neighbor n (except to the original sender, if the message has been received) with a probability v_n. Initially, all values v_n are set to a constant value v_0, but they are periodically updated by a monitoring procedure (algorithm 7) running on every peer.

This procedure monitors the reception rate of game events originated at each node of the overlay, exploiting the information stored in the node. As soon as p observes a lower game event reception rate from a peer j, it selects the neighbor q from which it usually receives messages containing game events generated by j; then, p sends q a stimulus message to request the increase of the value of p stored at q. This stimulus decays over time, i.e. its effects terminates after a deadline and v_p gets back the default value v_0.

For further details on procedures ComputeThreshold and RetrievePeersLowRate, see [5].

2.5.2 Stimuli associated to generators

The second algorithm adapts the dissemination threshold in a different way (algorithm 8). Each peer p maintains an array of dissemination thresholds, one for each node in the network. As soon as a new message generated by s has to be disseminated by p, a threshold $\gamma_s \leq 1$ is computed for every

Algorithm 6 Adaptive gossip with stimuli associated to receivers: gossiping procedure executed by p

```
Require: $message$ generated at $p$ or received from a peer $q$
1: $N_p \leftarrow p$'s neighbors $\setminus$ $q$          ▷ $q$ = NULL if $message$ originated at $p$
2: if $message$ is a duplicate then
3:     Return
4: end if
5: for all $n \in N_p$ do
6:     currentTime ← GetTime()
7:     $v_n$ ← ComputeThreshold($n$, currentTime)
8:     if Random() $< v_n$ then
9:         Send($message$,$n$)
10:    end if
11: end for
```

Algorithm 7 Adaptive gossip with stimuli associated to receivers: monitoring procedure executed by p

```
1: loop
2:     Sleep(monitoringPeriod)
3:     peerList ← RetrievePeersLowRate()   ▷ Retrieve peers with low reception rate
4:     for all $j \in$ peerList do
5:         $q$ ← Forwarder($j$)          ▷ Neighbor that sends messages from $j$
6:         Send($q$,'low rate from $j$')
7:     end for
8: end loop
```

neighbor of p. The value γ_s is used to determine whether the message has to be gossiped to a given neighbor.

Algorithm 8 Adaptive gossip with stimuli associated to generators: gossiping procedure executed by p

```
Require: $message$ generated at $p$ or received from a peer $q$
 1: if $message$ is a duplicate then
 2:     Return
 3: end if
 4: $N_p \leftarrow p$'s neighbors $\setminus$ $q$        ▷ $q$ = NULL if $message$ originated at $p$
 5: $s \leftarrow$ peer that generated $message$
 6: currentTime $\leftarrow$ GetTime()
 7: $\gamma_s \leftarrow$ ComputeProb($s$, currentTime)
 8: for all $n \in N_p$ do
 9:     if Random() $< \gamma_s$ then
10:         Send($message$,$n$)
11:     end if
12: end for
```

2.5.3 Stimuli associated to generators and receivers

The third variant is derived from the previous one and uses a different mechanism to adapt gossip threshold, while the rest of the algorithm remains the same. Each peer maintains a set of arrays of dissemination thresholds, one for each neighbor. In this case, each stimulus changes the probability of disseminating the messages originated by a specific node which should be forwarded to a given neighbor. The aim of the protocol is to generate a higher amount of more specific stimuli.

2.6 Degree-dependent Dynamic Gossip (DDG)

The algorithm proposed in this work belongs to the family of eager push protocols. It combines a probabilistic component with a deterministic one, because each message is spread according to a certain probability, which is decided for each node in a dynamic and deterministic way. It is assumed that the algorithm runs on an overlay network, in which each node can communicate only with its neighbors in the network graph.

DDG tries to exploit the characteristics of overlay networks to make nodes gain awareness of their rough position within the graph (i.e., their distance from the center) and the 'importance' of their contribution to data dissemination. This awareness is obtained by making each node exchange degree information with its neighbors, in order to better tune dissemination probability.

After the generation of a new message (algorithm 9), node p computes the value of its degree and attaches it to the payload of the message - shown in line 2 with *struct* notation of C language - then sends it to all its neighbors. Algorithm 10 describes the gossiping procedure executed by node p upon the reception of a message from a neighbor q: if the message msg is not known, i.e. its ID isn't stored in p's cache, node p retrieves the value of q's degree, which is stored together with the payload of the message, and saves it in q's position of its neighborhood array, denoted Λ_p^q. In the following part of the procedure, node p forwards the message to its neighbors with a probability γ_n which is computed according to the degree of each neighbor n of p (lines 6-10). Then, a random threshold is computed, and if it's less than or equal to γ_n, p attaches the value of its own degree to the message and forwards it to node n.

In the early steps of the algorithm, neighborhood information is missing, thus p is forced to compute forwarding probability by selecting the recipients of the message uniformly at random (line 7). However, after the initial bootstrap phase, neighbors' data are filled by means of information piggybacked on messages, and the algorithm can start working properly. The core of nodes' selection is the function ComputeProbability, whose details will be described later.

The key point of gossip algorithms is that each node should forward messages in order to achieve the highest possible coverage with the lowest possible network overhead (in this case, represented by the amount of redundant messages). A few assumptions on network structure or nodes' global knowledge make this goal be harder to reach, but they allow the algorithm to be more suitable to real network infrastructures. DDG tries to solve data dissemination problem on P2P networks by tuning dissemination probability to nodes' neighbors degree. The rationale behind DDG is that nodes with a low degree 'compensate' their little amount of links with a higher reception probability (i.e., neighbors increase their dissemination probability), whereas nodes with high degree have a higher probability to receive a message from one of their neighbors, thus reception probability can be safely lowered. This last countermeasure is taken to avoid a flood of redundant messages.

As it's easy to see, this algorithm allows an automatic and continuous monitoring of network status. Assuming that each node can easily know if one of its neighbors is online or offline, node churn doesn't have a strong influence on message propagation, because dissemination probability is consequently tuned - obviously unless a link or node failure disconnects a whole portion of the network overlay graph. With regard to assumptions (section 1.3.2) and constraints (section 1.2), message creation can be arbitrarily asynchronous, multiple messages can be generated by different nodes at the same time, and no assumptions on nodes' buffer size are made. Moreover, no membership management service is needed, because each node dynamically builds its local membership information on its own.

Algorithm 9 Degree-dependent Dynamic Gossip: generation of a new message executed by node p

```
1: $msg$ = GenerateMessage()
2: $msg.degree$ =Size($\Lambda_p$)
3: for all $n \in \Lambda_p$ do
4:     Send($msg$, $n$)
5: end for
```

Algorithm 10 Degree-dependent Dynamic Gossip: gossiping procedure executed by node p

```
Require: $msg$ received from neighbor $q$
1: if $msg \in cache_p$ then
2:     return
3: end if
4: $\Lambda_p^q \leftarrow msg.degree$
5: for all $n \in \Lambda_p \backslash \{q\}$ do
6:     if $\Lambda_p^n = 0$ then
7:         $\gamma_n = 1/$Size($\Lambda_p$)
8:     else
9:         $\gamma_n =$ ComputeProbability($\Lambda_p^n$)
10:    end if
11:    threshold = Random()
12:    if threshold $\leq \gamma_n$ then
13:        $msg.degree \leftarrow$ Size($\Lambda_p$)
14:        Send($msg$, $n$)
15:    end if
16: end for
```

2.7 Impact of network topology on gossip algorithms

It's worth mentioning the analysis performed in [30], in which the role of overlay topology in gossiping in ad hoc networks is evaluated. Researchers modelled a static ad hoc network as a set of distributed processes communicating by message passing, defined by an undirected graph.

Tests have been conducted on several topologies, including random and scale-free graphs, using three different algorithms: Neighborhood Independent Strategy (i.e., Fixed Fanout Gossip), Standard Gossiping Strategy (i.e., Probabilistic Edge Gossip), and Neighborhood Dependent Strategy (NDS). This last protocol is similar to Probabilistic Edge Gossip, but nodes are separated into two sets and each set has a different dissemination probability, based on nodes' degree, so that nodes with a high degree gossip with a high probability, and nodes with a low degree gossip with low probability. Moreover, a variant of NDS has been considered, in which the number of redundant messages is reduced using a blacklist of already received messages.

Simulation results have shown that scale-free topology turns to be the best in terms of dissemination speed and nodes reached when using Probabilistic Edge Gossip. Instead, NDS achieves the same results in terms of nodes reached on both random and scale-free graphs, but dissemination speed is higher in scale-free graphs. The use of a blacklist turns out to be more efficient with scale-free topology in terms of amount of sent messages. Finally, Fixed Fanout Gossip performs better on random topologies than in scale-free only with low fanout values. High fanout values increase the gain with scale-free topology, in terms of number of messages and dissemination speed.

The overall conclusion is that scale-free graphs exhibit the most favorable topology for information dissemination through the most common gossip protocols. For further information, see [30].

Chapter 3

DDG: testbed and performance evaluation

The probabilistic component of gossip protocols doesn't often allow an a priori analysis of computational cost or upper bounds and lower bounds for forwarded messages or dissemination latency. These issues become even bigger if algorithms are supposed to run on heterogeneous network configurations with unpredictable message creation rate. Therefore, the best way to analyze the behavior of gossip algorithms is to test them under several network configurations and evaluate their performances by means of common metrics (as described in 1.3.3). DDG has been tested in the simulation environment created by Parallel and Distributed Simulation Research Group [38] (Department of Computer Science, Università di Bologna), and simulation results have been compared with those of other known gossip algorithms.

3.1 Simulation environment

DDG has been executed on a three-tier simulation environment: the core of simulation is the Advanced RTI System (ARTÌS) [32], a parallel and distributed simulation middleware, inspired by the High Level Architecture standard [31]. ARTÌS has been integrated with the Generic Adaptive Interaction Architecture (GAIA), a framework which is in charge of migrating simulation elements in the distributed environment to improve performances. On the top of this complex architecture stands the Large Unstructured NEtwork Simulator (LUNES) [34], which uses the services provided by ARTÌS and GAIA to simulate complex protocols on top of network graphs.

Parallel And Distributed Simulation (PADS) is the acronym used to refer to execution of concurrent simulation processes over tightly coupled, or loosely coupled computation architectures, respectively [32]. In other words, a parallel and distributed simulation environment is composed by a set of Physical Execution Units (PEUs) such as hosts, CPUs or CPU-cores, connected to a common network (e.g., shared memory, LAN, Internet).

3.1.1 ARTÌS

The complexity of simulated system under PADS approach can strongly influence performances, due to communication and synchronization services, which are used by model components (formally known as *federates*) to interact. Therefore, interprocess communication may become the main bottleneck of the distributed simulation paradigm [32]. ARTÌS performs adaptive evaluation of the communication bottlenecks. It supports both *conservative* and *optimistic* synchronization: the first is implemented with time-stepped approach and the Chandy-Misra-Bryant algorithm, while the second relies on a Time Warp algorithm implementation.

ARTÌS follows a component-based design and it is formed by a set of modules organized in a stack-based architecture. At the bottom of the architecture is located the communication layer, which can handle different network protocols as well as manage shared memory. Above it stands the runtime core layer, composed by management modules inspired by a typical HLA-based simulation middleware. Examples of such modules are Data Distribution Management or Federation Management. The user simulation layer uses the service provided by the underlying core by means of a set of APIs, namely the University of Bologna APIs. Additional orthogonal modules are

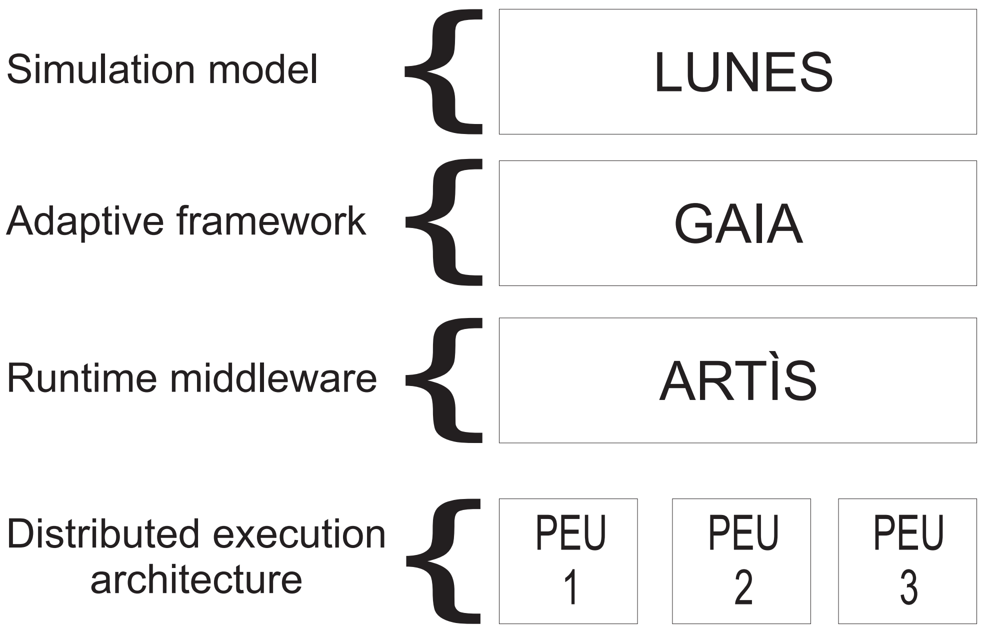

Figure 3.1: Simulation environment stack (taken from [33] and modified)

dedicated to other specific features, like the adaptive runtime management of synchronization and communication overheads [32].

ARTÌS supports the Concurrent Replication of Parallel and Distributed Simulations (CR-PADS), i.e. the concurrent execution of many independent simulation runs based on the same model definition, in order to decrease the wall-clock time required to complete a set of simulations. Moreover, a further enhancement of performances in complex simulation scenarios is achieved by both data marshalling (implemented in ARTÌS) and Intel® Hyper-Threading™ technology.

3.1.2 GAIA

As stated before, parallel and distributed simulation has to face some problems related to resource management, both in terms of network overhead and physical execution units load. Entites on the same PEU are able to communicate via low latency and low overhead networks, namely shared memory. On the other side, simulated model entities (SMEs) allocated on distributed

PEUs must communicate, for example, by means of a LAN connection, which has a higher latency. The best solution to reduce communication and synchronization overhead is to allocate the whole set of entities on the same PEU, but it turns to be the worst solution in terms of load-balancing. GAIA framework aims at reducing simulation costs by adapting model partitioning and execution architecture to runtime requirements. Basically, GAIA performs a selective migration of SMEs, to optimize both communication between distributed PEUs and load of a single PEU.

GAIA allows the reduction of communication overhead by using two heuristics: the first (base heuristic) monitors communication pattern of SMEs and spots potential candidates to be migrated, i.e. highly interacting SMEs, while the second (group heuristic) analyzes the results of basic heuristic to find and evaluate entire groups of SMEs to be migrated.

The load balancing mechanism is based on the presence of synchronization barriers during the simulation execution, which are exploited to obtain a ranking of logical processes (groups of SMEs) based on arrival to the synchronization barriers. The logical processes on top of the ranking are marked as 'fast', while the ones at the bottom are marked as 'slow'. The load-balancing mechanism triggers additional migrations to improve the balancing of the distributed system, enabling the slowest logical processes to migrate some SMEs to the fastest ones.

3.1.3 LUNES

On the top of the simulator stack (figure 3.1) there is LUNES, an agent-based discrete-event simulator. The goal of LUNES is to provide a tool for the simulation of complex protocols on large graphs of whatever topology. It is written according to a modular approach, which allows an easier integration of external tools as well as the possibility to create and implement new protocols within the simulator. The main modules reflect the phases of protocols' simulation:

- *Network topology creation*: in the current version, LUNES uses igraph

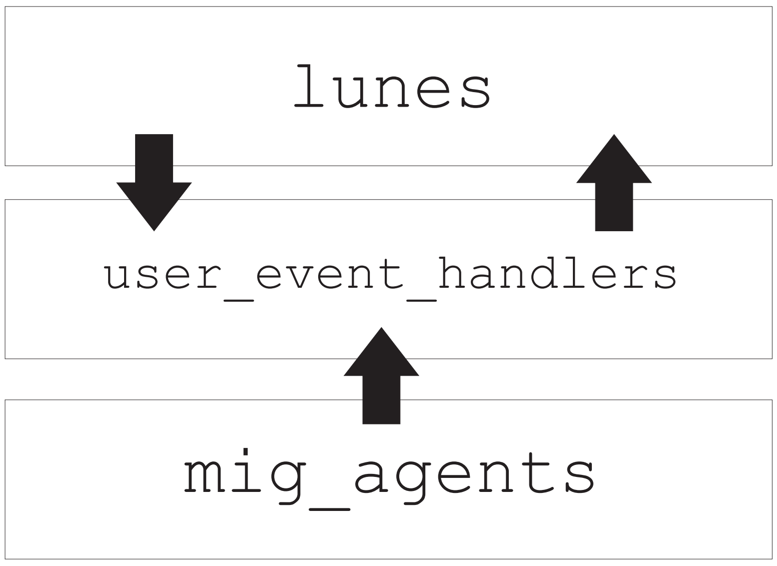

Figure 3.2: LUNES modules stack

[39] tool to create and manipulate both directed and undirected graphs. They can be used for an on-the-fly evaluation of communication algorithms as well as stored into 'corpuses' for a later use. It's worth underlining that igraph represents only one possible way to generate network graphs, and the usage of an external tool doesn't imply a static topology. Instead, during the simulation execution it's possible to modify network topology and deal with dynamic systems [34].

- *Protocol simulation*: the whole LUNES platform relies on the services provided by GAIA and ARTÌS, thus the implementation of new algorithms doesn't have to deal with low-level simulation issues. Moreover, LUNES offers a set of functions that help the user deal with the common operations of dissemination protocols (message forwarding, actions on message reception and so on).

- *Performance evaluation*: together with protocol simulation, the performance evaluation module is the most demanding, both in terms of disk space and computational resources. The simulation of a network with a few hundred nodes creates a huge amount of traces (some giga-

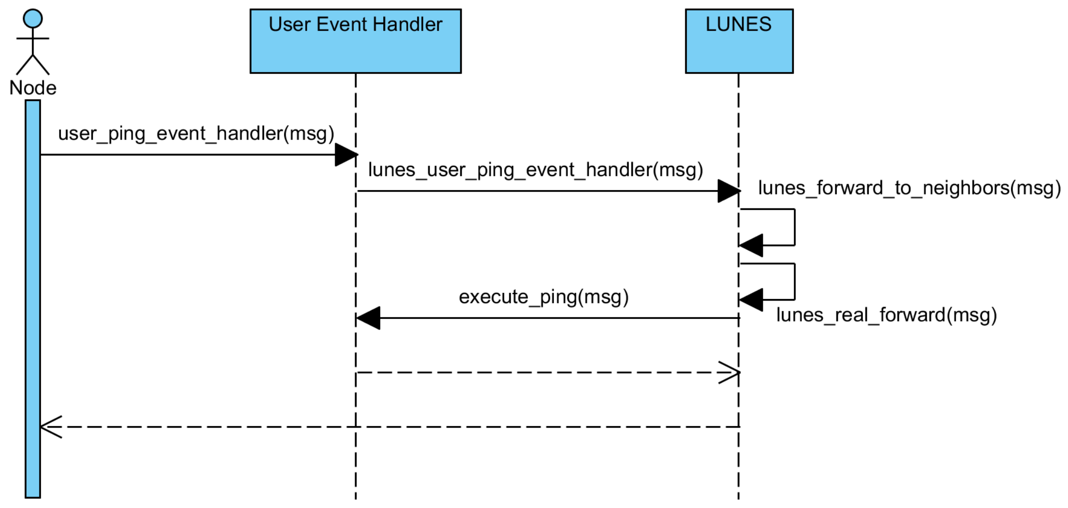
Node
User Event Handler
LUNES
user_ping_event_handler(msg)
lunes_user_ping_event_handler(msg)
lunes_forward_to_neighbors(msg)
execute_ping(msg)
lunes_real_forward(msg)

Figure 3.3: Sequence diagram of activations on ping message reception

bytes per run), which are stored in temporary folders to be parsed and analyzed at the end of the simulation. Trace analysis is implemented using a mixture of shell scripts and C language code, in order to be efficient and easily extensible.

The whole simulator is written in C language for efficiency reasons, and it's completed by a set of shell scripts which allow a batch execution of provided gossip algorithms. These scripts can be easily modified to fit whatever protocol.

As shown in figure 3.2, the core of LUNES can be roughly divided into three communicating modules, here identified by the respective source code file names. These modules use other orthogonal services, for example hash table management or trace management, which are not depicted in the above scheme.

The main simulation loop is run by `mig_agents`, which is in charge of calling the appropriate handlers after the reception of a low-level message (i.e., a message sent by GAIA). The set of functions of `user_event_handlers` represents a bridge between user simulation level and the GAIA-ARTÌS platform: in fact, events like the reception of a new message trigger a specific call to the upper layer, the `lunes` module, which handles the highest abstraction level of protocols. However, high-level `lunes` functions always call primitives of `user_event_handlers`, which communicate directly with the simulation level. For example, the reception of a new message (called *ping message*) triggers the function `user_ping_event_handler` in `user_event_handlers`, which calls the `lunes_user_ping_event_handler`. Figure 3.3 depicts the Unified Modeling Language (UML) sequence diagram of function calls after the reception of a new message: the simulation level calls the correct event handler, which communicates with user level by means of `lunes` primitives. The core of dissemination protocols lies in `lunes_forward_to_neighbors` and `lunes_real_forward` functions, which implement algorithms' dissemination rules. However, these two user-level functions rely again on user event handler module to execute the real message delivery, by calling `execute_ping`. In the end, this function is in charge of initializing message properties and sending it to GAIA layer.

3.2 Performance evaluation scenario

Degree-dependent Dynamic Gossip has been tested on the simulation environment described above. As already stated, no assumptions on network topology or nodes' knowledge have been made, thus letting DDG be a suitable solution for communication on whatever kind of network infrastructure. DDG performances have been compared with those of two of the algorithms described in section 2.2, namely Probabilistic Broadcast Gossip and Probabilistic Edge Gossip (from now on, it will be called Fixed Probability Gossip).

3.2.1 Network graphs

The tests have been executed on two graph corpuses, each of them containing 100 connected graphs, and each graph composed of 100 nodes. The corpuses differ by their topology:

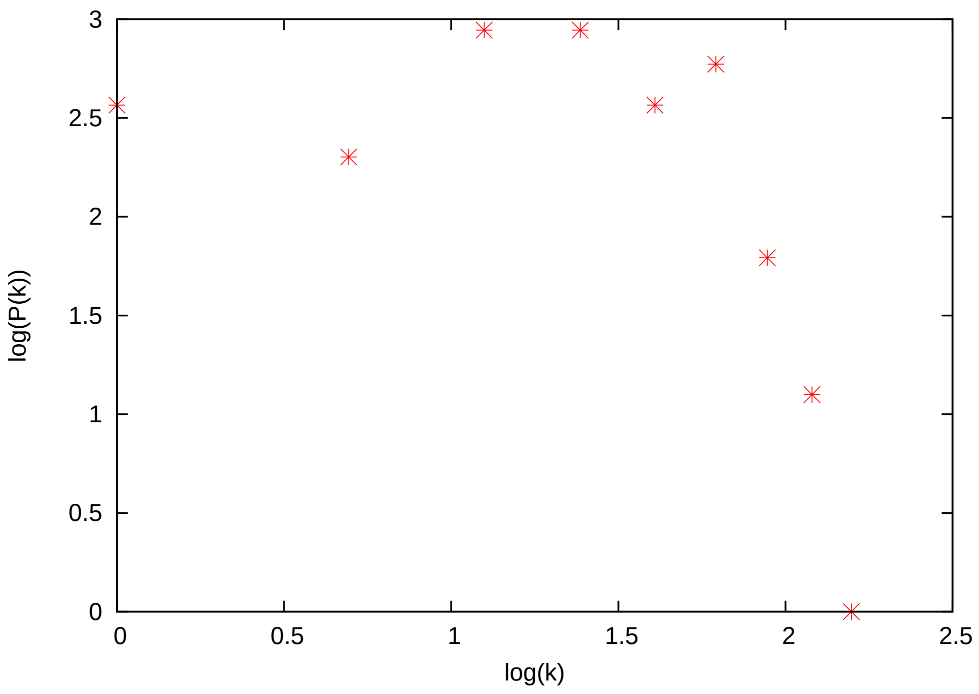

Figure 3.4: Degree distribution of a random graph (double logarithmic scale)

- *Random networks*: random graphs corpuses are included in LUNES distribution. They are built using a function provided by igraph, according to the Erdös-Rényi model [35]. The graphs of the corpus have the following characteristics: each node has two edges, that is 200 edges in the whole network, without self-loops, and graph diameter is 8.

- *Scale-free networks*: scale-free graph corpuses aren't included in LUNES package. For these simulations, ad-hoc scale-free corpuses have been created according to the Barabási-Albert construction model [36] by using a modified version of the tool developed and described in [37]. Three corpuses have been generated, each of them having different starting graph dimension; more precisely, fully connected graphs of 3, 5 and 7 nodes respectively.

Figures 3.4 and 3.5 depict the degree distribution of a sample of both random and scale-free corpus' graphs in double logarithmic scale. Figure 3.4

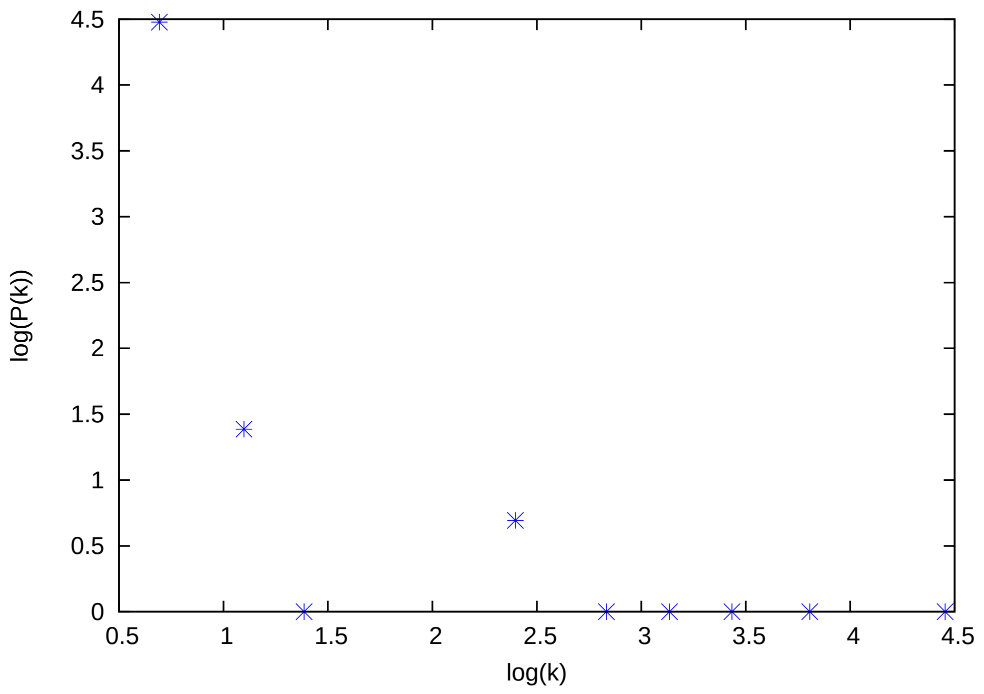

Figure 3.5: Degree distribution of a scale-free graph (double logarithmic scale)

clearly shows that, except some outliers, the degree of nodes varies within a narrow range. The workload in real networks with random topology is equally shared among all peers, but such networks are also more prone to partitioning after random failures.

On the other side, figure 3.5 highlights the main topological peculiarity of scale-free graphs, that is a minimum amount of highly connected nodes (*hubs*) against a large part of nodes with a low degree. Therefore, scale-free networks have a low diameter, which in general ranges from $loglogN$ to $logN$, being N the number of nodes [5]. This means that a message requires very few hops to travel from a node to any other node. The presence of hubs makes the workload not balanced, because hubs must maintain a high number of active connections and they will likely substain a higher workload than the other low-degree nodes [5]. Moreover, scale-free networks are known to be resilient against random failures.

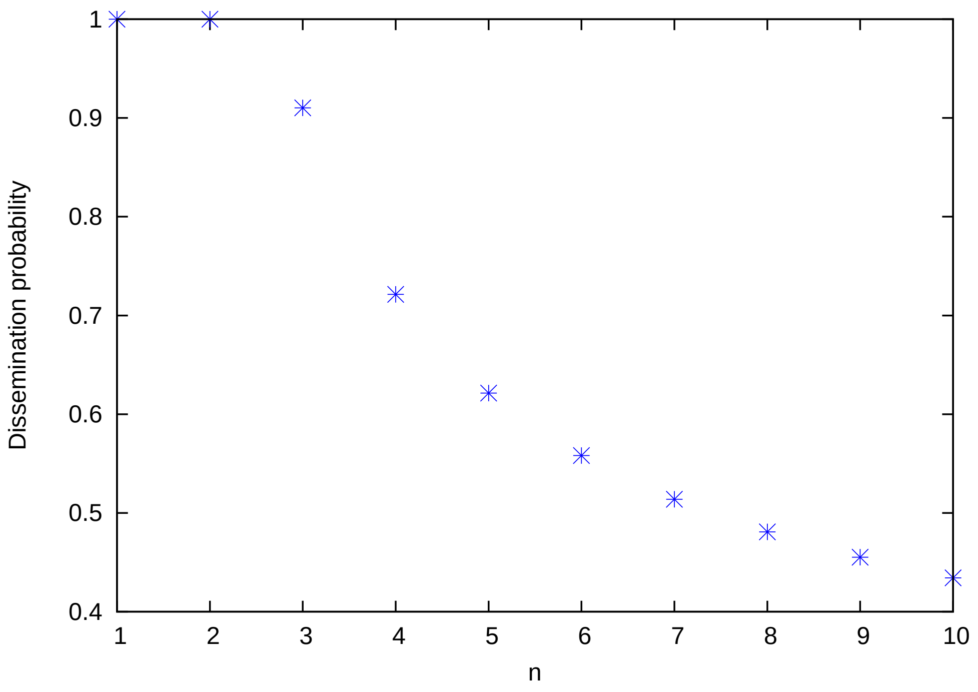

Figure 3.6: Polynomial probability function with $\alpha = 1$

The choice of an initial graph of 3, 5 and 7 nodes has been made to test the behavior of the algorithms when increasing the number of potential hubs. Simulation results will be shown later.

3.2.2 Dissemination probability

The peculiarity of DDG is that gossip's probabilistic component is mixed with a deterministic one. More specifically, dissemination probability is computed considering the degree of each node's neighbor, as described in section 2.6. After the reception of a message, node p checks the dimension of each neighbor's q neighborhood and computes the dissemination probability, according to a specific function. For experimental evaluation, the two following functions have been used:

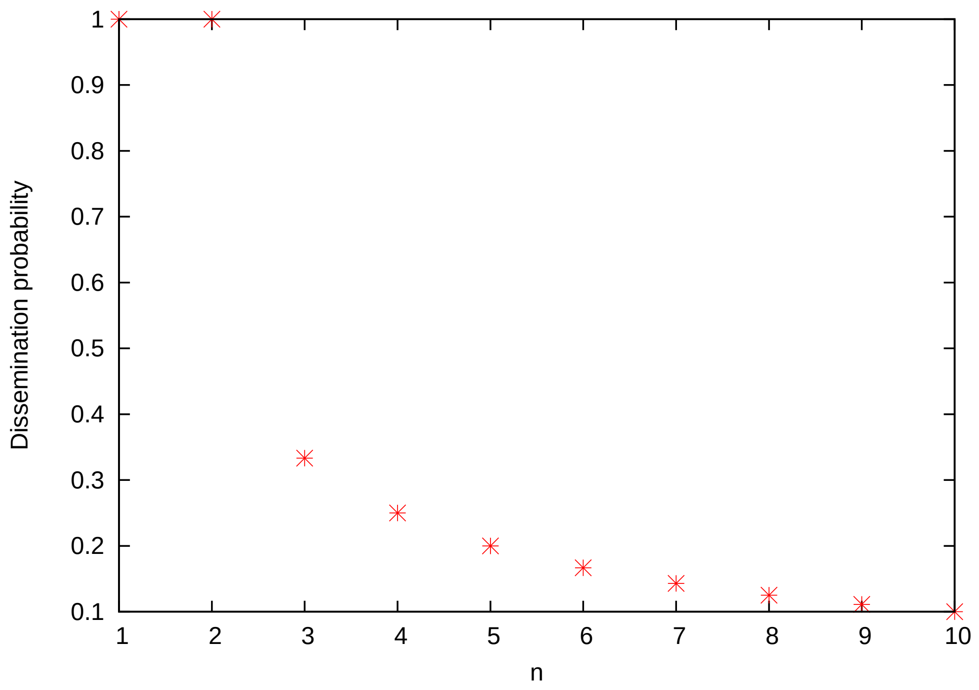

Figure 3.7: Logarithmic probability function with $\alpha = 1$

$$P(n) = \begin{cases} 1, & n = 1,2 \\ \frac{1}{n^{\alpha}}, & n > 2 \end{cases} \qquad P(n) = \begin{cases} 1, & n = 1,2 \\ \frac{1}{ln(\alpha \cdot n)}, & n > 2 \end{cases}$$

being n the number of neighbors of q, and α a user parameter. Figures 3.6 and 3.7 show an example of both functions with $\alpha = 1$ for some values of n.

3.2.3 Model parameters

Apart from dissemination probability, which is a DDG-specific parameter, LUNES has been configured as follows: each simulation run is 5000 time steps long and each node in the network can generate new messages during the whole simulation lifespan; the time between successive messages is generated according to a typical exponential distribution.

Nodes' cache has a size of 256 items, and it's managed using the Least Recently Used (LRU) replacement algorithm. Moreover, each message has a

Time-To-Live (TTL) property, in order to limit its lifetime in the network. As usual, each hop reduces this value up to discarding. For these simulations, the TTL has been set to 8, a value that is always greater than or equal to the diameter of both random and scale-free graphs used for the simulations.

Algorithms have been tested on 10 runs (each of them on a different graph) for each of the 100 chosen values of α on the whole corpus. The result of each run block is computed as the average of the 10 runs.

3.3 Metrics

As described in section 1.3.3, there are many possible metrics to evaluate network graphs algorithms. Among those provided by LUNES, the following ones have been chosen:

- *Coverage*: the fraction of nodes which actually received the messages (higher is better).
- *Delay*: the average number of hops that a message traverses before reaching a node (lower is better).
- *Overhead ratio*: accurately described in [5], overhead ratio ρ is defined as follows:
$$\rho = \frac{Delivered\ messages}{Lower\ bound}$$
where 'delivered messages' is the total number of messages that are delivered in a simulation run by a specific dissemination protocol, and the 'lower bound' is the minimum number of messages (in each graph) that are necessary to obtain a complete coverage [5].

3.4 Simulation results

Simulations aim at comparing the two configurations of DDG presented in 3.2.2 with Fixed Probability Gossip and Probabilistic Broadcast on random and scale-free graphs. Experimental evaluation is focused on showing the cost of the three algorithms, represented by overhead ratio on x-axis, with respect to their effectiveness, i.e. coverage and delay on y-axis.

3.4.1 Random graphs

Even though degree distribution is almost uniform in random graphs, both polynomial (figures 3.8 and 3.9) and logarithmic (figures 3.10 and 3.11) functions perform better than the other algorithms in terms of **coverage** (in some cases more than 10%, overhead being equal). However, they both show in some cases a bit higher **delay** (less than one hop on average). This is caused by DDG forwarding policy, which reduces the amount of redundant messages by lowering reception probability of high connected nodes. As a result, DDG shows a higher coverage and a lower average value of total sent messages in each dissemination, but each message is forced to follow a little longer path to reach all nodes.

3.4.2 Scale-free graphs

The most interesting experimental results are provided by tests run on scale-free networks. First of all, there is a loose dependency between coverage and initial graph dimension for all the algorithms: in fact, figures 3.32, 3.34, 3.36, and 3.38 show that the overhead-coverage results are almost the same for initial graphs of 3, 5, and 7 nodes. Fixed Probability and Probabilistic Broadcast are able to reach a complete **coverage** with $\rho \sim 3$, while all DDG configurations reach the same point with $\rho \sim 2$, i.e. they show a better exploitation of network topology and fewer messages are forwarded (figures 3.14, 3.16, 3.20, 3.22, 3.26, and 3.28). This is not an unexpected result: in fact, the key characteristics of scale-free networks are their particular degree distribution and the existence of *hubs*. DDG is able to exploit these peculiarities due to its deterministic component: nodes with a very low degree (the largest part in scale-free networks) are always eligible to receive new messages, while hubs' degree limit forwarding redundancy.

With regard to **delay**, DDG performs in almost all configurations better than the other algorithms (figures 3.15, 3.17, 3.21, 3.23, 3.27, and 3.29). Even in this case, the reason can be found in its deterministic component: scale-free networks have a low diameter, thus messages should spread all over the network in a few hops. However, Fixed Probability and Probabilistic Broadcast don't consider the distance of nodes from the center of the network, because probabilistic dissemination is applied to all nodes the same way. Therefore, messages may 'bounce' between lots of nodes (increasing overhead ratio) before reaching poorly connected ones. The dynamics of dissemination probability applied by DDG overcomes this issue, because hubs act as 'forwarding amplifiers' of nodes with low connectivity.

Opposed to the behavior towards coverage, the dimension of initial graph has a strong influence on delay of all algorithms (figures 3.33, 3.35, 3.37, and 3.39): the bigger is the size of initial graph, the higher is the average delay. A possible explanation for this phenomenon is that a higher number of potential hubs makes nodes with low degree express different *preferential attachment*, thus network topology doesn't appear as a proper *star*. In other words, during the *growing phase*, each new node has a wider choice of hubs to attach to. As a result, network topology is less centralized and messages have to traverse a bit more hops to reach all nodes.

3.4.3 DDG dissemination probability

The deterministic component of DDG strongly influences algorithm performances, but the two tested functions seem to behave similarly (figures 3.12 and 3.13 for random graphs, and 3.18, 3.19, 3.24, 3.25, 3.30 and 3.31 for scale-free graphs), even though probability distributions' slopes are quite different. The explanation for the similar behavior of both polynomial and logarithmic function can be found in the very high amount of messages produced by the nodes, together with the considerable length of simulation (5000 time steps) and the small size of graphs. The slope of probability distributions is the cause of the high density of values both in random and scale-free graphs exhibited by logarithmic function when varying the value of α: the influence of α as a multiplication coefficient is indeed lower than as an exponent.

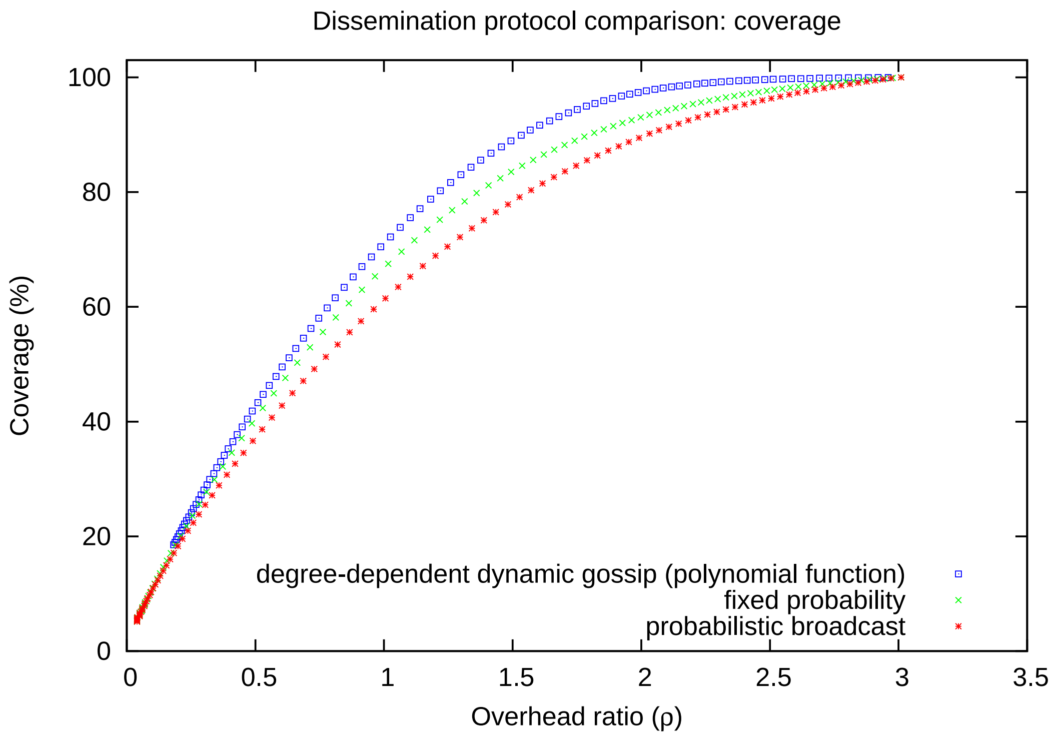

Figure 3.8: Fixed Probability, Probabilistic Broadcast, and DDG with polynomial function: *coverage* on random graphs

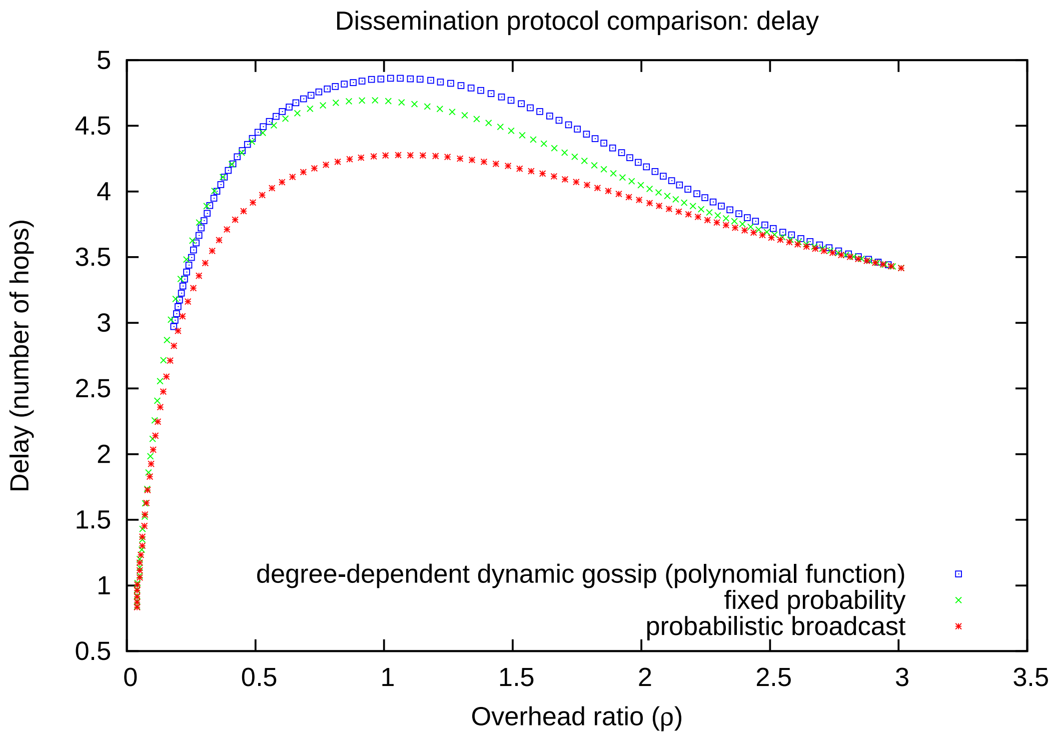

Figure 3.9: Fixed Probability, Probabilistic Broadcast, and DDG with polynomial function: *delay* on random graphs

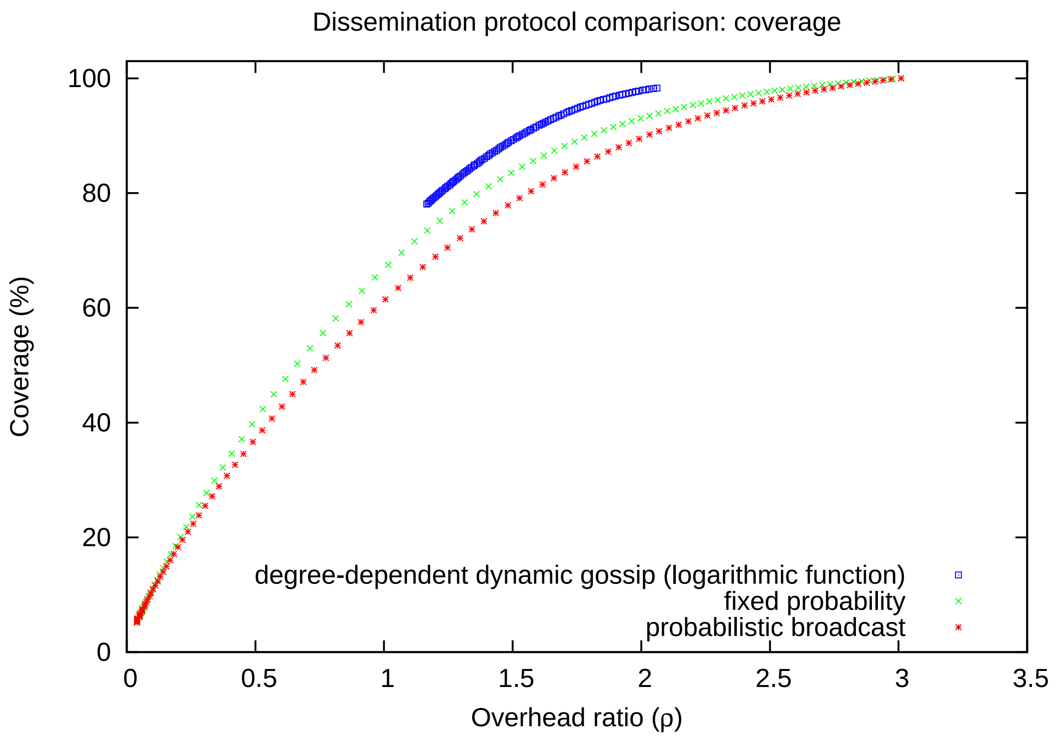

Figure 3.10: Fixed Probability, Probabilistic Broadcast, and DDG with logarithmic function: *coverage* on random graphs

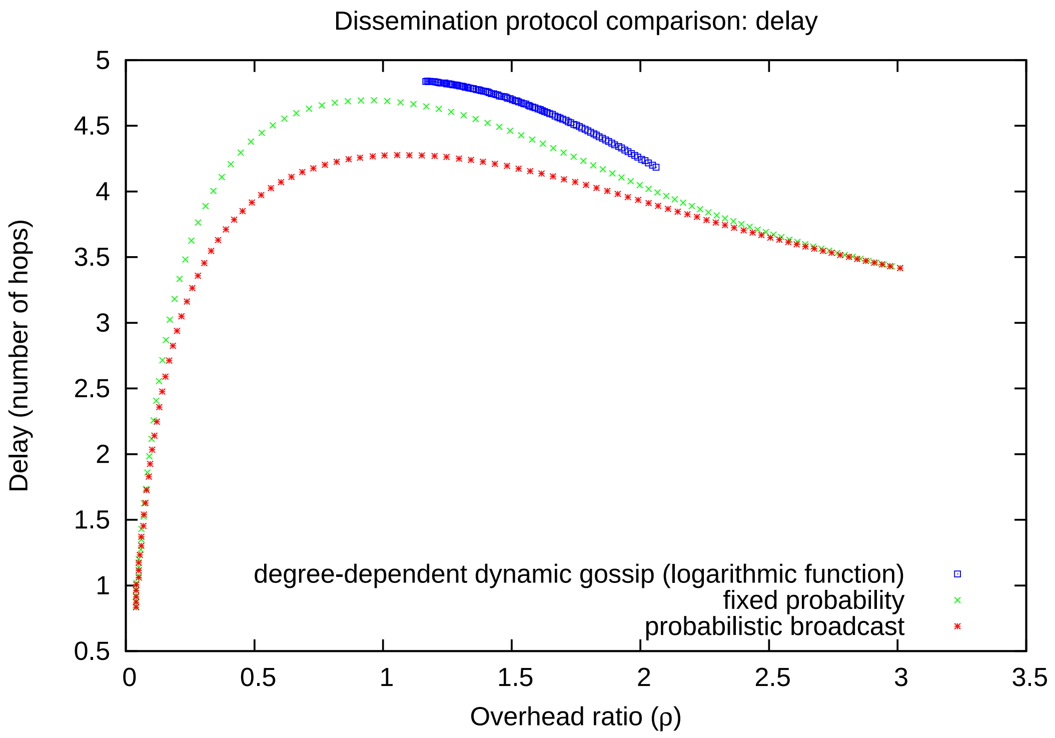

Figure 3.11: Fixed Probability, Probabilistic Broadcast, and DDG with logarithmic function: *delay* on random graphs

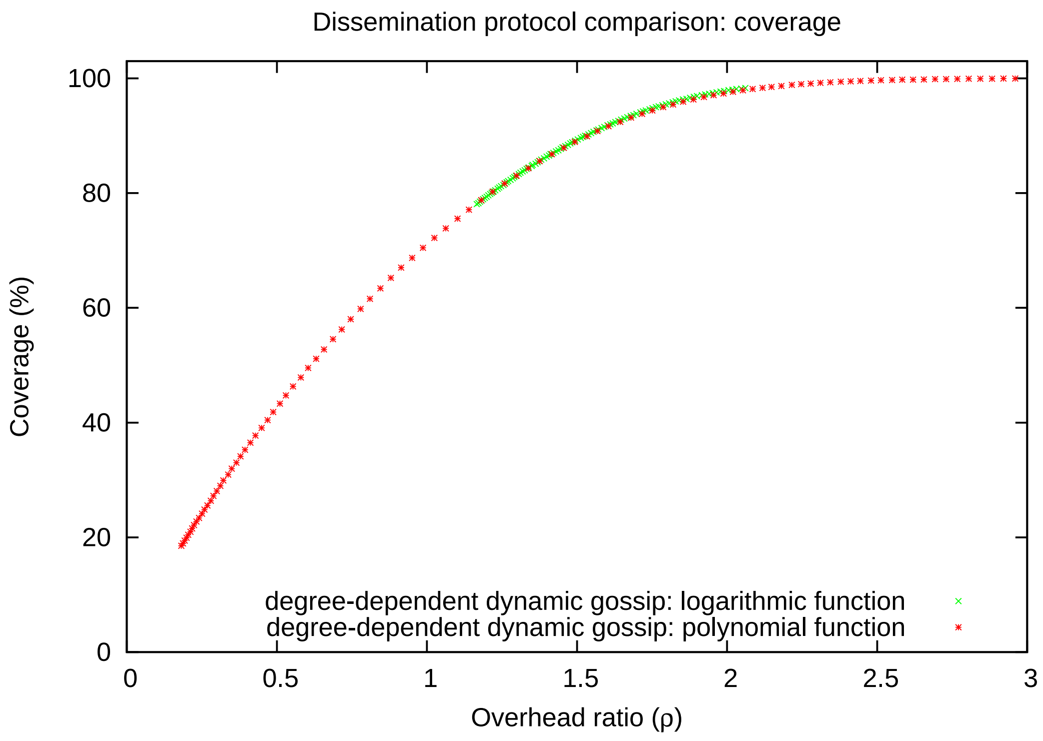

Figure 3.12: DDG with polynomial and logarithmic function: *coverage* on random graphs

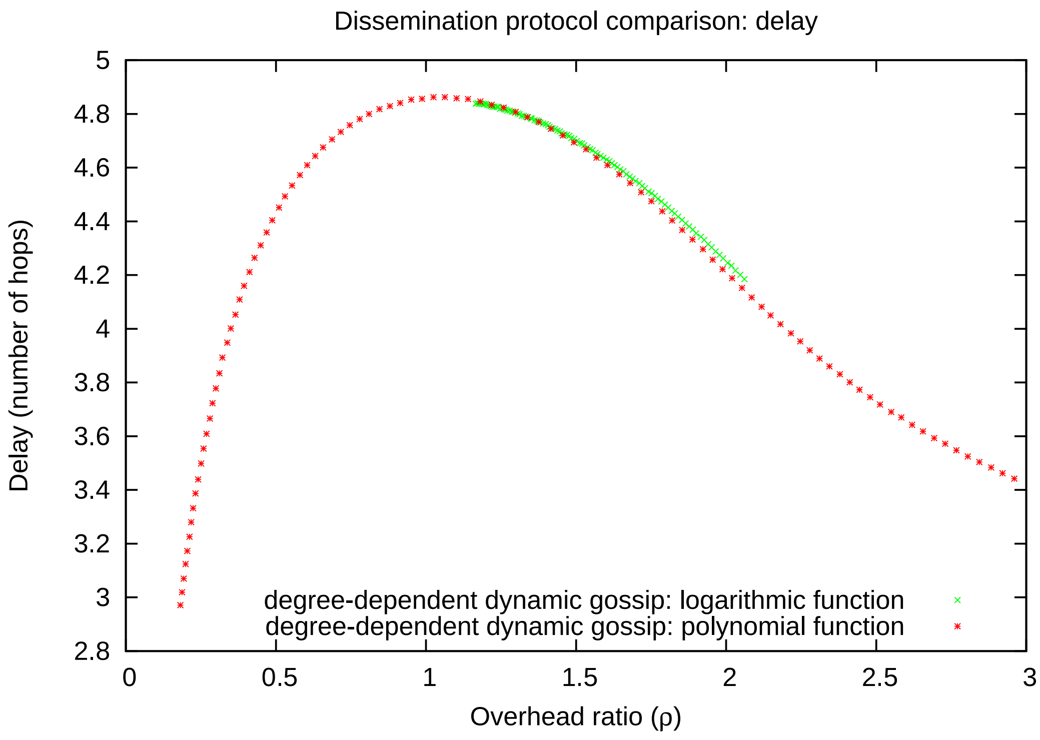

Figure 3.13: DDG with polynomial and logarithmic function: *delay* on random graphs

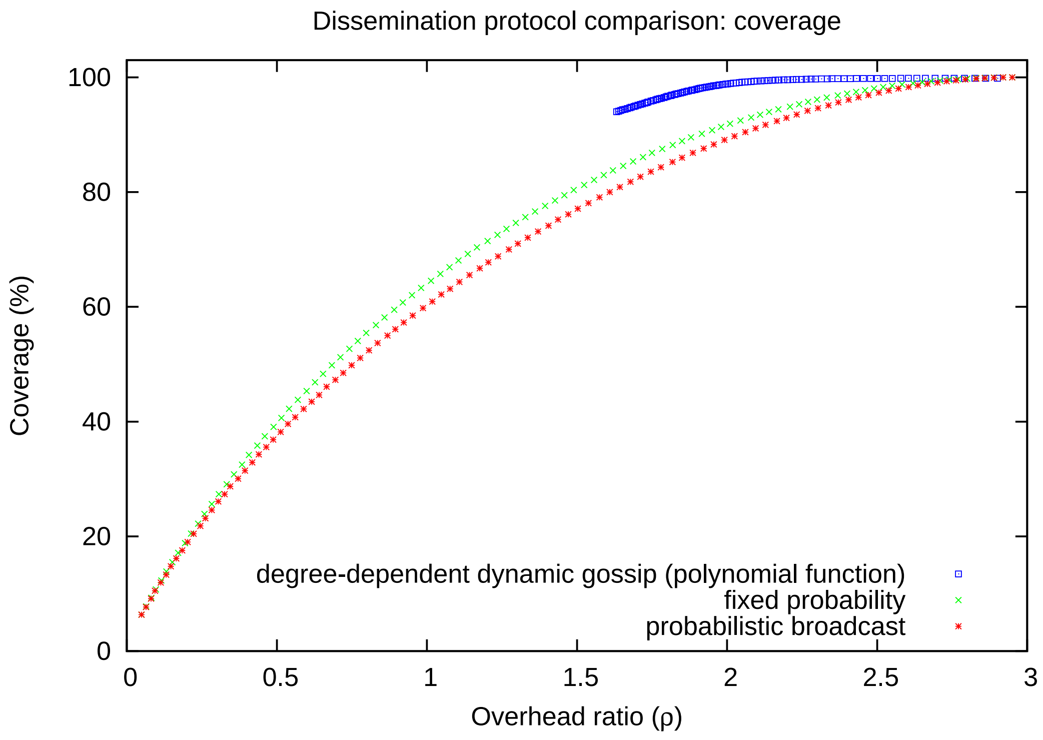

Figure 3.14: Fixed Probability, Probabilistic Broadcast, and DDG with polynomial function: *coverage* on scale-free graphs with 3 initial nodes

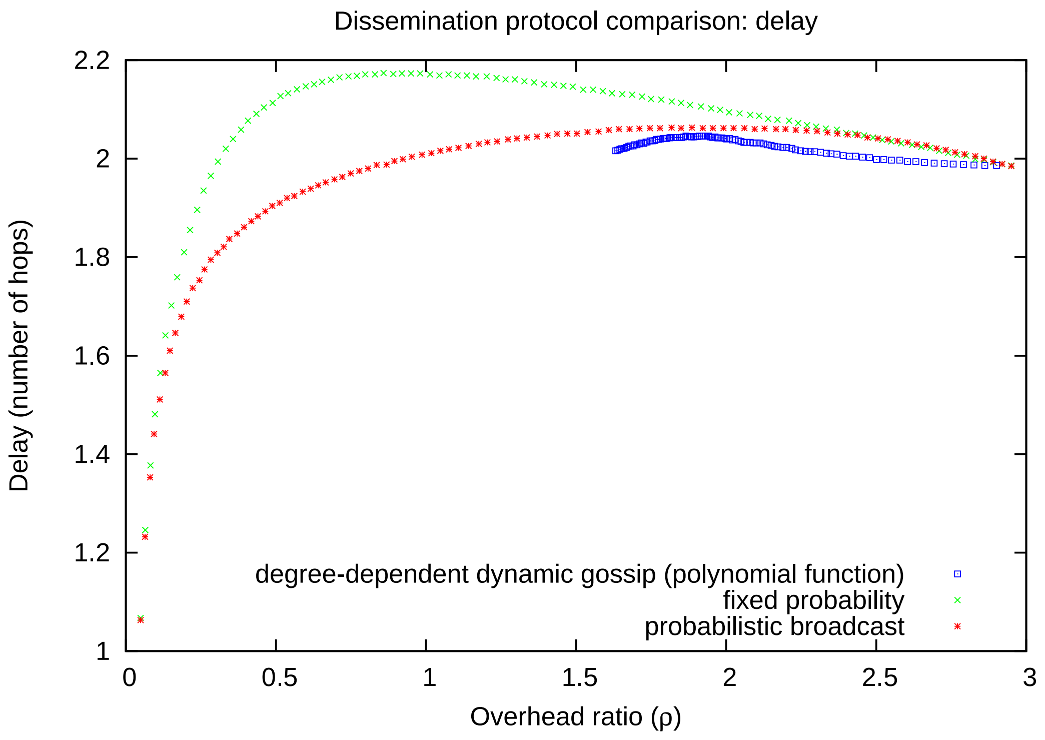

Figure 3.15: Fixed Probability, Probabilistic Broadcast, and DDG with polynomial function: *delay* on scale-free graphs with 3 initial nodes

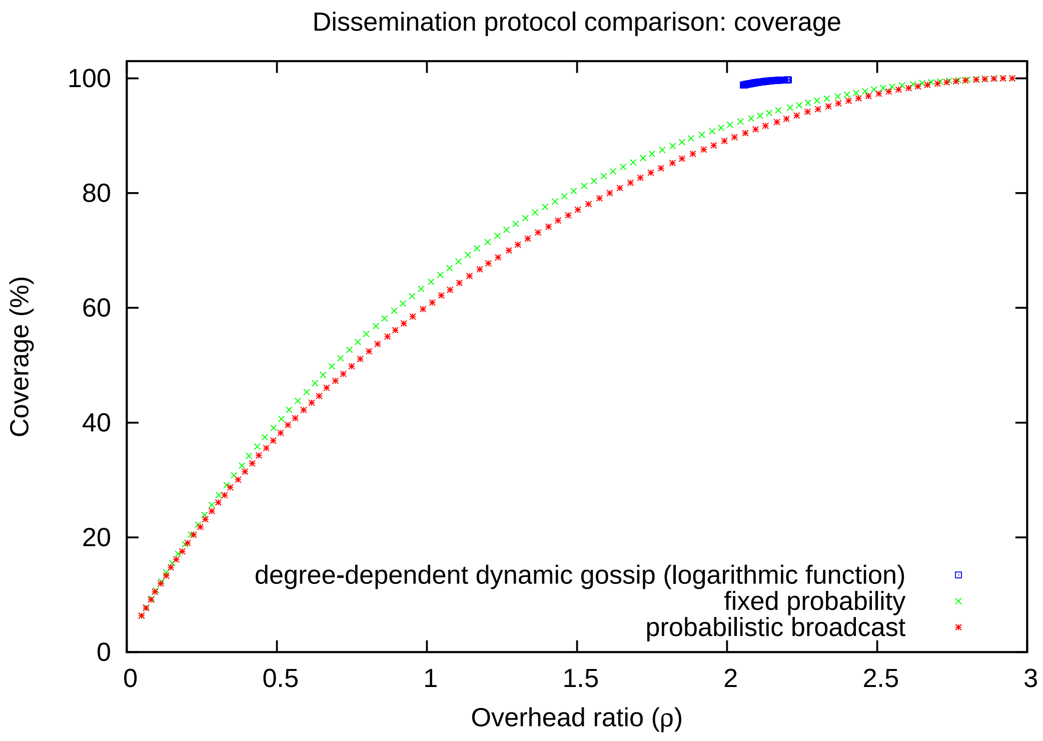

Figure 3.16: Fixed Probability, Probabilistic Broadcast, and DDG with logarithmic function: *coverage* on scale-free graphs with 3 initial nodes

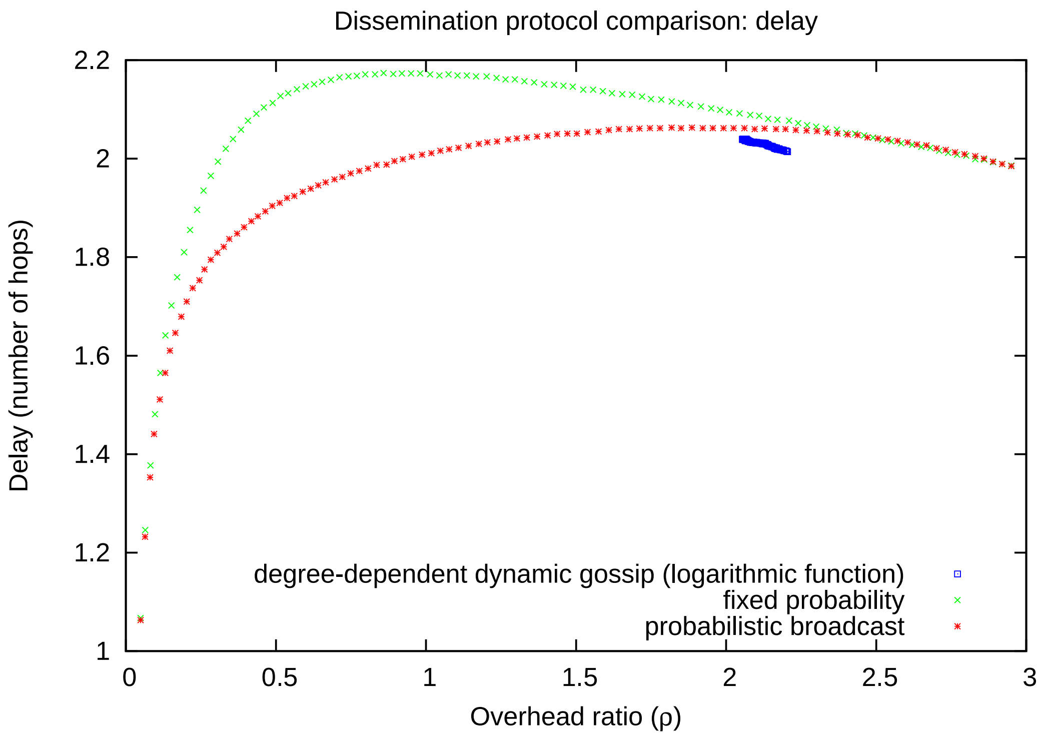

Figure 3.17: Fixed Probability, Probabilistic Broadcast, and DDG with logarithmic function: *delay* on scale-free graphs with 3 initial nodes

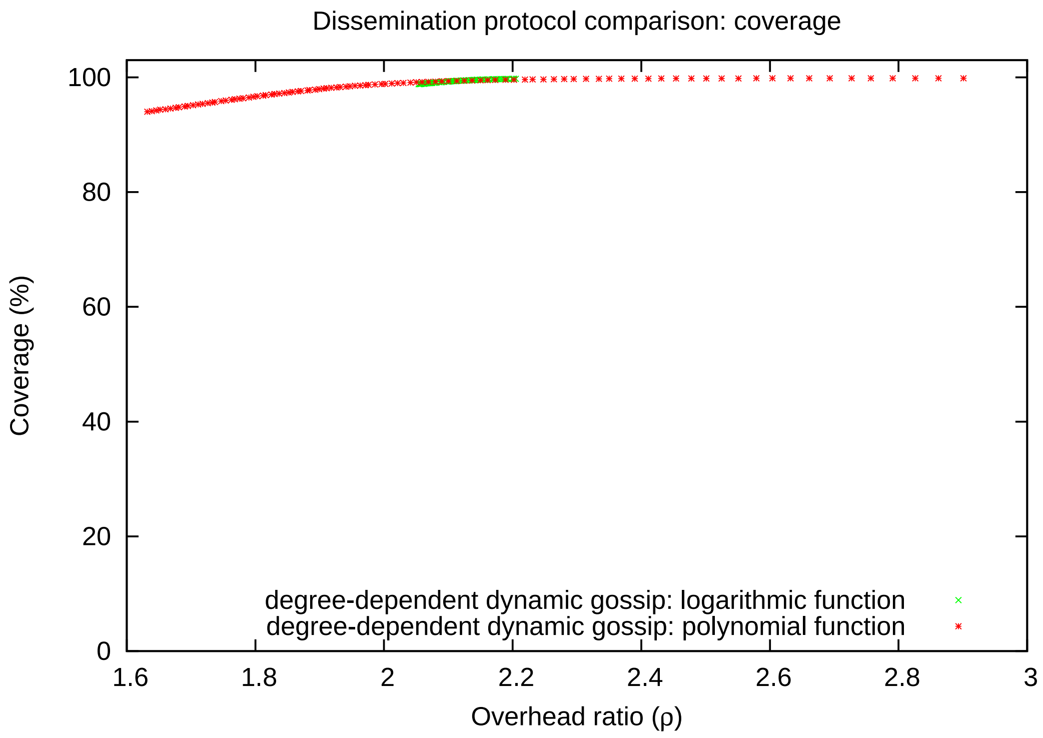

Figure 3.18: DDG with polynomial and logarithmic function: *coverage* on scale-free graphs with 3 initial nodes

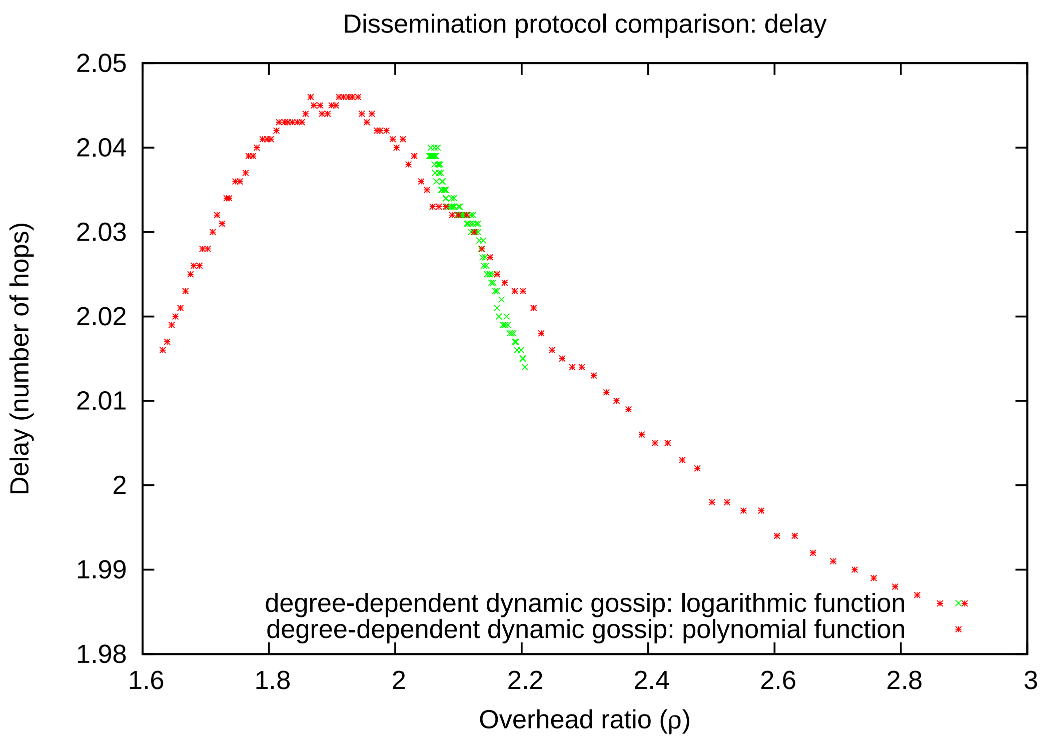

Figure 3.19: DDG with polynomial and logarithmic function: *delay* on scale-free graphs with 3 initial nodes

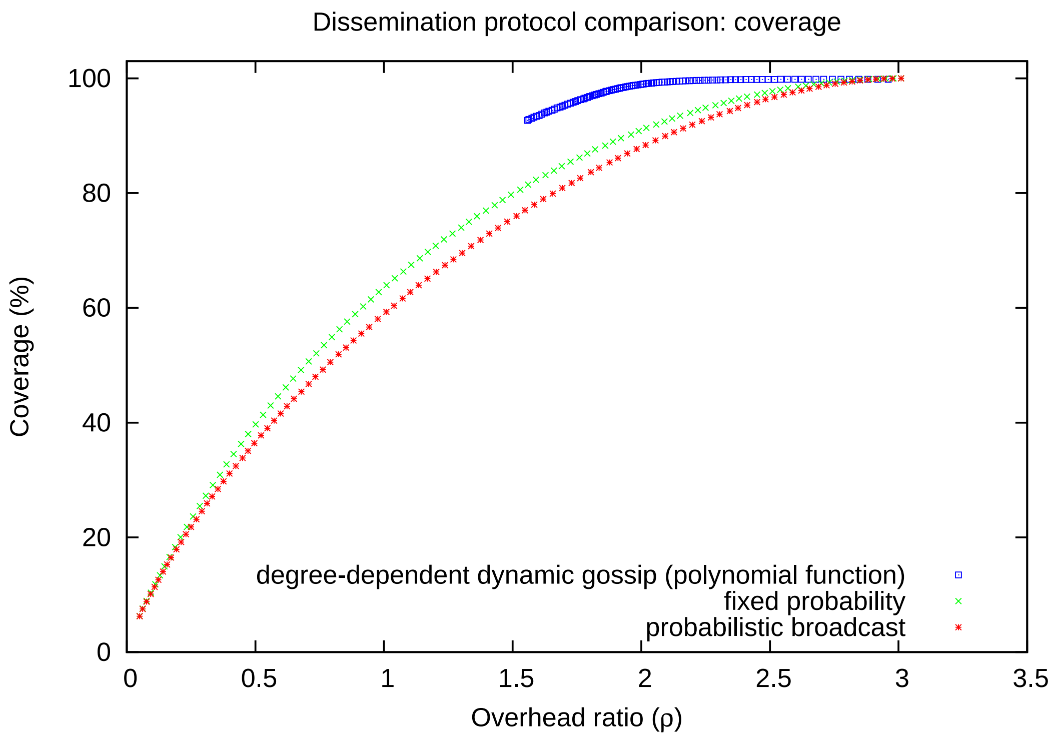

Figure 3.20: Fixed Probability, Probabilistic Broadcast, and DDG with polynomial function: *coverage* on scale-free graphs with 5 initial nodes

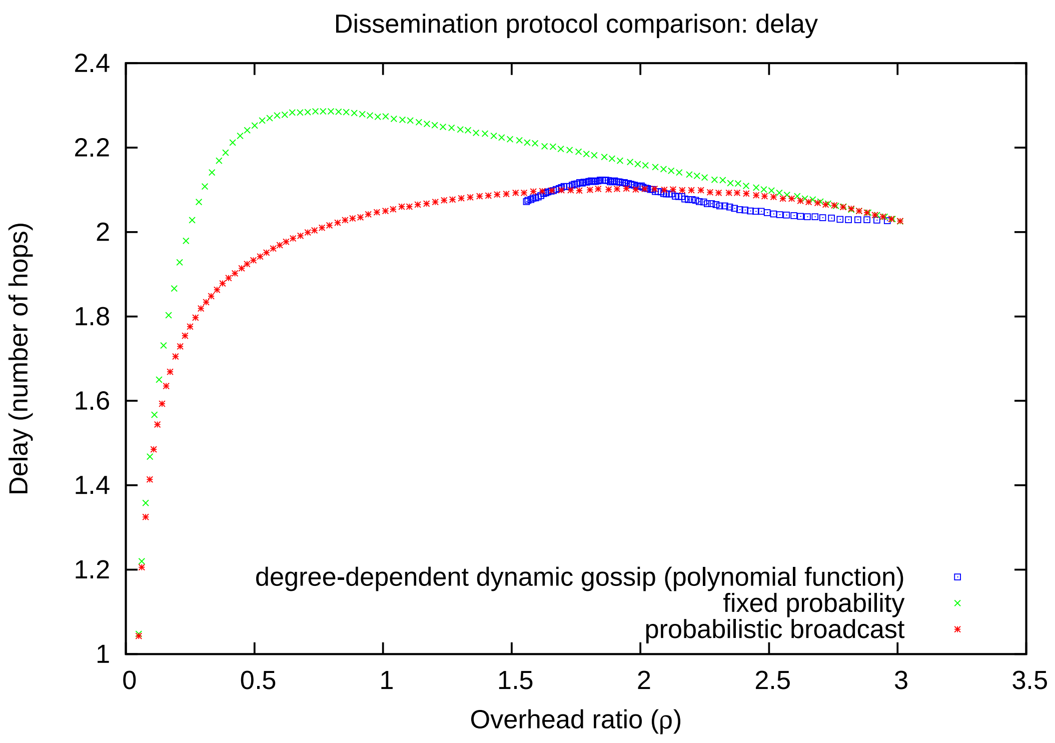

Figure 3.21: Fixed Probability, Probabilistic Broadcast, and DDG with polynomial function: *delay* on scale-free graphs with 5 initial nodes

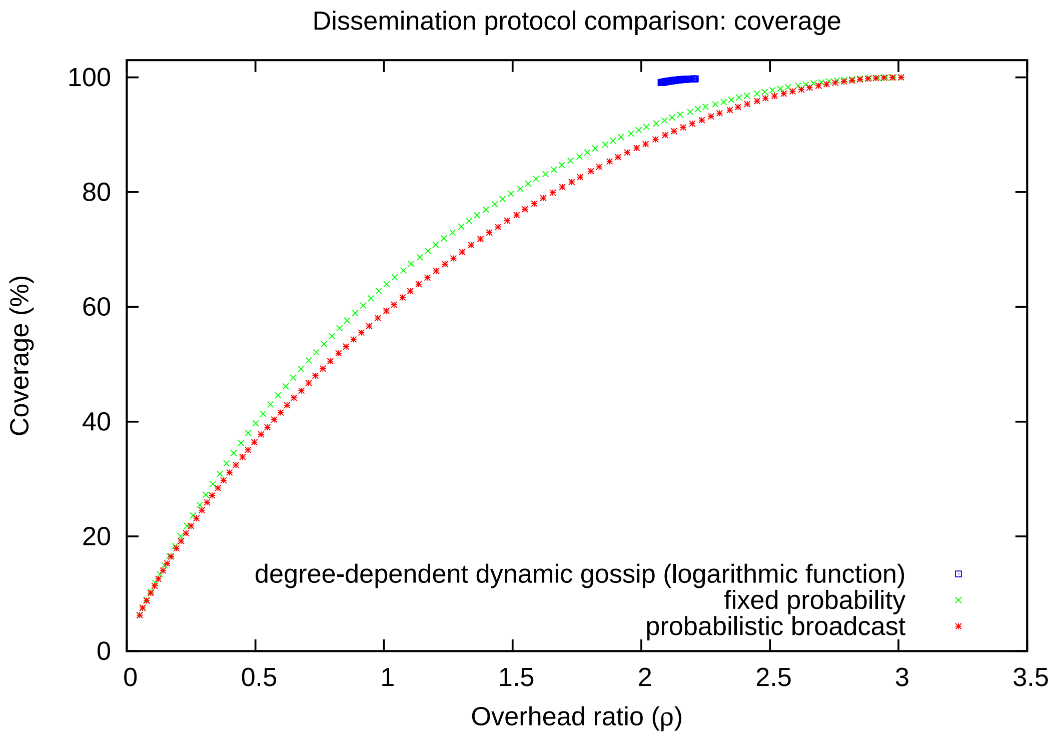

Figure 3.22: Fixed Probability, Probabilistic Broadcast, and DDG with logarithmic function: *coverage* on scale-free graphs with 5 initial nodes

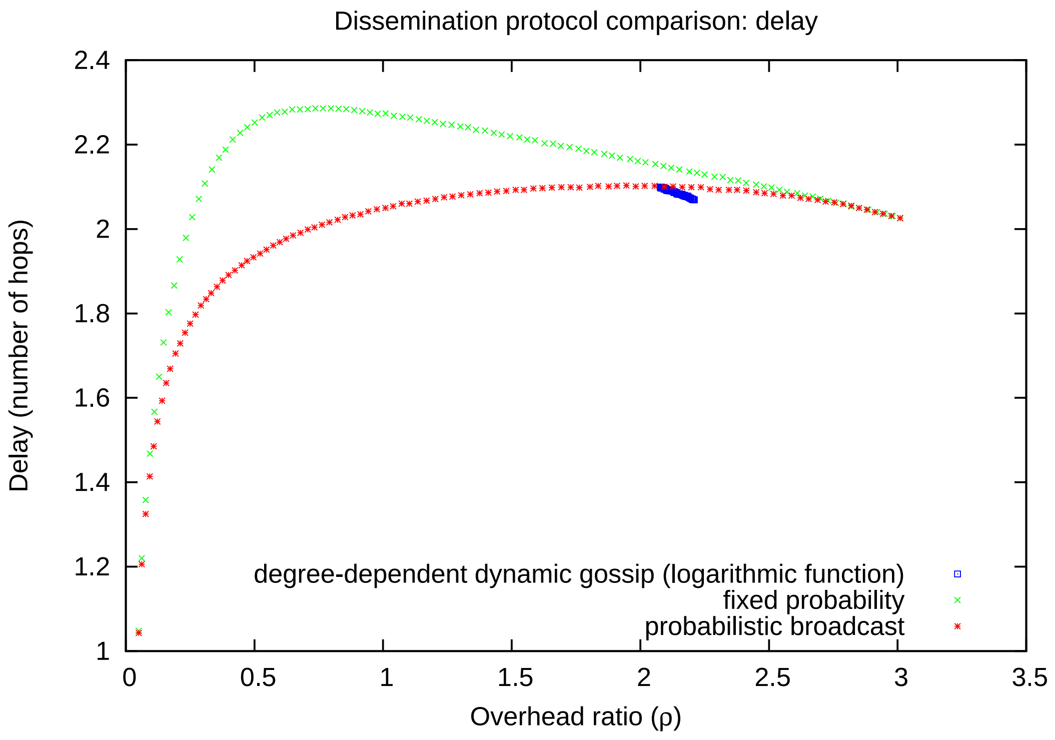

Figure 3.23: Fixed Probability, Probabilistic Broadcast, and DDG with logarithmic function: *delay* on scale-free graphs with 5 initial nodes

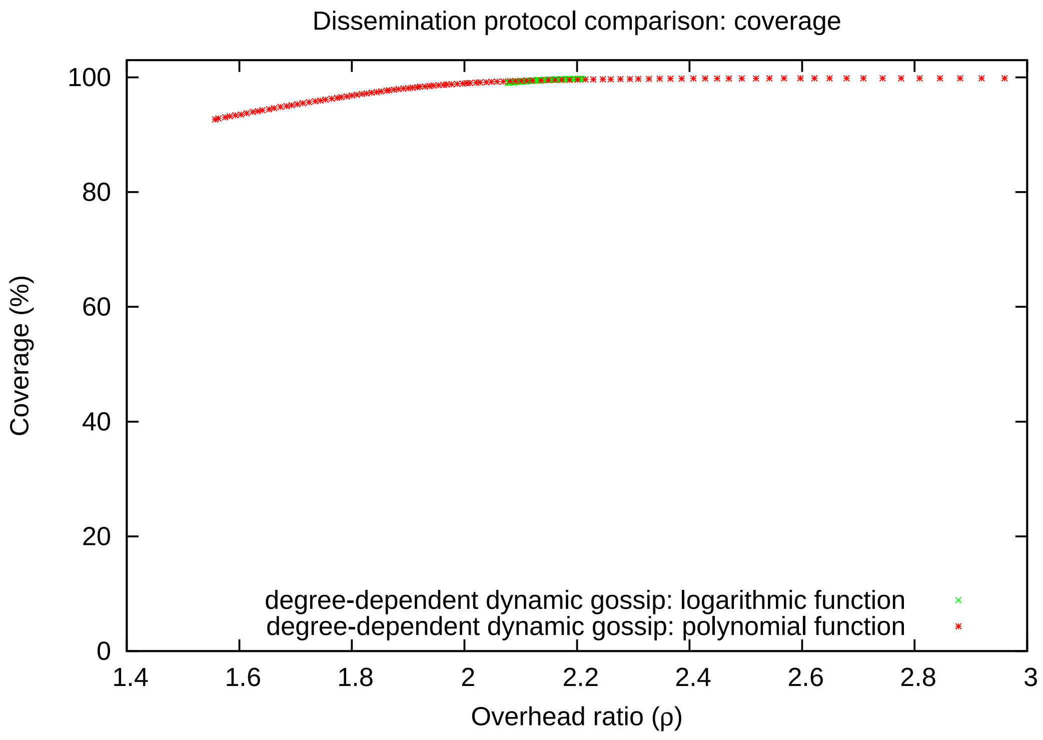

Figure 3.24: DDG with polynomial and logarithmic function: *coverage* on scale-free graphs with 5 initial nodes

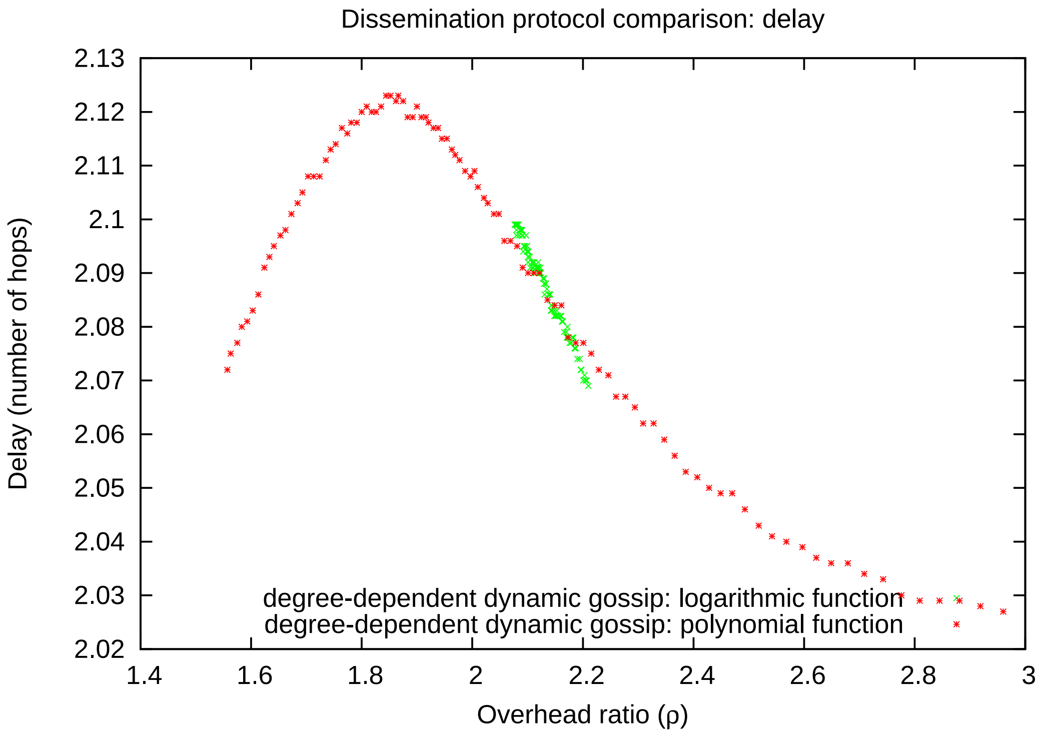

Figure 3.25: DDG with polynomial and logarithmic function: *delay* on scale-free graphs with 5 initial nodes

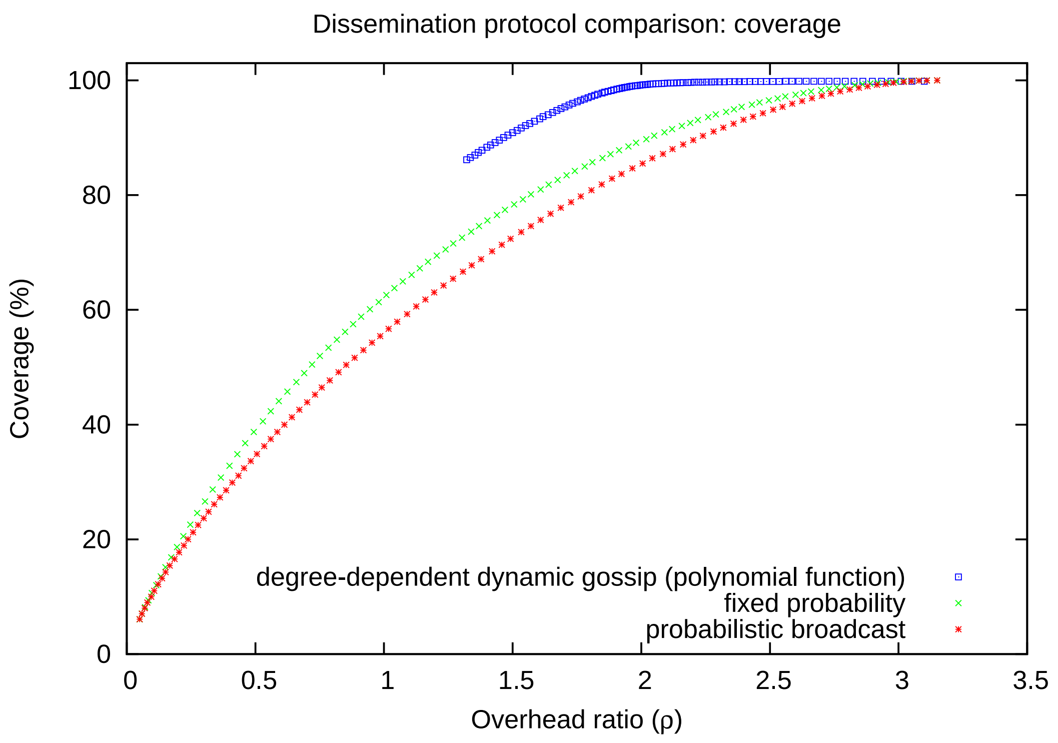

Figure 3.26: Fixed Probability, Probabilistic Broadcast, and DDG with polynomial function: *coverage* on scale-free graphs with 7 initial nodes

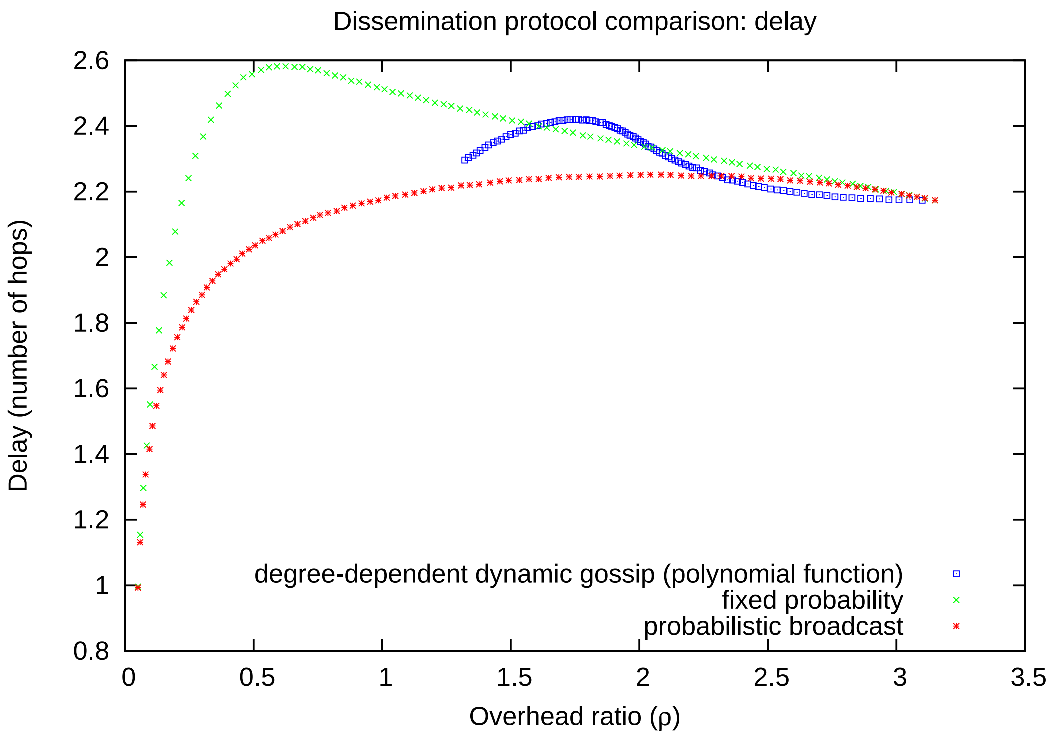

Figure 3.27: Fixed Probability, Probabilistic Broadcast, and DDG with polynomial function: *delay* on scale-free graphs with 7 initial nodes

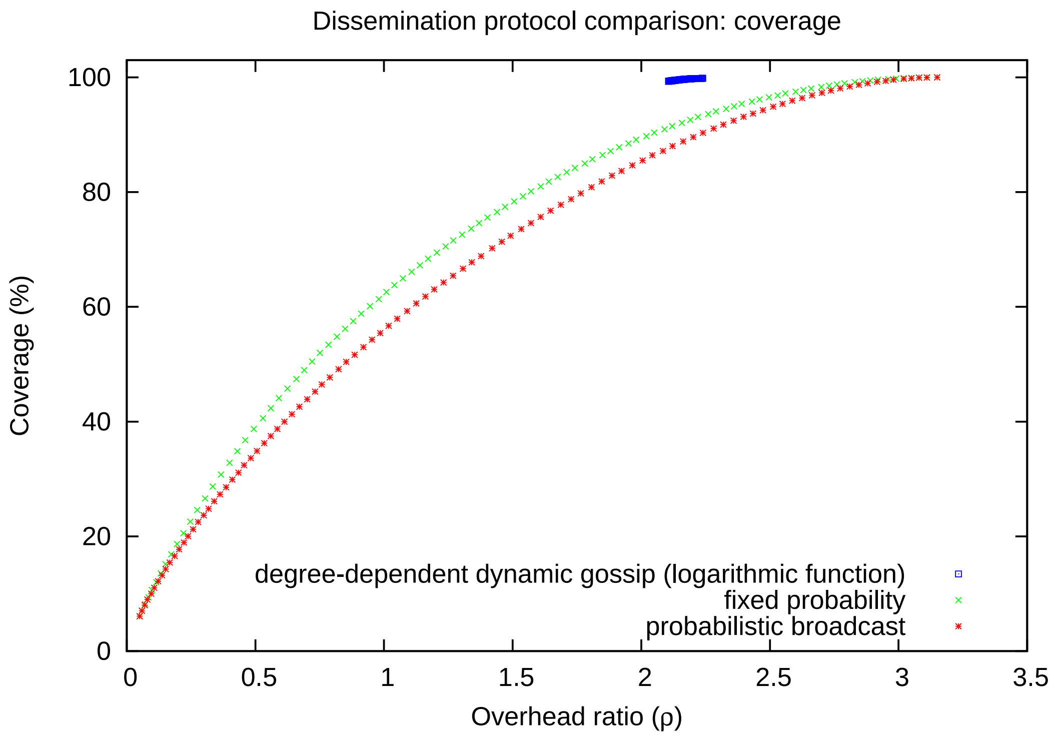

Figure 3.28: Fixed Probability, Probabilistic Broadcast, and DDG with logarithmic function: *coverage* on scale-free graphs with 7 initial nodes

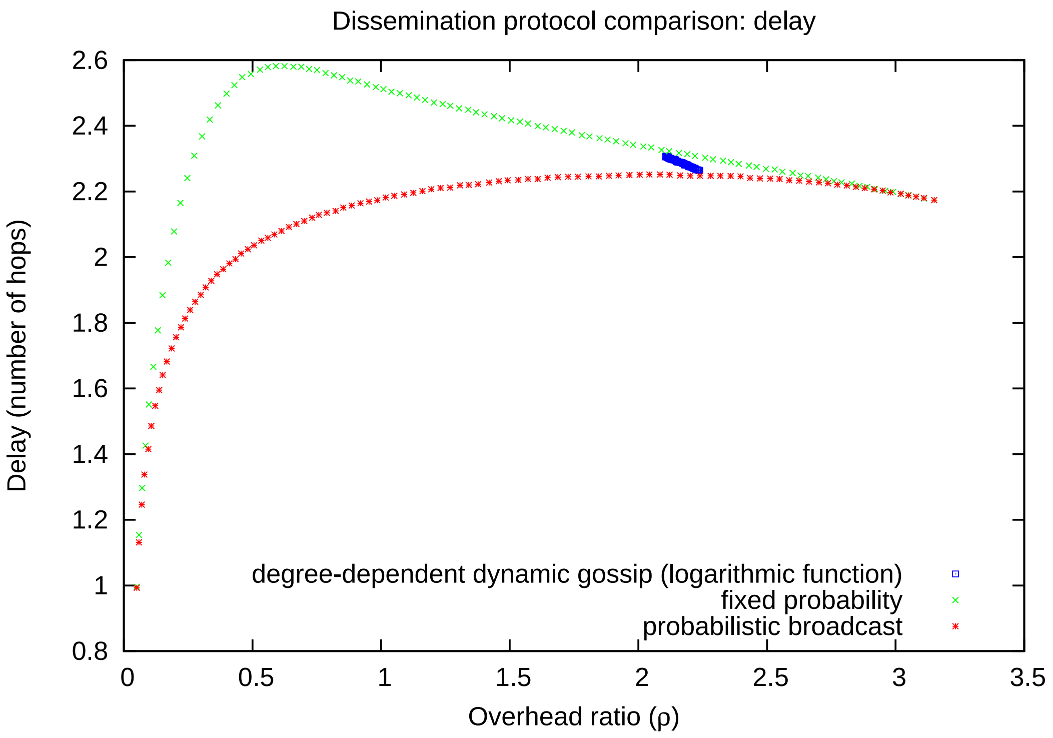

Figure 3.29: Fixed Probability, Probabilistic Broadcast, and DDG with logarithmic function: *delay* on scale-free graphs with 7 initial nodes

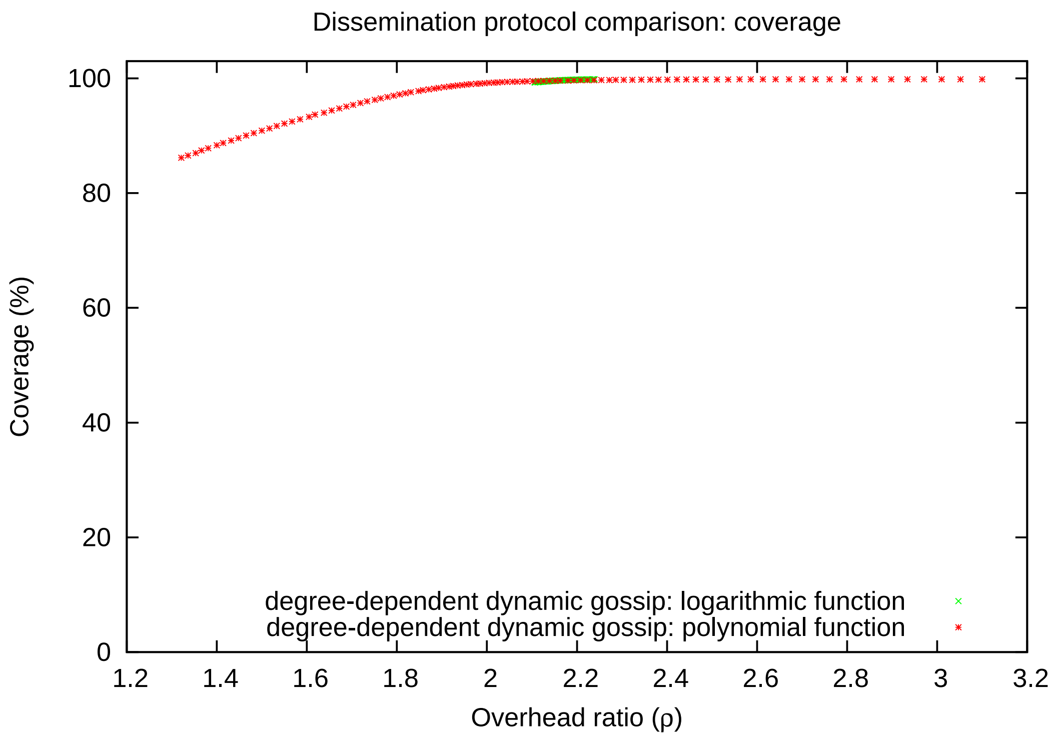

Figure 3.30: DDG with polynomial and logarithmic function: *coverage* on scale-free graphs with 7 initial nodes

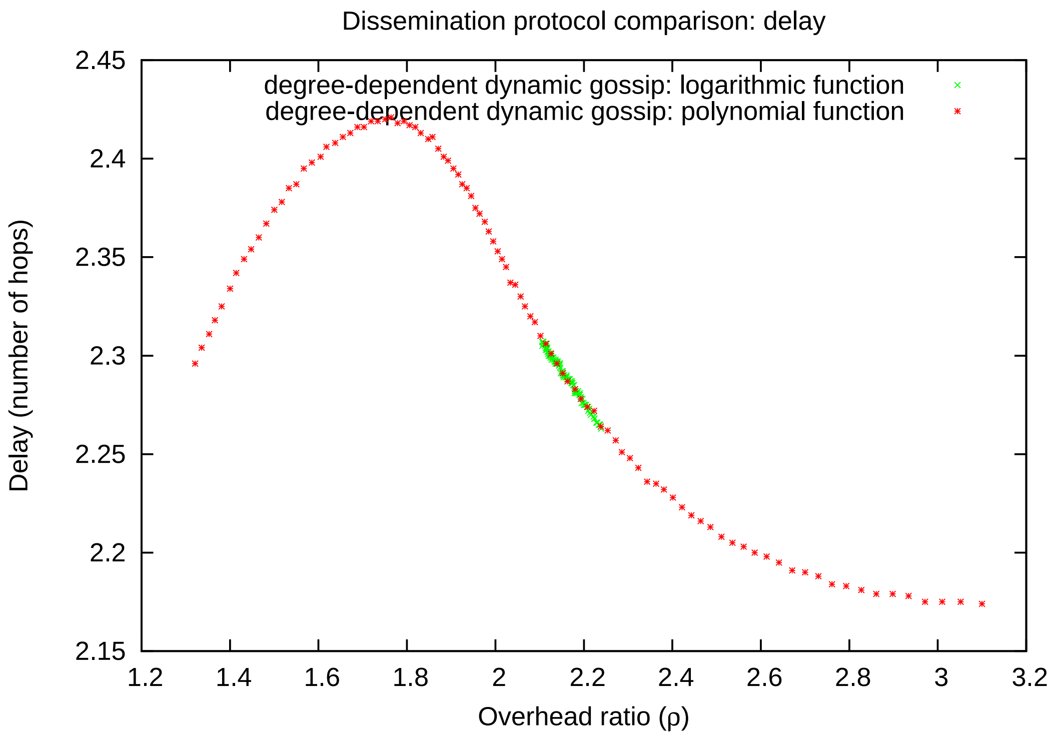

Figure 3.31: DDG with polynomial and logarithmic function: *delay* on scale-free graphs with 7 initial nodes

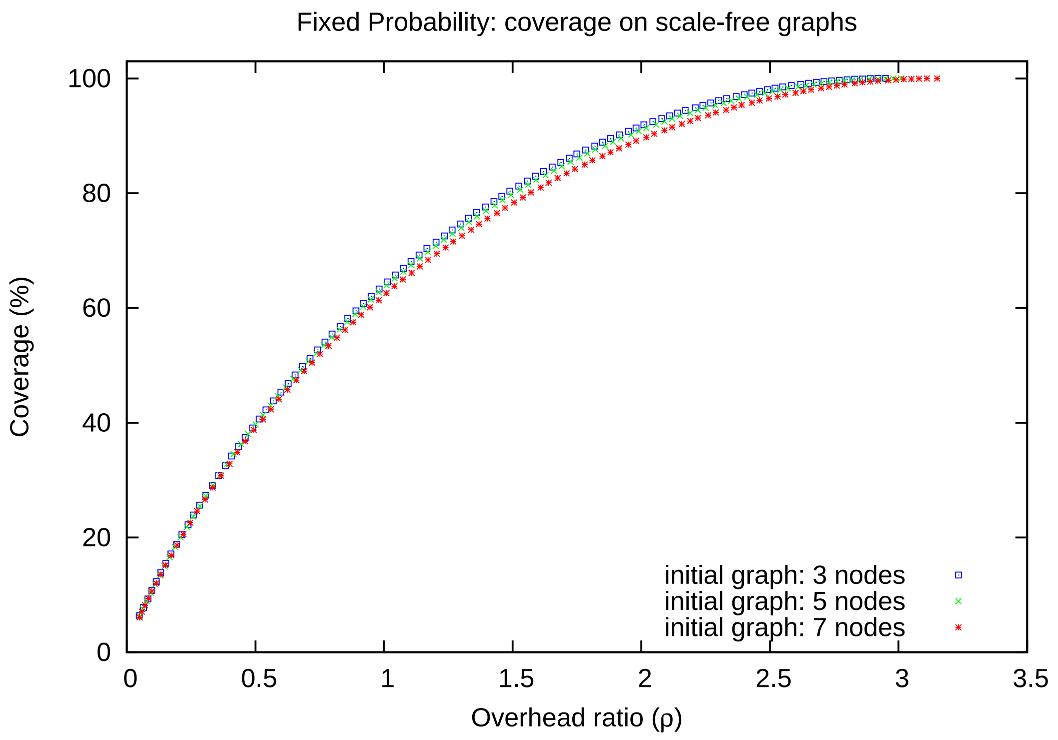

Figure 3.32: Fixed Probability: *coverage* on scale-free graphs with 3, 5, and 7 initial nodes

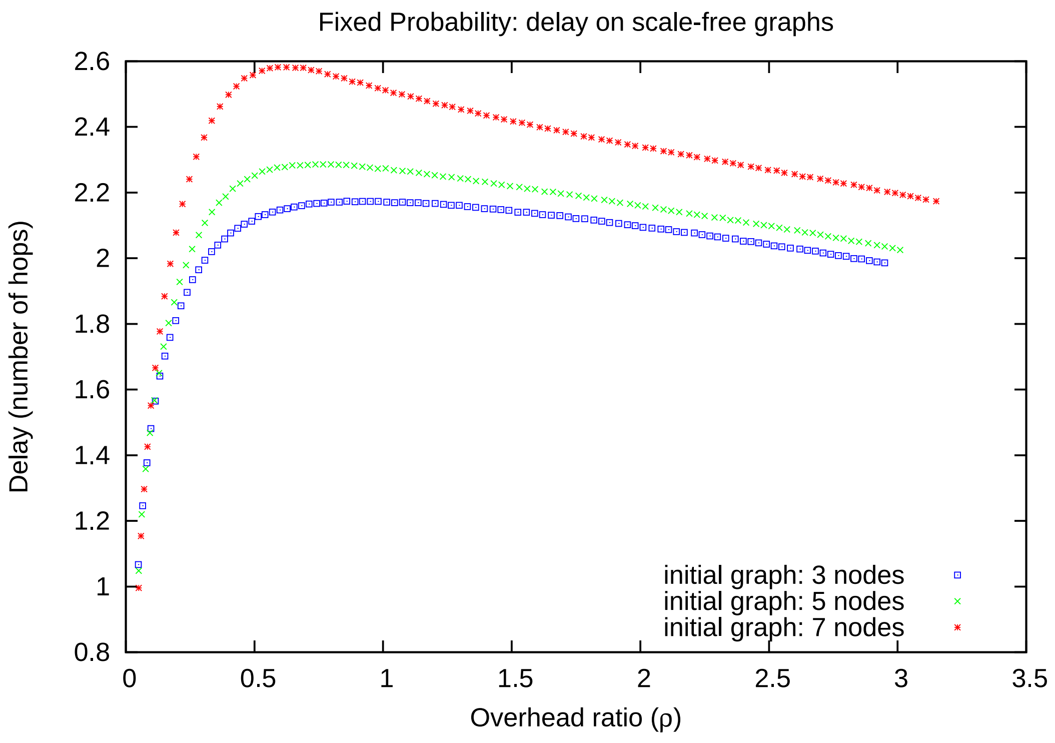

Figure 3.33: Fixed Probability: *delay* on scale-free graphs with 3, 5, and 7 initial nodes

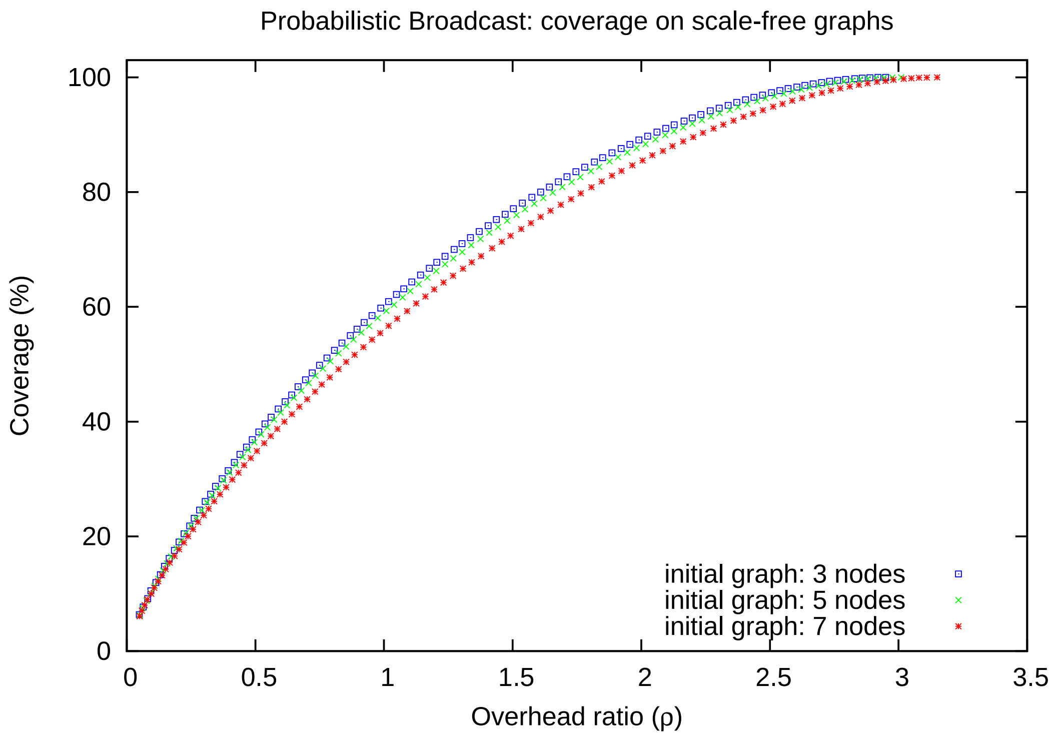

Figure 3.34: Probabilistic Broadcast: *coverage* on scale-free graphs with 3, 5, and 7 initial nodes

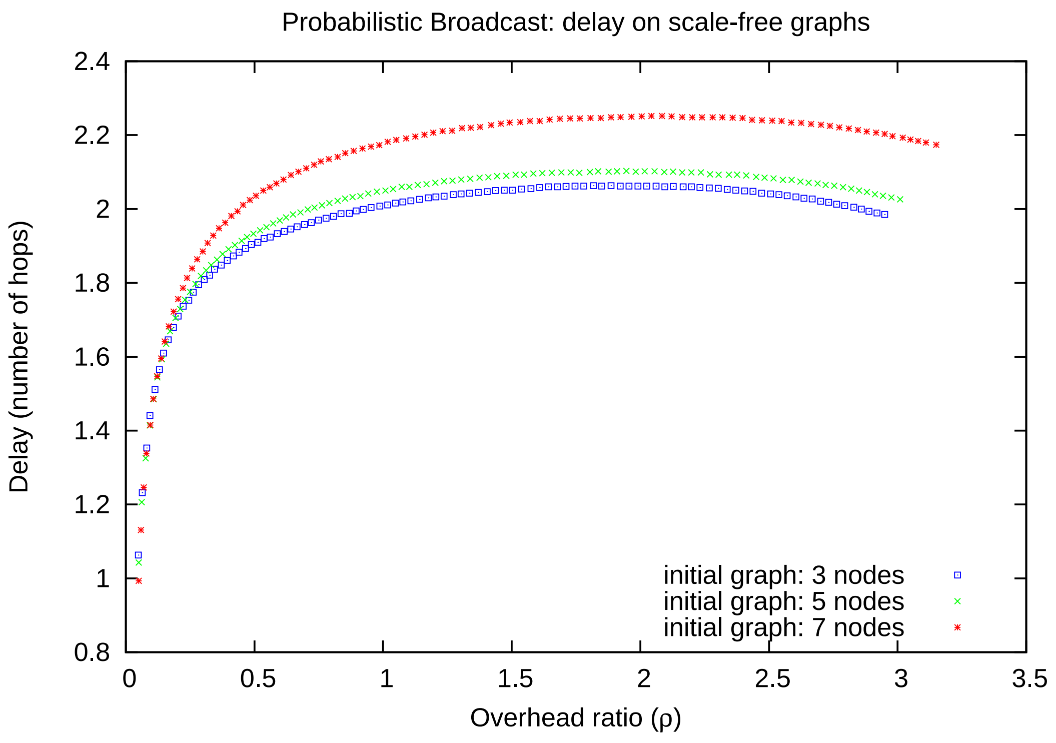

Figure 3.35: Probabilistic Broadcast: *delay* on scale-free graphs with 3, 5, and 7 initial nodes

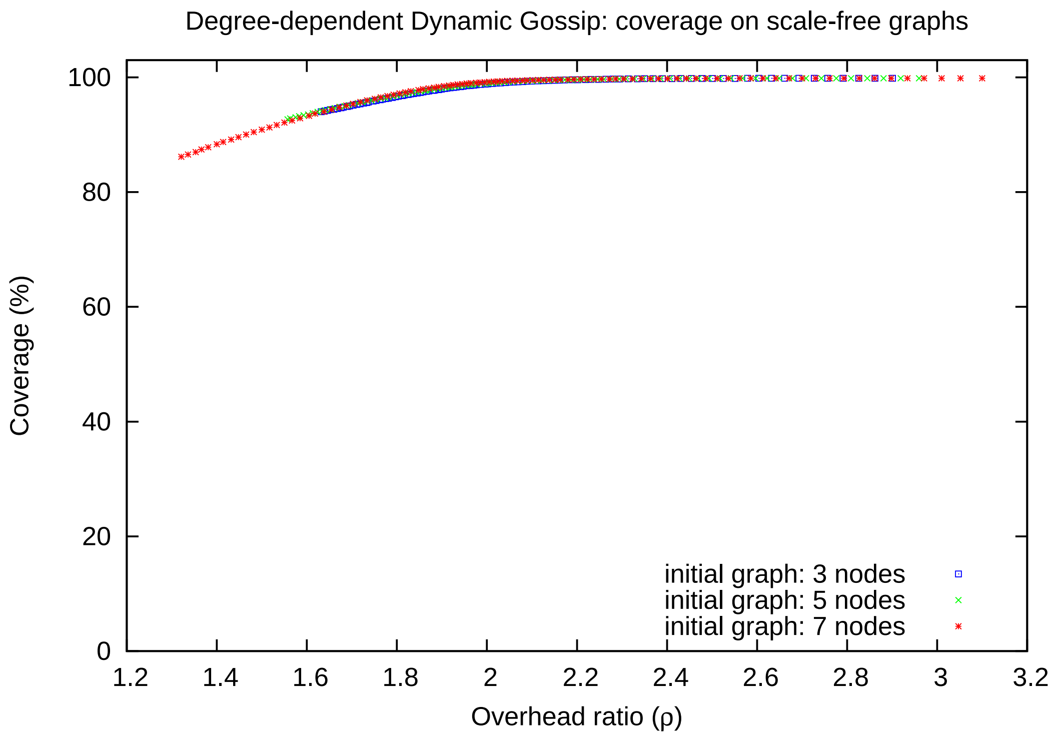

Figure 3.36: DDG with polynomial function: *coverage* on scale-free graphs with 3, 5, and 7 initial nodes

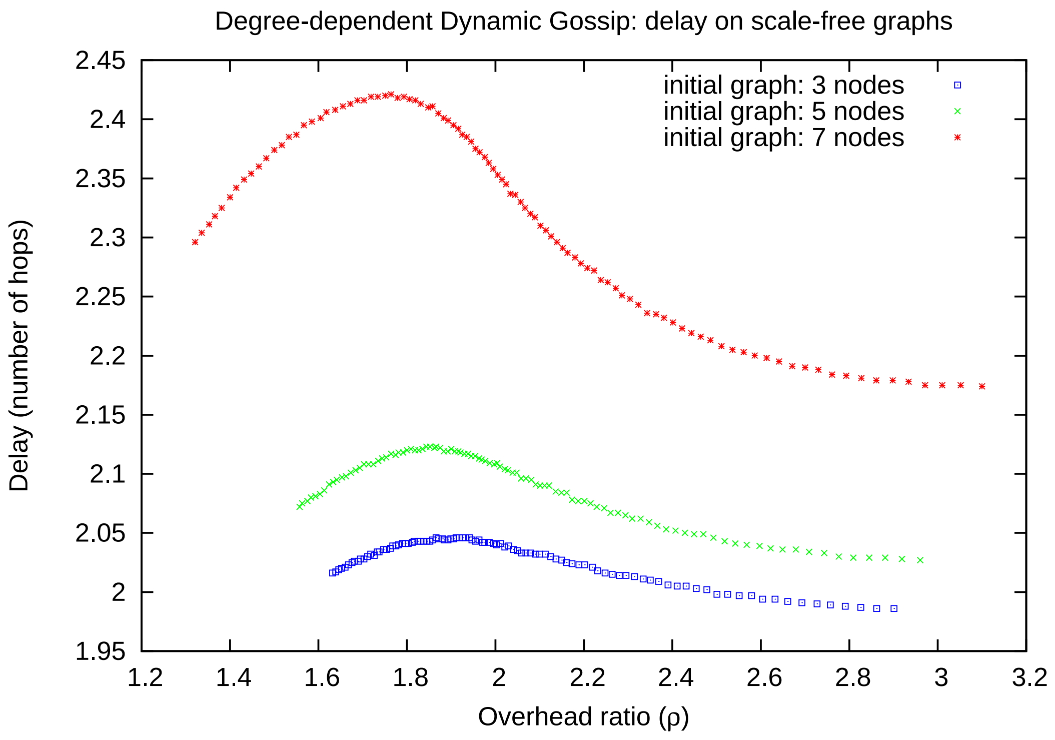

Figure 3.37: DDG with polynomial function: *delay* on scale-free graphs with 3, 5, and 7 initial nodes

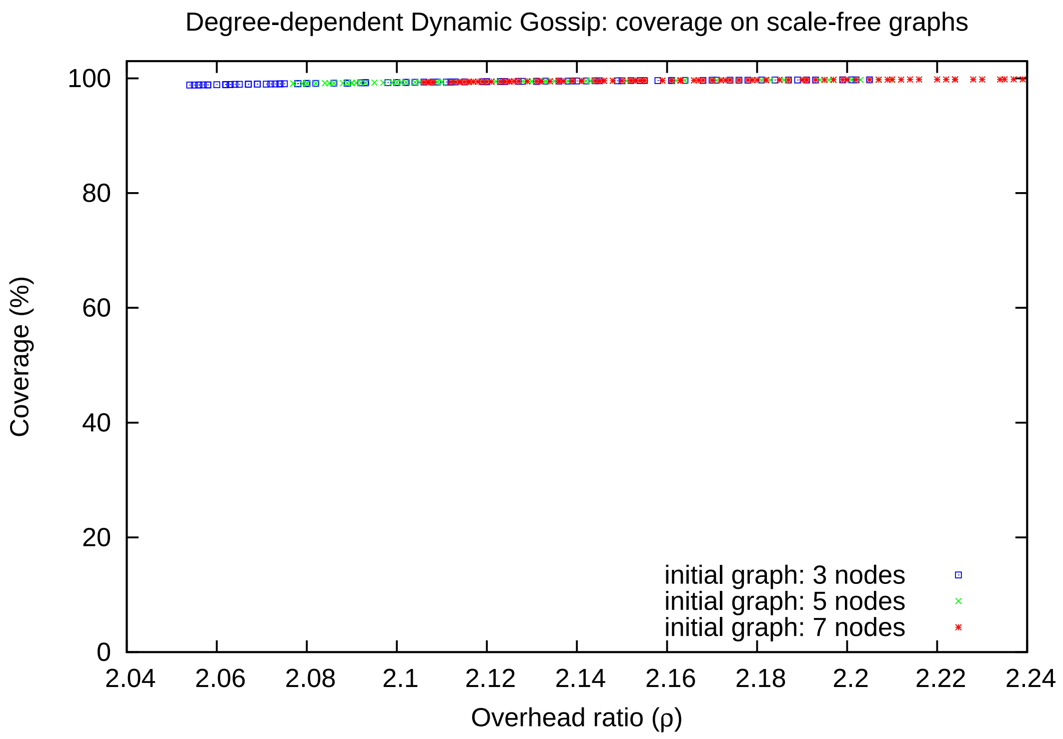

Figure 3.38: DDG with logarithmic function: *coverage* on scale-free graphs with 3, 5, and 7 initial nodes

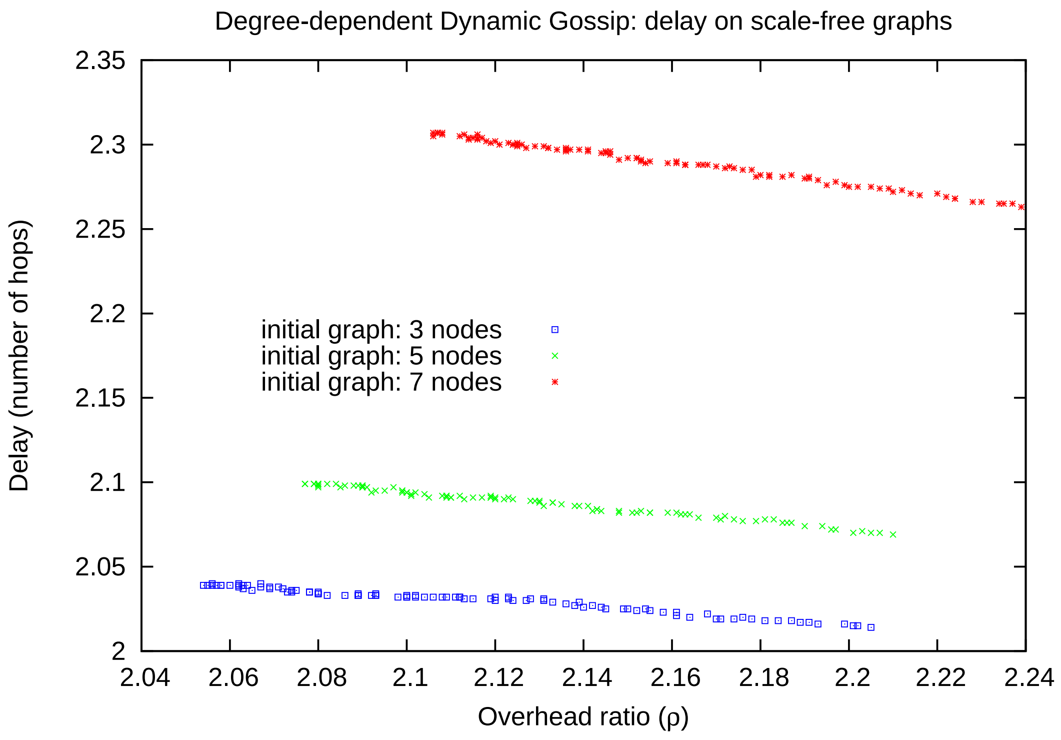

Figure 3.39: DDG with logarithmic function: *delay* on scale-free graphs with 3, 5, and 7 initial nodes

Chapter 4

Future work

Experimental evaluations have shown that DDG performs better than some of the most common push gossip approaches, obtaining a much higher coverage with a bit higher delay. However, time and limited computational resources didn't allow a deeper analysis of some further details of DDG, which could undergo a redesign in order to improve performances.

4.1 Network topology

Simulations have shown that DDG performs well on graphs with both high and low edge dependency, i.e. scale-free and random topologies. However, it has to be determined whether the algorithm could be considered fully topology-independent, so tests have to be performed on other known models (for example small-world). Moreover, limited computing resources didn't allow DDG to be tested on larger graphs (thousands of nodes) to verify its scalability.

4.2 Dissemination probability

The rationale behind polynomial and logarithmic functions is that dissemination probability has to be inversely proportional to a function of node's degree. However, further experiments should be performed to verify how other degree-dependent probability distributions behave on aforementioned topologies, and an a priori analysis of the most suitable distributions should be made. The tuning of dissemination probability may be also improved by adding a pull-based component to DDG: nodes could monitor message reception rate of their neighbors and eventually increase dissemination probability of 'neglected' nodes or decrease dissemination probability of node which generate too much network traffic, thus making DDG shift from a pure push to an adaptive approach (see 2.5).

4.3 Graph knowledge

The 'basic' version of DDG uses neighborhood dimension as the only lightweight information to tune dissemination probability. More sophisticated techniques can be designed and implemented, such as an heuristic by means of which each node can discover how far it is from the center of the network and exploit this information to decrease overhead on both network infrastructure and nodes. Other lightweight data structures may be forwarded together with messages, in order to keep further statistics and consequently tune dissemination parameters or even dynamically switch between different probability distributions.

4.4 Network dynamics

The last aspect which requires a deeper analysis is the behavior of DDG as well as other classic push gossip protocols towards modifcations of the simulated system, both statically and dynamically (i.e., during the whole simulation lifespan). More precisely, static elements which can be modified are nodes' cache size and maximum value of message's TTL, while the most important dynamics of networks is the change of topology and reachability, caused by node churn. Even in this case, the above parameters may be exploited to tune dissemination probability and, consequently, message forwarding rate.

Conclusions

Today, client-server paradigm is not efficient enough to be the only technique employed for data dissemination. Since a decade, especially after the birth of BitTorrent protocol, peer-to-peer paradigm has become one of the most used dissemination approaches, due to its efficiency, resilience and scalability, at the price of a possibly lower bandwidth than centralized architectures.

This thesis has addressed the problem of efficient data dissemination on peer-to-peer networks, focusing on the perspective provided by an almost new class of algorithms, the so-called 'gossip protocols'. The main purpose of gossip algorithms is to minimize the overhead on both network infrastructure and nodes, trying to maximize the spread of messages within the network. Many probabilistic forwarding approaches have been studied, each of them having pros and cons. In chapter 2 only some of the possible solutions to the problem have been analyzed and summarized, with particular regard to the most general and adaptable ones. In fact, literature is full of variants of gossip algorithms, specifically designed to perform well under certain hypotheses, related to both network infrastructure and algorithm's final purpose (for example, gossip protocols tuned to better adapt to video streaming). Moreover, it has been chosen to focus the analysis on those algorithms which have been considered almost completely 'free from constraints', in the sense that they are supposed to run in whatever environment with a few assumptions on it. The algorithms have been observed at a high abstraction level, by means of graph theory.

A large part of the material provided by literature has been the source of inspiration for Degree-dependent Dynamic Gossip, which belongs to the family of push gossip protocols, and whose experimental results are shown in chapter 3. DDG shows better performances than some of the most common push gossip algorithms, even if many other tests and improvements have to be performed, as described in chapter 4. To the best of my knowledge, gossip algorithms, as any other solution, have to be considered as one of the feasible alternatives to solve data dissemination problem on peer-to-peer networks, and their applicability must be evaluated for every instance of a specific problem.

Gossip is certainly a viable solution to many problems (see 1.4) and research can contribute by finding other fields of application. However, literature shows that, even in the case of gossip protocols, there isn't any 'silver bullet', which can completely solve the data dissemination problem in the most effective and efficient way. In fact, it's very important not only to understand whether gossip algorithms are a suitable approach to solve a problem, but also what kind of gossip approach is the most correct (push, pull, hybrid or others) depending on environmental constraints, such as network infrastructure, predicted workload and network size.

Bibliography

[1] OECD (2012), *Machine-to-Machine Communications: Connecting Billions of Devices*, OECD Digital Economy Papers, No. 192, OECD Publishing.

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

[3] Ruijing Hu, Julien Sopena, Luciana Arantes, Pierre Sens and Isabelle Demeure. *A fair comparison of gossip algorithms over large-scale random topologies.*

[4] Patrick T. Eugster, Rachid Guerraoui, Anne-Marie Kermarrec, Laurent Massoulié. *Epidemic information dissemination in distributed systems.* Computer, volume 37, issue 5, pages 60 - 67.

[5] 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.

[6] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, Maarten van Steen. *Gossip-based unstructured overlay networks: an experimental evaluation.* Technical Report UBLCS-2003-15, December 2003.

[7] João Leitão. *Gossip-based broadcast protocols.* Master thesis, Department of Informatics, University of Lisbon. DOI: http://hdl.handle.net/10455/3076.

[8] Lorenzo Alvisi, Jeroen Doumen, Rachid Guerraoui, Boris Koldehofe, Harry Li, Robbert van Renesse, Gilles Tredan. *How robust are gossip-based communication protocols?.* ACM SIGOPS Operating Systems Review, Volume 41 Issue 5, October 2007, Pages 14 - 18.

[9] Ming Cao, Daniel A. Spielman, Edmund M. Yeh. *Accelerated gossip algorithms for distributed computation.* In Proc. 44th Annu. Allerton Conf. Commun. Control Comput., Monticello, IL, Sep. 2006.

[10] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, Devavrat Shah. *Analysis and optimization of randomized gossip algorithms.* Decision and Control, 2004. CDC. 43rd IEEE Conference on, volume 5, pages 5310 - 5315.

[11] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, Devavrat Shah. *Gossip algorithms: design, analysis and applications.* INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE. Volume 3, pages 1653 - 1664.

[12] Paolo Frasca, Fabio Fagnani. *Broadcast gossip averaging algorithms: interference and asymptotical error in large networks.* arXiv preprint arXiv:1005.1292, 2010.

[13] Damon Mosk-Aoyama, Devavrat Shah. *Fast distributed algorithms for computing separable functions.* Information Theory, IEEE Transactions on, july 2008. Volume 54, issue 7, pages 2997 - 3007.

[14] Robbert van Renesse, Yawn Minsky, Mark Hayden. *A gossip-style failure detection service.* Middleware'98, 1998, pages 55 - 70.

[15] Zygmunt J. Haas, Joseph Y. Halpern, Li Li. *Gossip-Based ad hoc routing.* INFOCOM 2002. Twenty-First Annual Joint Conference of the

IEEE Computer and Communications Societies. Proceedings. IEEE. Volume 3, pages 1707 - 1716.

[16] Ali Ghodsi, Sameh El-Ansary, Supriya Krishnamurthy, Seif Haridi. *A self-stabilizing network size estimation gossip algorithm for peer-to-peer systems.* Technical Report T2005:16, SICS (2005).

[17] Mauro Franceschelli, Alessandro Giua, Carla Seatzu. *Load balancing on networks with gossip-based distributed algorithms.* Decision and Control, 2007 46th IEEE Conference on, pages 500 - 505.

[18] Mauro Franceschelli, Alessandro Giua, Carla Seatzu. *Load balancing over heterogeneous networks with gossip-based algorithms.* American Control Conference, 2009. ACC '09, pages 1987 - 1993.

[19] Eytan Modiano, Devavrat Shah, Gil Zussman. *Maximizing throughput in wireless networks via gossiping.* SIGMETRICS '06/Performance '06 Proceedings of the joint international conference on Measurement and modeling of computer systems, pages 27 - 38.

[20] Alexandros G. Dimakis, Soummya Kar, José M. F. Moura, Michael G. Rabbat, Anna Scaglione. *Gossip algorithms for distributed signal processing.* Proceedings of the IEEE, november 2010, volume 98, issue 11, pages 1847 - 1864.

[21] Konrad Iwanicki, Maarten van Steen. *The PL-Gossip algorithm.* Technical Report IR-CS-034.

[22] Pascal Felber, Anne-Marie Kermarrec, Lorenzo Leonini, Etienne Rivière, Spyros Voulgaris. *PULP: An adaptive gossip-based dissemination protocol for multi-source message streams.* Peer-to-Peer Networking and Applications, March 2012, Volume 5, Issue 1, pp 74-91.

[23] Márk Jelasity. *Gossip.* Self-organising software, 2011. pages 139 - 162.

[24] Spyros Voulgaris, Maarten van Steen. *Hybrid Dissemination: Adding Determinism to Probabilistic Multicasting in Large-Scale P2P Systems.* Middleware 2007, 2007. Volume 4834, pages 389-409.

[25] Spyros Voulgaris, Daniela Gavidia, Maarten van Steen. *CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays.* Journal of Network and Systems Management, June 2005. Volume 13, Issue 2, pages 197-217.

[26] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, Maarten van Steen. *The peer sampling service: experimental evaluation of unstructured gossip-based implementations.* Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware, 2004. Pages 79 - 98.

[27] Spyros Voulgaris, Maarten van Steen, Konrad Iwanicki. *Proactive gossip-based management of semantic overlay networks.* Concurrency and computation: practice and experience. Volume 19, issue 17, 10 December 2007, pages 2299 - 2311.

[28] Dionysios Kostoulas, Dimitrios Psaltoulis, Indranil Gupta, Kenneth P. Birman, Alan J. Demers. *Active and passive techniques for group size estimation in large-scale and dynamic distributed systems.* Journal of Systems and Software, october 2007. Volume 80, issue 10, pages 1639 - 1658.

[29] Mayur Deshpande, Bo Xing, Iosif Lazardis, Bijit Hore, Nalini Venkatasubramanian, Sharad Mehrotra. *CREW: A gossip-based flash-dissemination system.* Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on.

[30] Benoît Garbinato, Denis Rochat, Marco Tomassini. *Impact of scale-free topologies on gossiping in ad hoc networks.* Network Computing and Applications, 2007. NCA 2007. Sixth IEEE International Symposium on, pages 269-272.

[31] IEEE Computer Society. *1516-2000 - IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules.*

[32] 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.

[33] D'Angelo, G. and Bracuto, M. (2009) *Distributed simulation of large-scale and detailed models.* Int. J. Simulation and Process Modelling, Vol. 5, No. 2, pp.120 - 131.

[34] Gabriele D'Angelo, Stefano Ferretti. *LUNES: Agent-based Simulation of P2P Systems (Extended Version).* Proceedings of the International Workshop on Modeling and Simulation of Peer-to-Peer Architectures and Systems (MOSPAS 2011). As part of The 2011 International Conference on High Performance Computing and Simulation (HPCS 2011), ISBN 978-1-61284-382-7.

[35] Paul Erdös, Alfréd Rényi. *On the evolution of random graphs.*

[36] Albert-László Barabási, Réka Albert. *Emergence of Scaling in Random Networks.* Science, vol 286, 15 October 1999.

[37] Giulio Cirnigliaro. *Progettazione ed implementazione di strumenti per la valutazione di reti complesse con proprietà scale-free.* Bachelor degree thesis, 2011. `http://amslaurea.unibo.it/2664/1/cirnigliaro_giulio_tesi.pdf`.

[38] `http://pads.cs.unibo.it/dokuwiki/doku.php?id=`

[39] `http://igraph.sourceforge.net/`

Acknowledgements

The last part of this work is dedicated to all the people who help me enjoy every single aspect of my life every day. I'm sorry for foreign language readers, but I prefer writing this section using my mother tongue.

Paradossalmente, una delle parti più complicate ma, allo stesso tempo, fondamentali di questo lavoro è proprio quest'ultima sezione, quella in cui occorre condensare, in poco spazio, un insieme pressoché infinito di pensieri, parole, gesti e momenti di vita vissuta. Troppe sono le persone a cui devo come minimo un ringraziamento, perché in fondo ritengo che ognuno di noi sia fondamentalmente ciò che lo circonda: nessuno, infatti, può definirsi 'completo' se non viene completato dagli altri. I ringraziamenti sono talmente tanti, che non saprei neanche con quale criterio esporli per non dimenticarne nessuno: pertanto, mi affido ad un libero flusso di pensieri.

In primo luogo, desidero ringraziare chi sostiene la mia vita tutti i giorni da quando sono nato, affrontando anche le asperità che la vita stessa pone innanzi con una naturalezza ed una 'normalità' che credo abbiano poche persone, ossia i miei genitori: mio padre, che mi ha sempre sostenuto ed ha sempre invidiabilmente minimizzato e reso praticamente 'spontanea' ogni possibile situazione di apparente difficoltà. A mia madre, invece, non può che andare il più grande ringraziamento: per descriverla non basterebbero volumi che, comunque, sarebbero insufficienti; inutile perdermi in parole che sarebbero solo parziali o retoriche, ritengo sufficiente dire che senza il suo fondamentale apporto, non sarei mai arrivato a questo traguardo; grazie mamma. Un altro grande ringraziamento va a tutto il resto della famiglia, che mi ha sempre supportato durante il mio percorso di vita, se pur da una grande distanza geografica: zie, zii, nonni e cugini, sempre pronti a farsi in quattro e che non posso che ringraziare.

Un altro importante riconoscimento va e deve andare a tutte quelle persone che sono parte della mia vita da ormai tanti anni e con le quali ho condiviso e condivido sostanzialmente tutto, che sopportano tutte le mie ‘rotture di scatole’, che sono sempre stati presenti e che non posso che considerare un’ulteriore parte di famiglia: Taso, Ricci, Zacca, Berna, la Silvia, la Fede, la Bea, la Marty e tutti i ragazzi della 5°C, che sono entrati nella mia casa molti anni fa e con i quali ho vissuto e voglio vivere importanti momenti che rimarranno sempre lì, come un’istantanea, impressi nella mia mente.

Un ringraziamento enorme va a tutti coloro i quali hanno contribuito alla riuscita di questa ‘titanica impresa’ degli ultimi anni, sia di triennale che di magistrale, sotto tutti gli aspetti: grazie a tutto lo staff di via Sacchi e, in particolare, ai miei ‘angeli della portineria’ Cesare, Floriana e Franca, che hanno pazientemente ascoltato e risolto tutte le mie (e non solo mie) problematiche di ogni tipo, sempre con il sorriso sulle labbra. Grazie a tutti coloro che hanno allietato la mia permanenza durante il corso della triennale e che non saprei nominare tutti: Gianno, Zavo, Diego, Francesco, Mike, Marv, Cops, Comand, Nico, Matteo, Andrea, Lorenzo, Sara, Gessica, Elisa, Eleonora e tutti quelli che non ho citato per nome. Grazie anche a Fux, Fabri, Davide, Matteo, Buccio, Marco, Villa, Alex, Giacomo, Timothy e Viviana che hanno, invece, alleviato l’irto percorso della laurea magistrale. Un ringraziamento particolare, però, va a tre persone: un primo grazie va a Teo Rigoni, il mio eterno compagno di progetti, senza il cui contributo umano, informatico e didattico, alcuni esami non avrebbero avuto assolutamente la stessa riuscita. Altri due enormi ringraziamenti vanno a due persone senza le quali sarei ancora bloccato e non starei scrivendo queste parole: Max e Spada, due grandi menti e meravigliosi compagni di viaggio; sono stati e sono colleghi, consiglieri, veri amici e autentici mentori. Da loro ho avuto lezioni di matematica, informatica ed anche di vita, delle quali non posso che fare tesoro. Troppo ci sarebbe da dire: grazie, grazie, grazie.

In ultima istanza (ma solo cronologicamente), desidero ringraziare Gabriele D'Angelo, magnifico insegnante, ma anche impareggiabile relatore, grazie al quale mi sono appassionato al mondo delle reti in ogni suo aspetto. Ha sopportato le valanghe di mail inviate da me, noiosissimo tesista, per due tesi di laurea ed un tirocinio, sempre con disponibilità e gentilezza, guidandomi passo dopo passo verso l'agognata meta.

License