INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Topology-aware Gossip Dissemination for Large-scale Datacenters

Miguel Jorge Cardoso Branco

Dissertation submitted to obtain the Master Degree in
Information Systems and Computer Engineering

Jury

President:	Professor João Emílio Segurado Pavão Martins
Supervisor:	Professor Luís Eduardo Teixeira Rodrigues
Member:	Professor José Orlando Roque Nascimento Pereira

November 2012

Agradecimentos

I would like to thank my advisor, Prof. Luís Rodrigues, for this opportunity, as well as the insight and advice throughout the thesis.

I would also like to thank João Leitão for the fruitful discussions, essential to the completion of this work.

Additionally, I thank my colleagues and the members of the Distributed Systems Group, especially João Paiva, Sérgio Almeida and Mauro Silva for their comments and support.

Finally, I wish to thank my family, friends, and Marta Teixeira for all the patience and support during this work.

This work was partially funded by the research grant (PTDC/EIA-EIA/102212/2008) in the context of the "HPCI" project and by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds.

Lisboa, November 2012

Miguel Jorge Cardoso Branco

Aos meus pais, à minha irmã e à Marta.

Resumo

Os protocolos de difusão epidémica são extremamente robustos e distribuem a carga de forma uniforme por todos os participantes, permitindo também evitar os fenómenos oscilatórios que caracterizam outras formas de difusão fiável. Desta forma, são excelentes candidatos para suportar a difusão e/ou recolha de informação em centros de dados de grande-escala. Infelizmente, neste contexto particular, soluções alheias à topologia da rede podem facilmente saturar os comutadores nos níveis hierárquicos mais altos da rede.

Nesta tese apresentamos um protocolo epidémico para centros de dados, que designamos por Bounded Gossip, que assegura uma distribuição adequada da carga pelos comutadores da rede, evitando que o tráfego epidémico seja uma fonte de estrangulamento do sistema. O nosso protocolo combina características importadas de algoritmos propostos anteriormente, tal como o Rumor Hierárquico e o CLON, as quais são enriquecidas com um protocolo de gestão da filiação e mecanismos de controlo de fluxo cientes da topologia. Os benefícios da nossa solução são ilustrados por uma avaliação experimental que compara o seu desempenho com o desempenho de soluções existentes na literatura, em termos de carga imposta na topologia de encaminhamento, custo global e latência do processo de disseminação.

Abstract

Gossip-based protocols are very robust and are able to distribute the load uniformly among all processes. Furthermore, gossip-protocols circumvent the oscillatory phenomena that are known to occur with other forms of reliable multicast. As a result, they are excellent candidates to support the dissemination of information in large-scale datacenters. However, in this context, topology oblivious approaches may easily saturate the switches in the highest level of the datacenter network fabric.

This thesis presents a novel gossip protocol for datacenters, named Bounded Gossip, that provides an adequate load distribution among the different layers of the switching fabric of the datacenter, avoiding being a source of network bottlenecks. Bounded Gossip embodies techniques from previous protocols, such as Hierarchical Gossip and CLON, and combines them with a topology-aware membership maintenance scheme and a topology aware rate-based flow control scheme. The benefits from our solution are illustrated by an experimental evaluation that compares the performance of Bounded Gossip with that of other competing protocols, in terms of imposed load in the routing topology, overall cost of communication, and dissemination latency.

Palavras Chave
Keywords

Palavras Chave

Protocolos Epidémicos

Centros de Dados

Conhecimento da Topologia

Sistemas Distribuídos de Larga Escala

Controlo de Fluxo

Keywords

Gossip Protocols

Datacenters

Topology-awareness

Large-scale Distributed Systems

Flow Control

Índice

List of Figures

Acronyms

ECMP Equal-Cost Multi-Path

IP Internet Protocol

LAN Local Area Network

LDH Last Delivery Hop

RMR Relative Message Redundancy

TCP Transmission Control Protocol

TTL Time-To-Live

WAN Wide Area Network

1 Introduction

Gossip, or epidemic, protocols are based on the periodic exchange of information between pairs of nodes. Originally proposed for implementing database replication, gossip-based dissemination protocols have proved to be very effective in supporting reliable dissemination of information in systems with large numbers of participants. Two of the main reasons for their success are their robustness and the ability to distribute the load uniformly among all nodes. This type of protocols has been used for a large number of different purposes, including reliable broadcast, data aggregation, membership maintenance, among others.

Another important aspect of gossip protocols is that they can avoid the oscillatory phenomena that are known to occur with other forms of reliable multicast (Birman, Hayden, Ozkasap, Xiao, Budiu, & Minsky 1999). Also, these protocols are based on point-to-point interactions, not requiring the underlying network infrastructure to support IP multicast. All these properties make gossip protocols particularly well suited to operate in large-scale datacenters, especially for information dissemination and state reconciliation.

However, part of the robustness of gossip-based protocols derives from their intrinsic redundancy. At each time a node gossips, it interacts with a small number of other nodes, typically selected at random. The total number of interactions in the system is usually large, and a given node (or link) is not unlikely to receive (transport) the same gossip message multiple times. Therefore, gossip protocols may stress the network (or key components of the network routing infrastructure) in an undesirable fashion. This problem may be exacerbated if the gossip protocol is oblivious to the underlying network topology. In fact, if the network is abstracted by a fully connected *clique* (*i.e.*, a single group of fully connected nodes that are able to communicate directly between themselves), gossip exchanges among disjoint sets of nodes may appear completely independent. Note that different logical links connecting nodes in the system might use the same routing equipment at the underlay level.

1.1 Motivation

Because datacenter network infrastructures are typically organized in a hierarchical tree-like topology, topology-oblivious epidemic approaches may easily saturate the routing equipment in the highest level of the datacenter network fabric. Therefore, gossip protocols operating in datacenters must rely on some form of topology-aware approach. A number of topology-aware gossip protocols have been proposed in the literature, including HiScamp (Ganesh, Kermarrec, & Massoulié 2002), Hierachical Gossip (Gupta, Kermarrec, & Ganesh 2006), and CLON (Matos, Sousa, Pereira, Oliveira, Deliot, & Murray 2009). As it will become more clear in the remaining of the thesis, these protocols still exhibit some limitations, namely: *i*) despite trying to minimize the load imposed on high level routing equipment, the resulting load may still be excessively high, *ii*) resource usage is not very efficient, or, *iii*) their latency can be significantly larger when compared with flat gossip schemes.

This thesis proposes a novel gossip protocol designed to operate on datacenters. This protocol, named Bounded Gossip, is based on a topology-aware approach, whose features allow it to provide an adequate load distribution among the different layers of the switching fabric of a datacenter and to achieve a better resource utilization.

1.2 Contributions

This work addresses the problem of optimizing the operation of gossip protocols in typical datacenter topologies. More precisely, the thesis analyzes, implements, and evaluates a set of components which combined allow to achieve better resource utilization during the execution of a gossip protocol with no penalty to reliability or dissemination latency. As a result, the thesis makes the following contributions:

- It proposes three separate topology-aware mechanisms that serve as building blocks to our gossip-based protocol:
  - A low-cost membership maintenance scheme that imbues the overlay network (which encodes logical neighboring relations across nodes) established among nodes with partially deterministic topology-aware properties;

  - A robust dissemination scheme that is both topology-aware and partially deterministic and that strives to minimize the coordination costs among nodes;
  - A rate-based flow control mechanism, which is also topology-aware;

- It derives a novel epidemic protocol, Bounded Gossip, that combines the previous mechanisms to achieve better resource utilization in typical datacenter network topologies.

1.3 Results

Considering the contributions described above, the results produced by this thesis can be enumerated as follows:

- An implementation of the Bounded Gossip epidemic protocol including all components.
- An experimental evaluation of the devised solution, comparing it to previous systems found in the literature.

1.4 Research History

This work was performed in the context of the HPCI project (High-Performance Computing over the Large-Scale Internet, PTDC/EIA-EIA/102212/2008). I have conducted my Master's work in the Distributed Systems Group (GSD) of INESC-ID. During my work, I benefited from the fruitful collaboration with the remaining members of the GSD team working on gossip-based solutions, namely João Leitão and João Paiva.

An earlier version of this work was published in Branco, Leitão, & Rodrigues (2012).

1.5 Structure of the Document

The rest of this document is organized as follows.

Chapter 2 provides an introduction to the different technical areas related to this work.

Chapter 3 introduces Bounded Gossip, providing the motivation and specification for each of the three components that form it.

Chapter 4 presents the results of the experimental evaluation study based on simulations.

Chapter 5 concludes this document by summarizing its main points and providing some pointers for future work.

2 Related Work

In this chapter we survey the main techniques proposed in the literature to render the operation of gossip-based protocols *topology-aware*, *i.e.*, to exhibit communication patterns that better match the topology of the underlying network.

The chapter starts by providing a historical introduction to gossip, and follows with a description of the characteristics that define a gossip protocol that is oblivious to the underlying network topology. Then, we present useful metrics to evaluate the performance of gossip-based systems and show general applications of such protocols. We also introduce the current datacenter topologies, which should be considered in the design of topology-aware epidemic protocols. After that, we identify two general strategies to optimize gossip-based solutions with regard to the network properties, describing and analyzing a number of existing topology-aware gossip systems.

2.1 Historical Introduction to Gossip

Gossip (or epidemic) protocols are named after the social or natural process they mimic in order to disseminate information in a network of computer nodes: their operation is inspired by the way a rumor, or a disease, spreads in a population. Gossip was originally proposed as a mechanism for supporting database replication (Demers, Greene, Hauser, Irish, Larson, Shenker, Sturgis, Swinehart, & Terry 1987), offering eventual consistency between databases in a robust way while keeping the system algorithm simple and distributing the load evenly throughout all nodes. Such applications of gossip have been extended in later works, such as leveraging epidemic protocols to propagate changes to the databases as transactions instead of individual items (Agrawal, El Abbadi, & Steinke 1997). This class of protocols has also been used as a building block to reliable broadcast applications (Birman, Hayden, Ozkasap, Xiao, Budiu, & Minsky 1999), adding reliability to IP Multicast or avoiding its need altogether by

using point-to-point interactions. Gossip-based multicast has been a focus of intensive research, resulting in many new protocols including sophisticated overlay maintenance schemes to control the information flow in a more precise way (Leitão, Pereira, & Rodrigues 2007b). Another focus of research is data aggregation using epidemic algorithms (Gupta, Renesse, & Birman 2001), where nodes in the system calculate an aggregate value based on all the individual values of each node. The work of Renesse, Birman, & Vogels (2003) presents a commercial solution that uses gossip to aggregate data in a hierarchical fashion with strong security features. In large-scale enterprise scenarios, gossip is a common mechanism for membership maintenance, being used in widely used systems like Dynamo (DeCandia, Hastorun, Jampani, Kakulapati, Lakshman, Pilchin, Sivasubramanian, Vosshall, & Vogels 2007) and Cassandra (Lakshman & Malik 2010). Further uses of epidemic algorithms include failure detection (Renesse, Minsky, & Hayden 1998), where nodes gossip information about their peers to discover which nodes are active and which ones have failed; and publish-subscribe (Voulgaris, Rivière, Kermarrec, & Van Steen 2005), where events are disseminated only to nodes that have expressed their interest in them, instead of being broadcast to all the nodes in the system.

2.2 Topology-oblivious Gossip

We start by introducing the most common characteristics of gossip protocols that do not take the underlying network topology into consideration. We will start by defining a simple gossip protocol (named *Flat Gossip*) before we incrementally present typical strategies employed to improve the performance of epidemic protocols in more complex scenarios. Finally, we will show how to leverage these properties to build applications that use gossip as a building block.

2.2.1 Flat Gossip

Flat Gossip protocols present a periodic behavior, which can be modeled as a sequence of *gossip rounds*. In each gossip round, a node exchanges information with a predefined number f of other nodes selected uniformly at random from the entire system population (the parameter f is called the fanout). Therefore, if a node has some novel information that needs to be disseminated, after one gossip round this information is also known by other f nodes. Since all nodes engage in gossip rounds periodically, the number of nodes that know the information

(sometimes referred as *infected* nodes) grows exponentially with the number of executed rounds. As a result, in this setting, the time required for a piece of information to be propagated to all nodes in the system is, on average, $O(log(N))$, where N is the total number of nodes executing the protocol (Karp, Schindelhauer, Shenker, & Vocking 2000).

This protocol is known as flat gossip exactly because gossip exchanges are performed with nodes selected uniformly at random, with no concern for their location in the underlying topology. Implicitly, the protocol assumes that each node has access to the full system membership (such that it can select gossip peers uniformly). In very large dynamic systems, to maintain full membership at every node becomes prohibitively expensive, given that every join and leave of individual nodes would need to be propagated globally. Therefore, many practical flat gossip protocols operate based on partial membership information, *i.e.*, they rely on the availability of a companion *partial membership service*, that offers to each node a sample view of the full membership (Ganesh, Kermarrec, & Massoulié 2001; Voulgaris, Gavidia, & Steen 2005; Leitão, Pereira, & Rodrigues 2007b). Protocols based on partial membership exhibit the same properties of gossip protocols based on full membership as long as local views include a uniform sample of the full membership (Eugster, Guerraoui, Handurukande, Kouznetsov, & Kermarrec 2003).

2.2.2 Partial Membership Services

In the context of gossip-based protocols, the goal of a companion partial membership service is to provide each node with a sample of the entire system membership. The main purpose of using such a service is to increase the scalability of the system. In fact, any large-scale system is faced with system dynamics, *i.e.*, some nodes depart from the system (for instance, due to failures, maintenance, etc.) and new nodes are added to the system. This phenomenon is known as *churn* (Rhea, Geels, Roscoe, & Kubiatowicz 2004). To propagate these changes to all other nodes in a timely manner would consume a significant amount of resources. By providing nodes with partial views, the effects of churn can be localized.

A partial membership service, also called a *peer sampling service* (Jelasity, Guerraoui, Kermarrec, & Steen 2004), aims at providing each node with a uniform sample of the entire system membership, called a *local* or *partial* view. Usually, the size of the local view is larger than the fanout value f. Thus, in its operation, the gossip protocol selects f nodes at random from its local view, instead of selecting f nodes at random from the entire membership. Therefore, when

a partial membership protocol is used, gossip no longer operates on an overlay that is a fully connected clique; instead, it operates on an overlay network with topological properties that derive from the way partial views are built.

There are two main approaches for maintaining partial views, namely the *reactive* and the *cyclic* apporach. The reactive approach only changes the partial view of a node in response to a change in the membership of the entire system (i.e., when nodes leave or join) (Ganesh, Kermarrec, & Massoulié 2001; Ganesh, Kermarrec, & Massoulié 2002). The cyclic approach continuously shuffles the partial membership information maintained by the nodes (Voulgaris, Gavidia, & Steen 2005). Finally, there are also protocols that combine features of the two previous approaches (Leitão, Pereira, & Rodrigues 2007b).

2.2.2.1 Scamp

Scamp (Ganesh, Kermarrec, & Massoulié 2001) uses a reactive approach to provide a peer sampling service. In this protocol, nodes join the system by sending a subscription request to a contact node, responsible for forwarding that subscription to all the nodes in its view and sending an additional C copies to random nodes in the same view. Nodes receiving a forwarded subscription keep the joining node in their view with probability inversely proportional to the current size of their partial view. It was shown that the views of nodes in the system converge to having size $(C + 1)log(n)$, without nodes being required to know the value of n.

Similarly to the subscription mechanism, and because the views do not update in the absence of node joins, an unsubscription mechanism exists, where nodes warn their peers when they decide to leave the network. Moreover, nodes that do not receive messages for a large period of time and thus suspect they have become disconnected from the network due to failure of nodes that had its identifier in their neighboring lists, execute a resubscription using one of their neighbors.

2.2.2.2 Cyclon

Cyclon (Voulgaris, Gavidia, & Steen 2005) is a cyclic approach to build a peer sampling service. Nodes in the system periodically exchange subsets of their partial views with a neighbor in their peer list. Two versions of this shuffling mechanism were proposed: *basic shuffling* and

enhanced shuffling. In basic shuffling, nodes select a peer at random and send to it a subset of their partial view, including their own identifier. The receiving node replies with a subset of its own partial view excluding its own identifier. To update their views, each node first fills in any empty slots that may be available and then replaces the entries it sent in the shuffle message with the entries it received from its peer.

In enhanced shuffling, node entries have an associated *age counter* that encodes the time that has passed since the creation of that node identifier. Nodes initiating a shuffle first increase the age counters of all the nodes in their view. Then, instead of selecting a peer at random, they select the peer in their view with the oldest age counter as their shuffle target. As before, they select random nodes in their view to include in the shuffle message, and add their own identifier with age counter equal to 0. The partial views update is done in the same way as in basic shuffling.

If a node does not reply to the shuffle request, it is deleted from the initiator's view, as a simple failure detection procedure. In enhanced shuffling, failed nodes are more likely to be contacted and thus having links to them repaired (*i.e.*, removed from the overlay), because they will have greater age counters.

2.2.3 Gossip Strategies

In the previous sections we have stated that gossip rounds happen periodically and that in each round a node exchanges information with other nodes. Obviously, this is a simplified description of how gossip protocols may operate, and does not capture all the alternative implementations that have been described in the literature. A more detailed description of the possible strategies for information exchange can be made by considering the gossip exchange unidirectional and by distinguishing the node that triggers the exchange (the *initiator*) from the node that is contacted (the *target*).

In this more refined setting, if the information flows from the initiator to the target we say that the gossip protocol operates in *push* mode. Otherwise, we say that gossip operates in *pull* mode. Naturally, some protocols exchange information in both directions, thus combining push and pull in each round, in what is called a *push-pull* approach.

Furthermore, when the initiator sends new information to the target without first inquiring

which information the target already knows, we state that the protocol operates in *eager push* mode. If the initiator first sends small identifiers of new information and requests the target to return the list of identifiers for which it requires information, and only later sends the actual data, we state that the protocol operates in *lazy push*. Lazy push may be useful if the information to be transmitted is large and its availability at the target can be checked by exchanging first small identifiers. In this case, lazy push saves network bandwidth in exchange for a longer latency in the information dissemination.

Finally, a push protocol can also operate in pure reactive mode, i.e., a node may initiate a gossip round as soon as it receives new information, instead of waiting for a predefined interval (cyclic mode).

2.2.4 Practical Convergence Techniques

Another practical challenge that arises when gossiping with peers is how to manage large updates. To achieve convergence (*i.e.*, ensuring that every node has the same information), one has to take into account that it may be impractical for a node to transmit all the information it has received since the bootstrap of the system.

Epidemic convergence approaches are typically divided into two separate categories (Renesse, Dumitriu, Gough, & Thomas 2008): *anti-entropy* and *rumor mongering*. In anti-entropy systems, nodes gossip a number of the most recent changes to their state and merge any differences found, whereas in rumor mongering systems, nodes gossip new information for a limited number of gossip rounds. A key aspect to take into consideration is that while nodes in an anti-entropy system continuously gossip their state, nodes in a rumor mongering system stop gossiping in the absence of new information.

2.2.5 Metrics for Evaluating Gossip Solutions

To evaluate the performance of an epidemic protocol, one needs metrics that can grade the system on its key features. Such metrics are useful to determine the characteristics of the solution, informing potential users of what to expect from the system's behavior through tangible and meaningful values. They can also provide insight into what could be improved in the solution, as well as offer points of comparison to other solutions.

We now present some metrics to evaluate key properties of gossip protocols, noting the distinction of their implementation in broadcast systems and state reconciliation systems.

2.2.5.1 Reliability

The robustness of the protocol is one of the key properties of these systems, so it is commonly evaluated. In broadcast protocols, robustness is typically measured in terms of *reliability*, defined as the percentage of nodes which receive a given message (*i.e.*, the number of infected nodes). In state reconciliation protocols, reliability can be measured in terms of the percentage of nodes that eventually converge to a final consistent state.

Reliability is closely tied to the number of peers to which nodes disseminate each piece of information. Therefore, the fanout parameter of an epidemic protocol is a fundamental factor when considering the reliability of the system. Since gossip protocols follow a *bimodal behavior*, either only a negligible subset of nodes receives the information or almost all nodes do (Eugster, Guerraoui, Kermarrec, & Massoulié 2004; Birman, Hayden, Ozkasap, Xiao, Budiu, & Minsky 1999). Reliability should then be tested in both stable scenarios and in the presence of multiple faults, when the number of nodes that receive the information decreases to values that could be potentially insufficient to ensure full coverage of the dissemination process.

2.2.5.2 Latency

The *latency* of the dissemination process is also an interesting metric for these systems. It is typically defined in broadcast systems by the amount of time that the flow of messages take to infect all the nodes in the network. As a simplification, latency can also be measured in terms of the *last delivery hop* (LDH) (Leitão, Pereira, & Rodrigues 2007a) of a message. In epidemic broadcast protocols, LDH is the number of gossip rounds that a message requires to be delivered to all nodes. Note that this metric can easily be converted to latency (in time units) by multiplying LDH for the gossip round time duration (assuming that the rounds have a longer duration than the latency of the physical link with highest latency). As for state reconciliation protocols, the latency of the system can be measured as the time it takes for all nodes to converge on a final state.

2.2.5.3 Redundancy

As gossip protocols tend to generate redundant information to mask node failures, it is also useful to measure the link stress in terms of messages per link, to determine the overhead on the network.

Another strategy to identify the redundancy penalty specifically is counting the redundant messages that are transfered for each new piece of information. If we divide the number of redundant messages by the number of nodes in the system, we discover how many redundant messages each node received, on average. An issue with such an approach can arise when not all the nodes receive the information, skewing our metric into displaying a false average penalty. For that reason, the *relative message redundancy* (RMR) was proposed in Leitão, Pereira, & Rodrigues (2007a). RMR is given by the expression $\frac{m}{n-1} - 1$, where m denotes the number of message payloads sent during the dissemination of one message and n denotes the number of nodes that actually received the message. This metric measures the average number of message copies (besides the first) that each node received and thus the overhead of the dissemination process, accounting for scenarios where not all the nodes received the information.

2.2.6 Gossip-based Systems

After introducing the general flat gossip protocol and some additional features that are usually employed to increase scalability in practical scenarios, we will now demonstrate how gossip can be used as a building block for a set of example applications.

2.2.6.1 Bimodal Multicast

An epidemic protocol to disseminate information reliably could work using a reactive push strategy as described above. To account for peers that did not receive the message in the push dissemination, the nodes can maintain a cyclic lazy push strategy with fanout f, periodically sending the IDs of messages they know to f neighbors, which in turn request the messages that they did not yet receive.

As the ID summaries grow with the number of messages, it is useful to limit the number of rounds a message is kept in the lazy push summary before its memory can be reclaimed,

similarly to a rumor mongering state reconciliation strategy. Such an approach has been used by Birman, Hayden, Ozkasap, Xiao, Budiu, & Minsky (1999).

2.2.6.2 Scuttlebutt

An example of an anti-entropy state reconciliation protocol is proposed in Renesse, Dumitriu, Gough, & Thomas (2008), a system in which nodes keep replicas of each others' state. To update their vision of the state of a peer p, a node gossips with another (not necessarily p), exchanging the maximum version number they have for objects of p. Then, should the version numbers differ, the node with higher version number for p sends the other as many updates for p objects as it can (*i.e.*, with regard to the maximum gossip message size), in chronological order.

The chronological order and the limit of the number of updates mitigate the exchange of outdated information that could occur when the two gossiping nodes have different versions but neither of them is up-to-date. Because both nodes will have to gossip with an up-to-date node to receive the latest state, ideally, they should not be required to waste resources exchanging outdated information.

2.2.6.3 Astrolabe

Astrolabe (Renesse, Birman, & Vogels 2003) is a commercial gossip-based solution that aggregates information from several hosts in a hierarchical fashion. The hosts themselves are leaves in a tree of *zones*, writing their information as attributes in a *Management Information Base* (MIB). Non-leaf zones also own their own MIBs which consist of aggregate values of their child zones attributes. The process repeats itself recursively throughout the tree, with multiple tree levels and multiple zones per level. *Representative Hosts* of each zone are responsible for gossiping their attribute values with sibling zone representatives. The values are then aggregated by programmable *Aggregation Functions* (which can be written in SQL) that compute a resulting value out of all the values for an attribute in child zones. The solution also encompasses several security measures, including Certificates in Aggregation Functions

2.3 Datacenter Topologies

In this section we describe the typical topology of a datacenter. Our protocol is designed to operate efficiently on topologies similar to the ones described here. More precisely, we aim at achieving the following relevant characteristics: *i*) minimize the load imposed on the switches of the network infrastructure; *ii*) use resources in an efficient fashion, as to maximize the throughput with a maximum acceptable communication load; and *iii*) minimize the overall latency of the gossip dissemination process.

2.3.1 Three-tier Architecture

The most common architecture for large datacenter networks is the three-tiered architecture (Benson, Akella, & Maltz 2010). In this architecture, the network is characterized by three levels of routing equipment.[1]

The top tier, usually named the *core tier*, is composed of a single core switch that spawns multiple aggregation switches at the second tier. The aggregation switches connect multiple edge switches that form the edge tier. Those switches are typically, top-of-rack switches and connect directly to nodes. In typical configurations, each edge switch connects from 20 to 80 nodes (Benson, Akella, & Maltz 2010). In the remainder of the thesis we refer to the set of all nodes connected directly to an edge switch as a *cluster*.

To minimize the load imposed over the core switch, some configurations rely on multiple core switches. In this type of architecture each of these core switches owns an independent physical link to each aggregation switch, providing redundant paths between every pair of switches at the aggregation tier. Although in the work presented here we focus on architectures with a single core switch, we argue further ahead in the thesis that our solution can also accommodate these redundant topologies, by uniformly distributing the load across all existing core switches.

[1] We will use the word switches to refer to both routers and/or switches, as it is common practice in the related bibliography (Benson, Akella, & Maltz 2010).

2.3.2 Two-tier Architecture

In smaller deployments, the three-tiered architecture is usually simplified to a two-tiered architecture. This is achieved by eliminating the aggregation tier and having the edge switches connect directly to the core switch.

In both these architectures, exchanging messages between two nodes located in different clusters becomes significantly more expensive. These costs reflect on both communication latency and switch processing (note that in intra-cluster communication only the edge switch needs to process the information). This further motivates the development of gossip-based protocols that promote gossip locality.

2.4 Topology-aware Gossip

In topology-oblivious protocols, nodes assume their peers (and corresponding links) have uniform characteristics. For instance, they may assume each link to have the same latency and bandwidth, or each node to have the same processing capacity when such is not the case. A subtle variation of this (and of great importance to the type of networks introduced previously in this thesis) is the case where nodes assume all paths are independent and they share no common underlay links. This can produce an excessive load in links present in a great number of paths.

In contrast, topology-aware epidemic protocols try to address the heterogeneity present in the network. There are two different ways to make a gossip protocol topology-aware: one consists in acting at the peer sampling level, by selecting neighbors using some algorithm that takes the underlying network topology into consideration; another is to act at the gossip protocol level (for instance, by biasing the probabilities of selecting some particular members from the partial view). Naturally, some protocols, including our own, act at both levels. We discuss both strategies in the next paragraphs, introducing solutions that exemplify them.

2.4.1 Biasing the Membership

One approach consists in changing the membership service in order to bias the local views provided to each node (Ganesh, Kermarrec, & Massoulié 2002; Tang & Ward 2005; Leitão, Marques, Pereira, & Rodrigues 2009). This can affect the overlay topology in different manners:

for instance, higher capacity nodes may be assigned a higher number of neighbors than low capacity nodes; neighbors can be selected such that overlay links have a lower average cost (considering a particular criteria, *e.g.*, latency), etc.

It is worth noting that the biasing of the overlay topology can bring both benefits and disadvantages. If performed in the wrong manner, biasing the partial view may introduce undesirable features. One can easily introduce an artificial *clustering* effect across the neighboring relations. This might happen because nodes usually try to bias the neighboring selection using transitive properties. For instance, taking into consideration the network latency between nodes. This can lead nodes to organize themselves in a clustered way, where small and highly connected groups of nodes become weakly connected among them. It has been shown that clustering can have a negative effect on the time required for information to spread across the whole system (Lin & Marzullo 1999), as well as on the connectivity of the system, increasing the probability of network partitions (Kermarrec & Steen 2007), especially in the presence of node failures.

2.4.1.1 HiScamp

The HiScamp protocol (Ganesh, Kermarrec, & Massoulié 2002) is a distributed membership protocol designed on top of the Scamp peer sampling service (Ganesh, Kermarrec, & Massoulié 2001). It was proposed to alleviate stress in core routers, using a hierarchical overlay structure. The system organizes nodes into clusters according to a proximity measure. Conceptually, each cluster represents a single node in a higher level gossip layer.

HiScamp restricts the partial views of nodes such that, at the higher level, each cluster maintains at most one single link to each of the other clusters. This translates to the lower level in the following manner: for any two clusters of nodes, for instance cluster a and cluster b, there is only one node in cluster a with a link to a member of cluster b. This way, there is less traffic between clusters, as the cluster members divide among themselves the responsibility of sending information to other clusters. These levels form a hierarchy of arbitrary depth, repeating the responsibility division at each level.

Similarly to Scamp, the degree of the nodes updates dynamically with the number of nodes and clusters in the system, using a subscription process on node join and an unsubscription process on node departure. Although nodes can leave the system without executing the unsubscription process, the authors claim that a subscription lease system can be implemented.

This approach reduces network load on high latency links but requires a good proximity metric as soon as the time of joining the system. Unfortunately, the resulting overlay topology presents only few, and completely random, links among nodes in distinct clusters. This has a significant negative impact over the latency and reliability of a gossip protocol executed on top of it. The authors of HiScamp argue that their solution can be extended to cope with hierarchical topologies with several layers, however the proposed solution increases the negative effects identified above.

2.4.1.2 GoCast

GoCast (Tang & Ward 2005) is a reliable multicast gossip system that builds a multicast tree to disseminate messages quickly through the links with lower latency, using an eager push approach. The system uses a membership restriction strategy to build neighbor lists comprised of C_{near} nearby neighbors and (fewer) C_{rand} random neighbors. Although a node has $C_{near}+C_{rand}$ neighbors in the overlay, only a subset of those links are used as multicast tree links.

If a branch of the tree fails, nodes can still receive messages through a periodic lazy push gossip by their other overlay neighbors (peers in the neighbor list that are not connected by tree links, only by overlay links). A node gossips the message IDs to each of its purely overlay neighbors using a rumor mongering approach with the duration of only one round.

Based on results from experiments, the authors conclude that the optimal values for the neighbor parameters are $C_{near} = 5$ and $C_{rand} = 1$, claiming that one random neighbor is sufficient to provide almost the same connectivity guarantees as larger random membership views. The degree of the nodes is aggressively maintained and, thus, varies little from the configured total ($C_{near} + C_{rand}$). The degree maintenance process ensures symmetrical neighboring sets, and the same multicast tree is used for every dissemination, regardless of the source.

2.4.1.3 X-BOT

X-BOT (Leitão, Marques, Pereira, & Rodrigues 2009) is an adaptive scheme which enables the topology of an unstructured overlay network to continually adapt itself to optimize a performance criteria provided by a companion oracle (for instance, latency). X-BOT can be used

to maintain a topology-aware unstructured overlay over which a flat gossip protocol can be executed.

Each node in the system maintains a small symmetrical list of neighbors, called the *Active View*, containing both optimized gossip targets and some other non-optimized (or *unbiased*) targets to maintain connectivity and thus guarantee reliability. TCP connections to these neighbors are used as a simple failure detector, to minimize the impact of having a reduced neighbor list.

Each node has also a larger secondary view, named *Passive View*, that contains only random nodes. This view is periodically refreshed to reflect the arrival and departure of nodes in the network. Peers in this list are not considered when selecting gossip targets. Instead, the *Passive View* provides potential new additions to the *Active View* if more suitable neighbors are found. This process ensures a dynamic biased neighbor list, that converges to the best possible list instead of stabilizing at a local minima overlay configuration.

Due to the symmetrical list of neighbors, the optimization process cannot be executed by each node locally. The authors propose then a four node optimization process, which, despite introducing some overhead in the protocol's optimization steps, still maintains locality to the nodes and their neighbors, not affecting the rest of the network and keeping the relevant properties of a random overlay.

Unfortunately, X-BOT design can only distinguish between close and distant nodes, being therefore unable to capture more complex topologies, namely the hierarchical topologies with several levels that are usually used to connect nodes in datacenters, being therefore inadequate to support efficient and topology-aware gossip solutions for datacenters.

2.4.2 Biasing the Gossip Target

Another approach to integrate topology awareness in epidemic protocols is to associate weights to neighbors and use those weights to bias the probability with which a node chooses each neighbor as a gossip target (Lin & Marzullo 1999; Carvalho, Pereira, Oliveira, & Rodrigues 2007; Leitão, Pereira, & Rodrigues 2007a). The weight can capture a *utility function*, in which case the probability of choosing a given neighbor for gossiping is directly proportional to the weight; or a *cost function*, in which case the probability will be inversely proportional to the weight.

Typically, the computation of the utility/cost function is encapsulated within an architectural component commonly named *Oracle*. For instance, an oracle that gives information about the latency of the links could exchange PING messages with neighbors and register the observed round trip time so as to estimate link latency. Alternatively, another implementation of an oracle could measure the hops from a node to another using TRACEROUTE and thus provide a cost function in terms of hops instead of time units.

2.4.2.1 Directional Gossip

Directional Gossip (Lin & Marzullo 1999) is a reliable multicast gossip system that tries to improve Wide Area Network (WAN) gossip in two separate ways.

First, it distinguishes WAN gossip from Local Area Network (LAN) gossip, as to not stress routers that connect the different LANs. The distinction is made by employing a hierarchical structure of gossip levels. At the WAN level, Directional Gossip runs a single, but optionally replicated, gossip server, instead of delegating to the LAN gossip nodes the task of disseminating messages to the other LANs.

Second, it addresses the issue of poorly connected nodes. It does so by leveraging a neighbor bias strategy, assigning weights to each neighbor, where each weight is the number of link-disjoint paths known to exist between the node and the neighbor. Two paths are link-disjoint if they do not share any link at the overlay level. It then selects the gossip target with probability inversely proportional to the weight of the neighbor. To increase reliability, Directional Gossip also reactively floods (*i.e.* sends a received message without waiting for the gossip round time) the neighbors with weight lesser than a preset value k.

Additionally, the protocol also considers techniques to deal with heterogeneous link loss rates, proposing an alternative weight function that takes the loss rates of the links into consideration and gives less importance to links that fail with high probability.

The reactive flooding of each message may present an overhead in link stress when compared to cyclic gossip solutions with flow-rate control (*i.e.*, where nodes send a maximum number of messages per gossip round). However, the overhead is still smaller than using a pure flooding solution, where each node sends the message to its entire neighbor list instead of identifying critical neighbors.

2.4.2.2 Payload Scheduler

Carvalho, Pereira, Oliveira, & Rodrigues (2007) propose a lower layer to multicast gossip protocols, called Payload Scheduler, to approximate gossip multicast to that of a multicast tree, despite not building an explicit tree. The proposed Payload Scheduler behaves similarly to a gossip target biasing solution. It assigns weights to neighbors according to a number of different strategies but, instead of using the weights to select an appropriate gossip partner, they use them to decide whether to send the message using an eager or a lazy push approach.

This way, they try to minimize the redundancy of messages that normal epidemic protocols produce, by approximating the gossip dissemination to a multicast tree. This approach creates a tradeoff between bandwidth and latency: the fewer eager push transmissions that are made (and thus less bandwidth wasted on message payloads), the more likely it is that each node receives the message id before the message payload (resulting in increased latency due to the additional request for the payload).

Some of the proposed strategies are of particular interest to us. The *Time-To-Live* (TTL) strategy uses an eager push approach only for the first rounds of the dissemination of each message. This follows the intuition that as the number of rounds increases, so does the number of nodes that have already received the message, therefore decreasing the probability of the gossip target peer needing the message payload. The *Radius* strategy tries to optimize the latency of messages using an eager approach only on nearby neighbors, according to an arbitrary measure of distance. The *Ranked* strategy always sends the full message payload to and from nodes that have higher capacity. This approach reduces the latency penalty in gossips with those nodes, since they are the least likely to be affected by the bandwidth overhead.

By combining the strategies above, it is possible to decrease the radius threshold according to the number of rounds elapsed since the origin of the message. This can be useful to emphasize long links in the first hops of a message, distributing it evenly throughout the network, and then use only low latency links to disseminate it quickly around the holders of the message, simulating a message with many sources. The inclusion of the Ranked strategy prevents an unoptimized usage of high capacity nodes, applying the above safeguards only for nodes with potential bandwidth issues.

2.4.2.3 Plumtree

Plumtree (Leitão, Pereira, & Rodrigues 2007a), similarly to GoCast, creates a multicast tree to disseminate messages in an eager push approach. It maintains two distinct sets of peers as well, *eagerPushPeers* and *lazyPushPeers*. Contrary to GoCast, however, Plumtree does not restrict the membership of each node to a majority of low latency neighbors. Instead, it biases a random peer sampling, selecting the closer neighbors to form a tree.

A neighbor is deemed close (moved to the eagerPushPeers set and thus contributing a link to the tree) when a previously unknown message is received from that neighbor. In turn, neighbors that deliver already known messages are grouped into the *lazyPushPeers* set. Communication to this set is made by lazy push gossip.

Because the tree branches are created following the paths of the first broadcast message, it is inherently optimized for the sender of that specific message. Messages sent by different sources experience a higher latency unless multiple trees are built and maintained.

To overcome the latency penalty for multiple broadcast sources that results from the tree creation process, Plumtree includes an optimization that continuously updates the tree by promoting lazy links to eager links, when the former deliver messages with less hopcount than the latter.

It is possible to define Plumtree as a Payload Scheduler strategy (similar to the Radius strategy described above), where the probability of employing an eager push approach is 1 for neighbors with low latency links and 0 for the others. The difference is that Payload Scheduler modifies the behavior of a cyclic strategy, reducing the load on high latency links, whereas Plumtree implements a pure reactive strategy, only disseminating information the moment it is received, without waiting for gossip rounds.

2.4.2.4 Hierarchical Gossip

Hierarchical Gossip operates by relying on a non-uniform selection of nodes when gossiping, in such a way that each node has a higher probability of gossiping with nodes close to the initiator in the underlay. The solution can take into consideration several levels of distance, offering therefore a possibility for capturing the inherent complexity of typical datacenter's topologies.

More precisely, let N be the total number of nodes, L the number of levels in the hierarchical topology, and K a small constant such that $K^L = N$. If we define the list of closest neighbors as $View_1$ and the list of the more distant neighbors as $View_L$ (with additional partial views $View_2$ to $View_{L-1}$ containing peers that are increasingly distant), the probability of a node gossiping to a neighbor contained in $View_x$ is $\frac{1}{K^x} * p(N, K)$. The probability of selecting one of the closest neighbors is thus $\frac{1}{K} * p(N, K)$, the probability of selecting one of the next to closest neighbors is $\frac{1}{K^2} * p(N, K)$, and so on, where $p(N, K)$ is a normalizing factor equal to $(\sum_{j=1}^{L} \frac{1}{K^j})^{-1}$.

Unfortunately, the peer selection employed by hierarchical gossip is still completely random in nature, as it does not take into consideration the freshness of messages being gossiped when selecting nodes to forward them to. This reduces the efficiency of Hierarchical Gossip's resource usage, inducing excessive load on key routing components when the information is already disseminated to all the remote regions and therefore only local message exchange would be required.

2.4.3 Hybrid Strategy: CLON

CLON (Matos, Sousa, Pereira, Oliveira, Deliot, & Murray 2009) is a recent work that explicitly tackles the problem of supporting efficient and reliable topology-aware gossip protocols for, and between, datacenters. It does so by using an integrated approach, which combines a topology-aware partially deterministic membership service, with a topology-aware gossip dissemination scheme.

Like HiScamp, CLON leverages the Scamp protocol to build a topology-aware overlay network connecting nodes in a datacenter. The design of this membership service promotes the maintenance of distant neighbors by each node in the system. On top of this topology-aware membership service, CLON executes a gossip dissemination scheme that manipulates the probability of selecting an overlay neighbor to receive a gossip message according to the freshness of that message.

Moreover, CLON leverages a similar strategy to Payload Scheduler's radius strategy, where nodes propagate messages through lazy gossip to remote peers. Each node queues the lazy gossip identifier announcements that it receives and discards them if the payload for the referenced message is instead received through local push gossip.

However, CLON is mostly concerned with providing gossip support for services deployed across multiple datacenters. This clearly reflects on the membership service employed by CLON, which can only distinguish between close and distant neighbors, being therefore unable to capture hierarchical topologies with several tiers. This significantly increases the average dissemination latency of the solution.

Summary

This chapter presented gossip protocols, starting with a full membership flat gossip solution and increasingly adding features required by most practical scenarios. We introduced metrics for evaluating such protocols, before presenting common datacenter topologies where topology-oblivious epidemic solutions may not obtain satisfactory performance results. Finally, the chapter described the possible ways to add topology-awareness to gossip and discussed a number of existing topology-aware solutions. In the next chapter, we will present our own solution, Bounded Gossip.

3 Bounded Gossip

In this chapter, we introduce our solution, which we named Bounded Gossip. We start by explaining how we can abstract the physical topology in a way that allows our protocol to take advantage of it, following with the description of the protocol itself, detailing each component. We then discuss the fault tolerance considerations we took into account during the design of the solution, before discussing some remarks about the implementation of our system in the simulator we used for the evaluation process. Finally, we conclude the chapter by presenting an early version of our protocol that was modified to achieve the final solution.

3.1 Abstracting the Physical Topology

To make our protocol as generic as possible, we make a number of assumptions that allow us to abstract the physical topology, namely by considering that the naming scheme used to identify nodes encodes some information about the location of those nodes in the physical topology. Note that similar assumptions are employed by competing protocols (Gupta, Kermarrec, & Ganesh 2006).

We assume that all switches in the datacenter are numbered following a hierarchical naming scheme. Furthermore, we treat the identifier space of any set of switches connected to the same switch as a circular space. The child switches of another switch are numbered from 0 to $N - 1$, where N is the number of child switches.

Consider the following example: say node n is connected to the first edge switch that spawns from the second aggregation switch that is directly connected to the top-level core switch. We explicitly attribute the identifier 0 to this core switch as to allow our solution to be easily extended to a scenario with multiple data centers[1]. In this scenario, the identifier of the

[1]This problem will be addressed as future work.

aggregation switch is 0.1. Consequently, the identifier of the edge switch to which n is connected is 0.1.0. An example for this numbering scheme is shown in Figure 3.1.

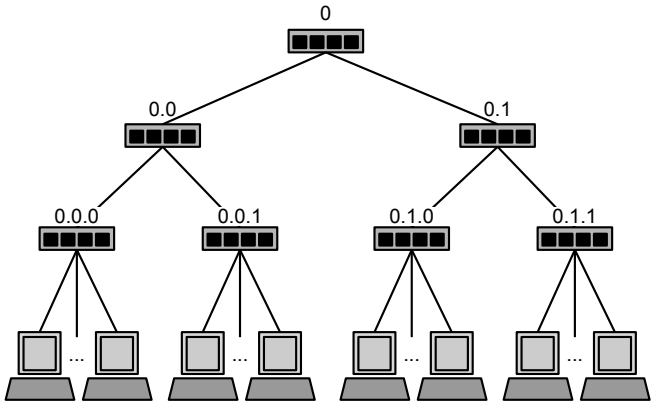


Figure 3.1: An example of the switch numbering scheme

We further assume that the IP address of a node enables any node to locally determine the identifier of the edge switch to which that node is connected. While the specific strategy to determine the node's location in the network from its IP address is outside the scope of this thesis, it is important to note that the translation process does not need to be direct, *i.e.*, an IP address of 1.2.3.4 may not mean that the node is connected to switches 1, 2 and 3 of the datacenter hierarchy. This notion is essential to allow our solution to support network topologies with different numbers of hierarchical levels.

3.2 System Architecture

In this section, we describe Bounded Gossip. We start by providing an overview of the Bounded Gossip building blocks and then proceed to make a detailed description of the operation of each of its components.

3.2.1 Overview

Our solution follows, and builds upon, the same intuitions behind the design of Hierarchical Gossip (Gupta, Kermarrec, & Ganesh 2006) and CLON (Matos, Sousa, Pereira, Oliveira, Deliot, & Murray 2009), which we refine and extend with new mechanisms.

Bounded Gossip is composed of three main components which complement each other to

provide a reliable, efficient, and topology-aware gossip dissemination scheme, that imposes a bounded load on switching components of the datacenter infrastructure. A peer sampling service is responsible for providing to each node in the system a set of partial views. The partial views are managed in such a way that their contents take into consideration the underlying network topology through the use of a limited form of determinism on the contents of those views. On top of this peer sampling service we devised a specially tailored gossip-based dissemination scheme, which leverages the characteristics of the resulting overlay. Our dissemination scheme induces a controlled amount of determinism to the gossip-based message dissemination pattern. This enables to lower the overhead imposed on switches while preserving fault tolerance. This dissemination scheme also takes into consideration the freshness of messages when forwarding them. Finally, the mechanisms above are complemented by a rate-based topology-aware flow control mechanism.

Our scheme can easily be configured to accommodate an arbitrary number of hierarchy levels in the datacenter topology. However, for the sake of clarity of exposition, we have opted to describe the operation of Bounded Gossip considering a three-tier network hierarchy. In the following we describe the operation of the three main components of Bounded Gossip.

3.2.2 Peer sampling Service

We rely on a gossip-based membership service that operates in a similar fashion to Cyclon (Voulgaris, Gavidia, & Steen 2005), where nodes periodically exchange samples of their partial views and update them with new nodes. In our solution however, each node maintains a set of distinct partial views, each view encapsulates information concerning each one of the hierarchical levels of the underlying topology. Furthermore, and contrary to other existing solutions, our membership service strives to induce a relaxed form of determinism in the way nodes are selected to fill partial views, such that it enables the emergence of topology-aware redundant tree-like topologies connecting clusters of nodes.

Similarly to Cyclon, every node periodically contacts a chosen peer and sends a sample of the contents of its partial views. When exchanging these samples, nodes also add their own identifier to the sample sent to its peer. Also similarly to Cyclon, we assume that node identifiers which are stored in partial views are enriched with an age counter, which is increased periodically by nodes to reflect the amount of time that has passed since the creation of that particular

identifier. More specifically, nodes increase the counters of the neighbors in their views before initiating a shuffle request.

When a node receives a sample of the system membership from a peer, it uses the enclosed information to update the contents of its local partial views, respecting the constraints that are imposed by our membership service, which we will describe in the next paragraphs. Additionally, nodes give preference to identifiers with lower age counters, as this increases the probability of the node that produced that identifier to be still active.

The partial views and the update process of our peer sampling service is very different from Cyclon. In our system, each individual node maintains L independent partial views of the system, where L is the number of hierarchical levels of the underlying network topology. We named these partial views PV_i where i indicates the hierarchy level encoded in the partial view contents (our solution supports an arbitrary number of hierarchy levels). Considering the 3-tier network topology discussed earlier, a node n owns the following partial views:

PV_0 represents the lowest hierarchy level of the topology. This view should contain all the identifiers of nodes in the cluster of n. To benefit the dissemination scheme, we assume that the identifier of n also appears in the PV_0 of n. Furthermore, the contents of these views are kept ordered considering the node identifiers. The size of this view depends on the network topology of the datacenter, having a size equal to the number of nodes in each cluster.

PV_1 contains identifiers of nodes which are reachable to n only by crossing a single aggregation switch. Nodes try to maintain in this view identifiers from K_1 deterministically chosen clusters. The preferred clusters of a node n are selected considering the id of the edge switch to which n is connected. Considering that n is connected to an edge switch with id $c.a.e$, n will give preference to nodes connected to switches with an id between $c.a.(e * K_1 + 1)$ and $c.a.((e + 1) * K_1)$ (notice that we assume that the identifier space of switches is circular at each hierarchy level). If one of the preferred identifiers is the same as n's edge switch identifier ($e \in [(e * K_1 + 1), ((e + 1) * K_1)]$), the node will ignore neighbors with that edge switch identifier and fill their slots in the partial view with nodes with edge switch identifiers the closest possible to the preferred ones.

PV_2 captures the highest hierarchy level of the underlying topology. This is achieved by storing identifiers of nodes which are accessible to n only by crossing the core switch. Similarly to PV_1, this partial view is built while trying to keep K_2 identifiers from different deterministically

chosen aggregation switches. Considering that n is connected through an aggregation switch with an identifier $c.a$, n will give preference to nodes connected through aggregation switches with identifiers between $c.(a * K_2 + 1)$ and $c.((a + 1) * K_2)$. The same replacement mechanism as described for view PV_1 takes effect for nodes with aggregation switch identifiers equal to a if a is one of the preferred identifiers. The edge switch identifiers are not relevant when managing the contents of this partial view. An example of such a bias is represented in Figure 3.2. In this scenario, we can observe some good properties of our membership scheme which will be leveraged by our dissemination process. If a message is generated in the first core zone (the blue zone), it can be disseminated to the next two zones, depicted in green and red. While the green zone can ensure the rest of the datacenter is infected by the message, the red zone is able to send redundant copies to the previously infected zones, which not only help the system recover from failures but also speeds up the delivery of the message in those zones by having more copies circulating when there are no failures. The core neighbors maintained by the rest of the zones are not depicted in the figure for clarity, but offer similar properties.

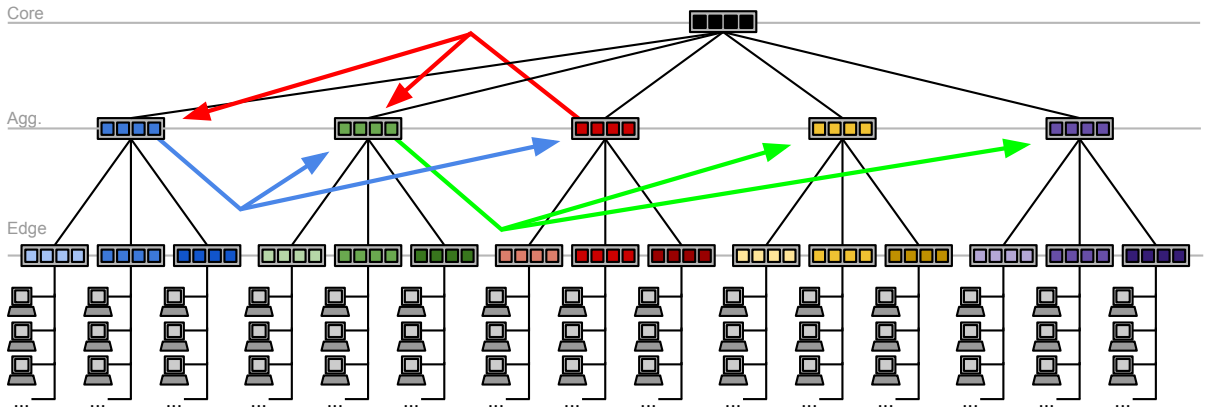


Figure 3.2: An example of membership bias at the core level

The K_i system parameters are chosen by the administrator and represent the size of the partial view at each level besides level 0. The determinism applied when choosing neighbors approximates the overlay links to that of a tree. In the absence of node failures, the dissemination process will then mimic the dissemination of a tree, minimizing dissemination latency. The size of samples sent in shuffle exchanges is also defined by the system administrator. Figure 3.3 illustrates an example membership for a node in the system.

To improve the reliability of our solution when node failures occur, we also rely on the following strategies: *i*) When exchanging samples of their views, nodes include the complete

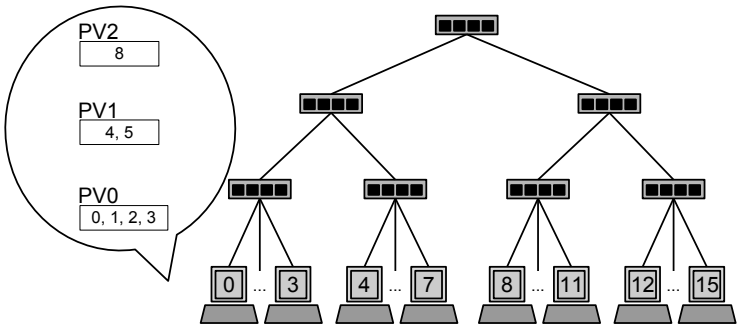
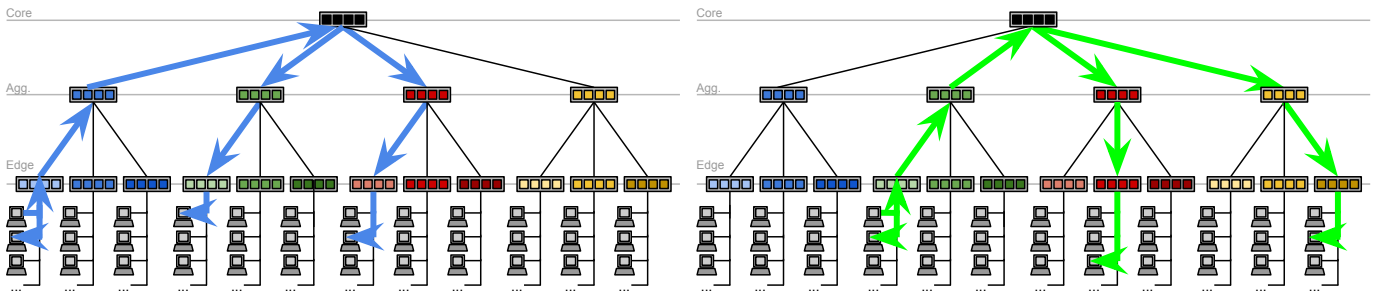


Figure 3.3: An example membership of node 0

contents of their PV_1 and PV_2 views. In scenarios with more hierarchical levels where including the contents of non-edge views is not practical (and would result in shuffle samples larger than intended), nodes can include only a subset of each view. This effect ensures that nodes exchange addresses of all types of neighbors. Otherwise, in deployments where PV_0 is much larger than the other views, partial views samples sent in shuffle exchanges would be frequently comprised of just neighbors from PV_0, which would have a negative effect in the number of live links maintained between remote neighbors. *ii*) They also remove from their partial views nodes that do not reply to previously issued shuffle requests. In this case, the membership service simply announces that such nodes have failed. This method of quick healing will not only stop a node from trying to contact failed neighbors but also from offering identifiers of failed neighbors in samples sent in the shuffle process. *iii*) Finally, as we discuss further ahead in the text, in our dissemination scheme nodes have specialized *roles*, such as edge, aggregation or core *roles*. When nodes with a role r choose another node with whom they exchange shuffle messages in each round, they bias their selection to promote exchanges with nodes on their PV_r view with probability of 99%. Otherwise, nodes interested in filling their core or aggregation neighbors slots would contact with great probability a neighbor in their PV_0, more likely to have the same remote neighbors as them and offering a large number of same-edge identifiers.

3.2.3 Gossip-based Dissemination Scheme

We rely on a gossip-based dissemination scheme that operates in a semi-deterministic fashion by leveraging the underlying topology-aware membership service. The algorithm is based on the principle suggested in a number of previous works, including CLON (Matos, Sousa, Pereira, Oliveira, Deliot, & Murray 2009), that in order to minimize latency, messages should be dissem-



(a) Step 1 (b) Step 2

Figure 3.4: Example of dissemination at the core level

inated primarily to remote nodes, and only then at more local levels. However, contrary to what happens in Hierarchical Gossip (Gupta, Kermarrec, & Ganesh 2006), nodes in our algorithm operate in the *infect and die* (Eugster, Guerraoui, Kermarrec, & Massoulie 2004) model, where each node processes a message only once. When a node processes a message for the first (and single) time, it (re)transmits it to f neighbors (where f is the *fanout* parameter).

To that end, and considering the 3-tier architecture mentioned previously, when an application-level message m is generated in our system, the source node starts to transmit it through the core links of the topology, disseminating message m to all the different zones of the datacenter connected to the core switch. When a source s sends m, it also forwards at least one copy of the message to its local cluster, to ensure the continuation of the dissemination in its own zone. An example of such behavior is present in Figure 3.4(a). The nodes that received the message sent in the first step will then continue the dissemination, leveraging the properties of the biased membership service. They will not only send the message to uninfected zones in the datacenter but also forward redundant copies of the message to areas that have most likely been previously infected but that can still be oblivious to the message due to node or message failures. Similarly to the first step, each node that forwards a copy of m also sends at least one copy to its cluster. An example of this behavior is shown in Figure 3.4(b).

When all the zones in the datacenter separated by core links are infected by the message, our system will start saving core switch load by stopping transmission at that level. Instead, nodes will focus on transmitting the message at the aggregation level to infect all the clusters in their core zone. Again, to ensure continued dissemination inside their cluster, the nodes will send at least one copy of the message to peers in their cluster. This process is illustrated in Figure 3.5.

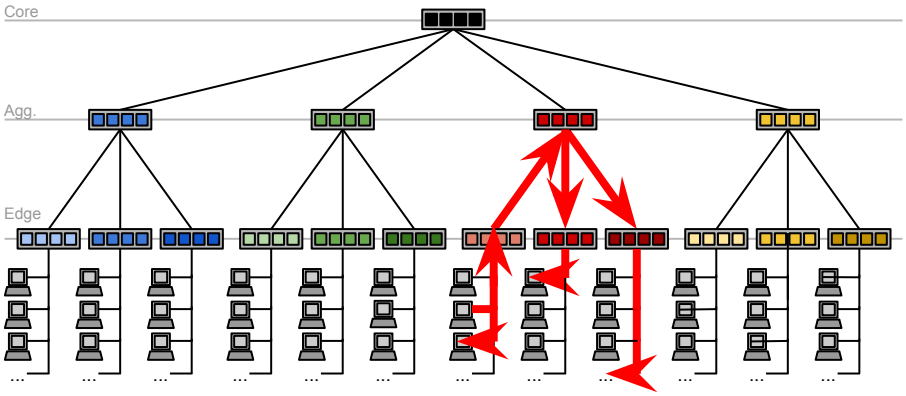


Figure 3.5: Example of dissemination at the aggregation level

Because nodes in the system are not able to know whether all the different zones in the datacenter are already infected by the message being transmitted, they use preconfigured round values to limit the number of rounds that a message can be disseminated using each of the hierarchy levels. In order to apply this strategy, that takes into account the freshness of messages, messages being disseminated by Bounded Gossip carry a T counter that indicates the number of times that the message has been retransmitted. To take the hierarchical topology into consideration, our dissemination process is controlled by a set of parameters $\pi_i, i \in [0, L[$, which limit the number of times a message can be retransmitted at each hierarchy level of the topology (notice that π_0 behaves as the typical *time to live* parameter of flat gossip solutions (Leitão, Pereira, & Rodrigues 2007b)). To know the required hierarchical level for a message, a node finds the maximum i such that $m.T < \pi_i$. The intended level for disseminating the message is thus i. This also means that the message still requires dissemination at all the levels lower than i, which will be considered at a later stage. By transmitting fresh messages through remote links we "parallelize" the message dissemination, avoiding cases where a message is known by all nodes in a cluster before being transmitted to a second cluster, taking therefore double the time to infect nodes in both clusters. These scenarios occur in overlays with a small number of remote links, such as HiScamp and some aggressive configurations of CLON.

However, to ensure that there is a bounded communication cost at each routing element of the datacenter network infrastructure, even when multiple messsages are being generated and thus scheduled for remote transmission, we rely on the PV_0 provided by the membership service to attribute, in a deterministic fashion, specialized dissemination roles across nodes of a cluster.

A replication factor R is used to attribute roles to nodes in the same cluster as follows: the

first R nodes in the PV_0 of elements of a cluster are chosen to disseminate messages using the highest hierarchical level of the datacenter network topology (they are thus called *core nodes*). The following R nodes in PV_0 are responsible for the dissemination at the next hierarchy level (*aggregation nodes*). Other nodes disseminate information only through the lowest hierarchical level (*edge nodes*). Please recall that PV_0 is a full view of the cluster and that it is sorted by node identifiers, so nodes can keep a consistent view of each member's role. In deployments with more hierarchy levels and partial views, similar roles are created and maintained.

Algorithm 1: Bounded Gossip Dissemination (part 1)

Variable Description

***f*:** fanout parameter; ***knownMessages*:** list of ids of received messages
***queue*:** local message queue; ***quota*:** available message quota
***r*:** hierarchical level of the node's role; ***time-to-live*:** messages' retransmission limit

```
1  upon event begin round do
2    quota ← resetQuota()
3    notified ← false
4    while queue ≠ ⊥ and quota > f do
5      msg ← queue.removeNextMessage()
6      h ← level(msg)
7      if msg.T < time-to-live do
8        quota ← quota − f
9        if h = 0 do
10         targets ← membership.getPeersInView(0, f)
11         newMsg ← msg.getCopyWithIncreasedLived()
12         for all peer ∈ targets do
13           trigger SEND(DATA, peer, newMsg)
14       else if h < r or r = 0 do
15         alreadySent ← 0
16         for role in [h to 0] do
17           targets ← membership.getPeersWithRole(role, f−alreadySent)
18           if role = h do
19             newMsg ← msg.getCopyWithSameLived()
20           else
21             newMsg ← msg.getCopyWithLivedFor(role)
22           for all peer ∈ targets do
23             trigger SEND(DATA, peer, newMsg)
24           alreadySent ← alreadySent + targets.size()
25           if alreadySent = f do
26             break for
```

Algorithms 1 and 2 denote the pseudocode for the dissemination procedure of Bounded Gossip. When a node produces or receives a message for the first time, it stores the message in its local queue (lines $57 - 60$). Our dissemination scheme is modeled to operate in rounds. Therefore, periodically in each round, each node checks its queue for messages and processes them until its quota is reached (the quota values are defined by the flow control mechanism and will be detailed later). Recall that each processed message is sent to f other nodes.

Algorithm 2: Bounded Gossip Dissemination (part 2)

```
27          else if myTurn() = true do
28            alreadySent ← 0
29            for role in [h to 0] do
30              if role = r do
31                targets ← membership.getPeersInView(r)
32                newMsg ← msg.getCopyWithIncreasedLived()
33                for all peer ∈ targets do
34                  trigger SEND(DATA, peer, newMsg)
35                alreadySent ← alreadySent + targets.size()
36                targets ← membership.getPeersWithRole(r)
37                newMsg ← msg.getCopy()
38                for all peer ∈ targets do
39                  trigger SEND(NOTIFICATION, peer, newMsg)
40                alreadySent ← alreadySent + targets.size()
41                notified ← true
42              else
43                targets ← membership.getPeersWithRole(role, f − alreadySent)
44                if role = h do
45                  newMsg ← msg.getCopyWithSameLived()
46                else
47                  newMsg ← msg.getCopyWithLivedFor(role)
48                for all peer∈ targets do
49                  trigger SEND(DATA, peer, newMsg)
50                alreadySent ← alreadySent + targets.size()
51                if alreadySent = f do
52                  break for
53      if notified ≠ true do
54        targets ← membership.getPeersWithRole(role)
55        for all peer ∈ targets do
56          trigger SEND(NOTIFICATION, peer, ⊥)

57    upon delivery DATA(sender, msg) do
58      if msg.id ∉ knownMessages do
59        knownMessages ← knownMessages ∪ msg.id
60        queue ← queue ∪ msg

61    upon delivery NOTIFICATION(sender, msg) do
62      if msg ≠ ⊥ do
63        if msg.id ∈ queue and level(msg) ≥ level(queue.get(id)) do
64          queue ← queue \ msg.id
```

For each message in a node's queue, there is a specific set of rules for its dissemination. Considering the message's round counter T, the node first discovers at which hierarchical level the message should be transmitted. Let h represent that level and r be the role of the node.

- If the message is already at the edge level (0), the node will forward the message to f nodes in its cluster, increasing the T counter (lines $9 - 13$).
- Otherwise, if $h < r$ but not 0, or the node has an edge role, the node redirects the message to the R nodes in its cluster responsible for retransmitting the message at level h (without increasing the T counter). Then, the node forwards the message to $f - R$ additional nodes in its cluster, starting by selecting nodes that are responsible for level $h - 1$ and so forth,

configuring T with appropriate values when forwarding the message for each level (lines $14 - 26$). This configuration is needed so that the next nodes do not repeatedly forward the message to the nodes responsible for message transmission at level h and lower. For instance, in the absence of this T configuration, the current node would first forward the message to peers with role h, $h - 1$, etc. Then, the nodes with role $h - 1$ would inspect the T parameter and discover that the message should be disseminated at level h, thereby forwarding it again to nodes with role h instead of processing the message at their own level.

- If $h >= r$, the node will start by sending the message to all the nodes in its cluster with roles between h and r (if applicable). Then, the node will perform its role, forwarding the message to all the K_r neighbors in the corresponding view (increasing the T counter), as well as to the other $R-1$ nodes in the cluster also responsible for level r. Furthermore, the node uses the remaining fanout to forward the message to additional nodes in its cluster, starting by selecting nodes that are responsible for level $r - 1$ and so forth, configuring T with appropriate values when forwarding the message for each level (lines $27 - 52$).

  However, this step is executed only by one of the R nodes responsible for level r in the cluster, using a deterministic criteria based on the number of the gossip round. This allows Bounded Gossip to avoid redundant transmission of messages. A message that is processed by one of the replicas responsible for level r can be discarded by the remaining replicas of the same level, unless the copy to be discarded required dissemination at a higher hierarchical level than the already processed copy (lines $61 - 64$). This additional verification is meant to reduce cases where older copies of the message were processed first, which was problematic in scenarios where the first copy required dissemination at the core level and the second copy only required dissemination at the aggregation level. One of the aggregation nodes of the cluster would process the second copy, just sending it through the aggregation switches and notifying the other aggregation replicas that the message had been processed. One of the other replicas had a copy of the message that required dissemination at the core level and would now ignore that requirement by removing the message from the queue. This erroneous behavior decreased the reliability of the protocol because the message would not traverse enough core links. To maintain synchronization between nodes with the same roles, they execute the dissemination strategy associated with their role in a round-robin fashion, ordered by increasing node identifier (recall that the

identifier space of each role is considered circular). When a synchronization error occurs and more than one node transmits at once, the nodes reconfigure the order taking into account the smallest node identifier across those that have sent messages in the previous round. Because the current active node is important for the synchronization, nodes notify their peers with the same role even if they do not send actual messages of that role (lines $53 - 56$).

This procedure allows to effectively propagate a message throughout all nodes in the datacenter in an efficient manner, while still promoting a controlled amount of redundant messages to mask both message omissions and node failures. The amount of redundant traffic is controlled by the parameter R, that also limits the number of nodes with the same role that can fail simultaneously with no loss of message reliability (even if $R - 1$ nodes fail, there is always one that received all the messages intended for that role). Our experiments have shown that configuring R with a value of 2 yields high fault tolerance to our dissemination scheme.

3.2.4 Flow Control Mechanism

In order to ensure a bounded dissemination traffic generated by Bounded Gossip, we rely on a simple, yet effective, distributed flow rate control mechanism. We remind the reader that each node maintains a queue which contains messages to be disseminated to its peers. To limit the number of messages transmitted per round, we use *quota* values for each node. Each round, nodes extract from its local queue a number m of messages such that $m \times f \leq quota$. Evidently this assumes that $quota \geq f$, *i.e*, the quota value is always greater than the fanout parameter of the protocol. The quota of each node depends on the node's role in the cluster, as there are different quota values for each hierarchical level, allowing for the limits of the dissemination traffic generated by Bounded Gossip to be finely-tuned across all levels of the topology.

To determine the dissemination load limit at each level i, we start by noting that there is exactly one node in each cluster sending messages through that level in each gossip round. In total, those nodes generate $Nclusters \times q_i$ messages per round. However, not all the q_i messages are sent through level i. The ratio of level i messages to total messages is $\frac{|PV_i|}{f}$. Multiplying the ratio for the total messages we calculated before yields our load limit for level i, $load_i = Nclusters \times q_i \times \frac{|PV_i|}{f}$.

Therefore, configuring the quota values to achieve a target load limit at each hierarchical level can be done through the expression $q_i = \frac{load_i}{Nclusters} \frac{f}{|PV_i|}$.

3.3 Additional Fault Tolerance Considerations

Because our protocol maintains a notion of node roles, it is crucial to guarantee that each role is fulfilled by at least one node at all times. While this is a simple goal in scenarios where no failures can occur, maintaining the roles in the presence of failures can present a challenge.

Besides the mechanisms described in the previous sections designed to improve reliability, nodes in the same cluster maintain TCP connections between themselves, as a simple failure detection mechanism. When the connection to a node fails, the other members of the cluster remove it from their views and reconfigure their cluster roles, as to ensure R replicas per level, as explained in Section 3.2.3.

In scenarios with a large number of nodes in each cluster, maintaining TCP connections to every edge neighbor may prove impractical. In those cases, it is possible to save some connections in the nodes with edge role by keeping TCP connections only to nodes which role must be replaced in case of failure. We have also experimented alternatives to the TCP connections such as flooding the PV_0 or gossiping the nodes failures as an internal cluster message. However, the first alternative either greatly reduces the quota values of nodes when failures occur or increases the edge traffic in those cases. The second alternative increases the recovery time and thus offers weaker reliability properties.

3.4 Implementation

For the evaluation of our solution, we decided to use the PeerSim simulator (Montresor & Jelasity), a Java simulator for peer-to-peer systems. PeerSim supports the use of two different engines, a cycle-based engine and an event-driven engine. Although gossip protocols modeled in rounds are easily supported by the cycle-based engine, we implemented all the tested solutions in the event-driven engine to leverage the added realism of message transport.

PeerSim prototype implementations are essentially composed of Protocols, Controls, and Initializers. Protocols encode the actual behavior of the system, Controls are run in configurable

moments of the simulation to monitor and perform modifications to the environment of the simulation, and Initializers are specialized Controls that are executed at the beginning of the simulation.

We implemented each solution by employing two PeerSim Protocols, one to model the membership service of the system and another to represent its information dissemination protocol. The membership Protocol implemented the PeerSim Linkable interface, which meant it was the one holding links to the peers of each node. We also developed our own interfaces that each Protocol implemented to encode information related to the datacenter topology, such as the position of each node in the network, abstracted in the way we previously described. By using these interfaces we were able to reuse the same Controls for every protocol instead of developing specific Controls for each. Naturally, we devised Initializers to assign the relevant information to each node.

The message transmission in the event-driven engine is achieved through the use of Transport Protocols, which can be configured to simulate different transmission delays or failure rates. One important aspect of the message transmission in our simulations is that we had to record the number of messages sent through each hierarchical level of the topology, to later use the data for extracting comparative results.

One of the most common flaws during the implementation process was an incomplete initialization of new instances of each Java Class. Because PeerSim relies on cloning objects instead of creating new ones, unexpected behaviors may occur when the programmer forgets to create new instances of attributes in the cloning method. This type of error is difficult to find because the protocol appears to function normally, and sharing an attribute throughout multiple instances of a Class can have a number of different effects depending on the attribute.

The simulator provides a good degree of flexibility of its components by employing a configuration method that relies on configuration files. Instead of changing and re-compiling the system, the programmer can simply change or add new configuration files to produce different scenarios. This is especially useful during the debug process, where it is of the utmost importance that the programmer can quickly experiment new cases and executions. Another important aspect of the configuration files is that they allow control of the seed of the randomness generator used by PeerSim during the simulation (it is also expected that the programmer always uses the same randomness generator), providing a quick way of reproducing any given execution.

To obtain results, we used Controls to monitor the simulation, writing relevant information in text files that later were treated by a number of Gnuplot scripts to generate most of the plots presented in the thesis and that would translate the results into something easily visible and interpreted.

3.5 Early Version of this Work

Before designing the final version of Bounded Gossip, an earlier design was considered and evaluated. In that version, the flow control mechanism was the same, but the other components had different behavior. For completeness, we now describe the differences between the two versions, regarding two topics: reliability and the *infect-and-die* model. A complete description of the early design was published in Branco, Leitão, & Rodrigues (2012). A comparison of the most relevant results will be presented in the next chapter.

3.5.1 Reliability Modifications

Previously, we introduced some strategies to improve reliability. In the early version of our protocol, some of those strategies were not present, resulting in weaker reliability results. The missing techniques were the following: *i*) Nodes only included random neighbors in their view samples, as opposed to biasing the selection to include non-edge neighbors. As discussed previously, this made the selection include much fewer remote neighbors and thus respond more slowly to failures, due to the reduced probability of finding active links that could replace links to failed nodes while still maintaining the restrictions we impose in our biased membership selection. *ii*) The membership service did not remove from the view the neighbors that did not respond to previous requests for sample exchanges. Due to that, there were times when nodes announced identifiers for failed peers, further increasing the number of nodes that kept those links and thus the probability of active nodes in the system trying to contact failed nodes. *iii*) The selection of the gossip peer with whom to exchange view samples was also uniformly random, which resulted in core and aggregation links being more static and thus less likely to be replaced by active links when needed (*i.e.* when the current link referenced a failed node).

The addition of these three strategies greatly improved the overall reliability of Bounded Gossip, as we will show in Chapter 4.

3.5.2 Infect-and-die Model

While the final version of Bounded Gossip, described previously in this chapter, adheres to the *infect-and-die* model, with each node processing a message exactly once, the early version of our solution deviated by a small margin from this model: in some cases, nodes could process the same message twice. This behavior occurred when a non-edge node received a message that required dissemination at a higher hierarchical level than that of the node's role.

Consider the following example, in a given three-tier architecture. Node a, responsible for sending messages through aggregation links, receives an event from the application layer requiring the generation of a gossip message, to be disseminated throughout the datacenter. Node a, running the previous version of our protocol, would inspect the message to find out it has a T counter of 0. Because such messages need to be disseminated using core links, node a would forward it to the nodes in the same cluster responsible for such transmissions. However, contrary to our final version, node a would not process that message immediately at the aggregation level, it would instead re-queue the message in its local queue, with a T counter configured to denote transmission at the aggregation level.

Note that aggregation nodes must fulfill their responsibility of disseminating these messages through aggregation roles, otherwise if all R aggregation nodes received the message prior to the aggregation dissemination phase (indicated by the T counter) they would just send the message to the core nodes and mark it as a known message, declining to process it later when they eventually received it for the second time, from those core nodes. Because the aggregation nodes in the previous version of our solution re-queued a copy of the message instead of immediately transmitting it at the appropriate level, there was a small deviation from the infect-and-die model. Additionally, the message was not sent through the aggregation level as soon as it could, and a small latency overhead could occur.

Summary

This chapter presented our solution, Bounded Gossip. We started by explaining how we abstracted the physical topology, encoding relevant information to be used by our protocol into switch identifiers. We continued by giving an overview of the solution, before detailing the different components of the system: the peer sampling service, the dissemination protocol

and the flow-control mechanism. For each component, we introduced the parameters of the system and explained how to configure them. Furthermore, we explained the fault tolerance considerations present in our work and offered alternatives to scenarios with different reliability requirements. Finally, we described the most relevant details of the PeerSim implementation, as well as a preliminary version of our own solution, discussing the shortcomings of that version.

The next chapter evaluates our solution against others found throughout the literature and which have been previously surveyed in Chapter 2. We also compare the different versions of our own protocol to better understand how some modifications affected its behavior.

4 Evaluation

To evaluate the performance of Bounded Gossip, we have simulated the protocol considering a datacenter network topology consisting of 1 core switch branching into 8 aggregation switches, each with 10 edge switches of clusters of 32 nodes. The total number of nodes in this network is $2,560$. All experiments were conducted using the PeerSim simulator (Montresor & Jelasity), using its event driven engine, as described in Chapter 3.

4.1 Experimental Settings

To offer comparative baselines, we have also executed other solutions found in the literature over this topology. In particular we have tested: *i*) a *Flat Gossip* solution operating over a full membership; *ii*) a flat gossip solution operation on top of *Scamp* (Ganesh, Kermarrec, & Massoulié 2001); *iii*) the *Hierarchical Gossip* solution operating with full membership information (Gupta, Kermarrec, & Ganesh 2006); and finally, *iv*) the CLON system (Matos, Sousa, Pereira, Oliveira, Deliot, & Murray 2009). All the protocol implementations were validated experimentally.

We configured every protocol to achieve 100% reliability. We set the f parameter of the Full Flat Gossip protocol to 13, and configured Scamp's redundancy factor C to match this degree. Due to the number of parameters of the CLON protocol, we decided to conduct the experiments with 3 different configurations: CLON1 adds the nodes to the system randomly, achieving a total number of core connections close to 7.5% of all connections; CLON2 uses a redundancy value C large enough to ensure a high degree for every node, so we can limit f and the core round limit; CLON3 uses an external method to add the nodes to the system, using as contact a node in the closest hierarchical level possible, allowing for a smaller number of remote connections. For the Hierarchical Gossip solution, we manually selected a probability generating value K of 6 to artificially increase the probability of a node using the non-edge links and thus

achieve the target reliability. We tested each protocol in a cyclic, *infect-and-die*, model, adding quota limits so we could limit the core load equally.

4.2 Evaluation Criteria

To better introduce our evaluation results, we will now present the most important concepts and metrics used in the next sections.

Core Switch Load represents the load in the core level switch(es) of the datacenter network fabric. We measure the core switch load in terms of messages, assuming that every message has the same size.

Aggregation Switch Load similar to the above for the aggregation switches.

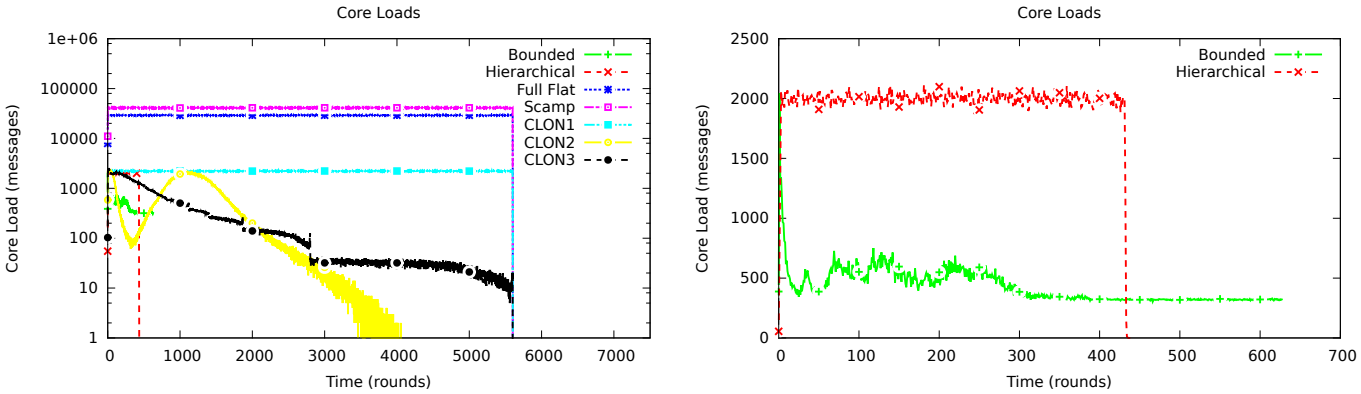
Edge Switch Load similar to the above for the edge switches.

Latency represents the number of gossip rounds that a message took since its generation to be delivered to all participants.

Throughput of a solution is the amount of messages that solution can deliver to all participants given a constraint to the number of total core level messages it can generate.

Reliability of a solution represents the percentage of generated messages that reach all the participants in the system. To obtain a finer detail, we calculate the percentage of nodes that received each application-level message and do an average of those values. Otherwise, if we counted only messages that reached every node, a message that reached almost all nodes would count as a 0 for the average, the same as a message that reached only a few nodes).

Load Distribution per Node shows how many protocol messages each node sent on average per application-level message generated in the system. We count the number of total messages sent by each node and divide it by the number of generated messages.



(a) All protocols

(b) Hierarchical and Bounded Gossip

Figure 4.1: Core switch load

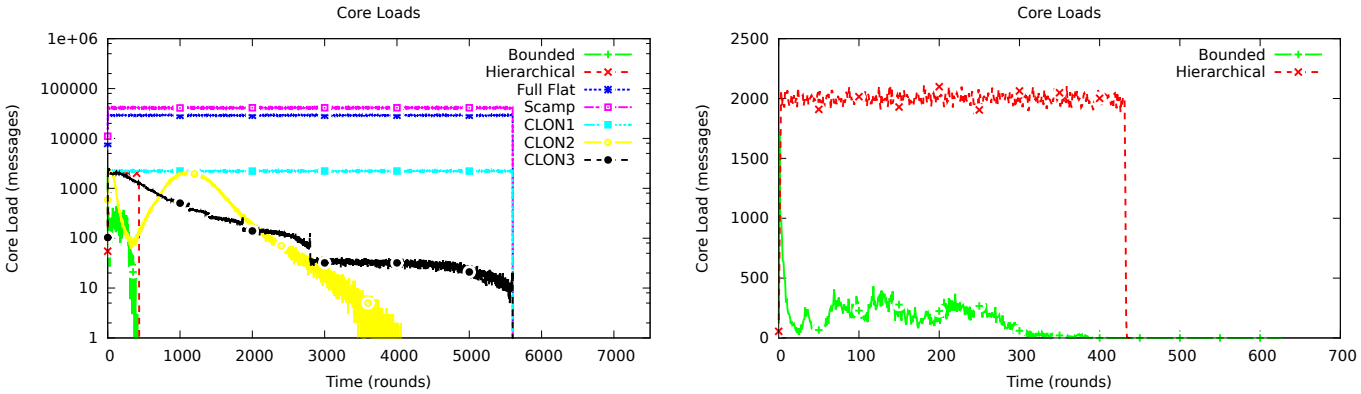
4.3 Core Switch Load

For the core switch load experiment, we injected 800 messages in the system every 10 Gossip rounds, for a total of 5,600 messages. We measured the core link load at each round. Figure 4.1(a) shows the resulting core load.

We can see that even the absolute minimum core switch load produced by Full Flat Gossip and Scamp greatly exceeds the configured quotas of the other protocols. This behavior is achieved by processing only one message at each gossip round, meaning that no less core switch can be produced while still executing the topology-oblivious protocols without some sort of flow control synchronization. The CLON1 configuration was not able to leverage the fanout and core rounds limit (otherwise it would lose reliability due to the reduced number of remote links used for each message) and therefore uses the maximum allowed load during the entire simulation.

Hierarchical Gossip maintained the maximum core switch load during all the simulation, wasting more total resources than Bounded Gossip, which only transmitted young messages through core links. It is also visible that the maximum core load in Bounded Gossip is only achieved in the scenario where 100% of messages are new and must be transmitted through core links. When both new and old messages are being transmitted, the core switch load is diluted through the rounds. A closer look at both protocols' behaviors is provided by Figure 4.1(b).

It is important to note that Bounded Gossip is the only solution that sends membership messages (*i.e.*, messages that do not contribute to the dissemination of application-level messages



(a) All protocols

(b) Hierarchical and Bounded Gossip

Figure 4.2: Core switch load (only dissemination messages)

but instead focus on maintaining the connectivity of the overlay network and the desired properties of the partial views) during the dissemination process. Scamp and CLON use a subscription mechanism that only sends messages when nodes join the system, which in our simulations is a process that happens exclusively before the generation of the messages. Our implementation of the remaining protocols operates over a full membership view. While this clarifies the contributions of each solution regarding the total number of messages sent (it reduces the likelihood of the results being confused with results of an implementation of any peer sampling service), such deployments of epidemic protocols are unfeasible in production environments, where systems have to deal with nodes joining and departing the system and maintain an updated membership view. For that reason, and in the interest of fairness, it is important to add to the comparison the number of messages sent by Bounded Gossip excluding membership messages. The results are visible in Figures 4.2(a) (for all the protocols) and 4.2(b) (for a detail of Hierarchical Gossip and Bounded Gossip).

In this comparison, it is even more evident that Bounded Gossip offers the best resource utilization, adapting the number of core messages to the amount of new application-level messages being generated in that time frame.

Another important set of results to compare is the number of core membership messages spent by the final version of Bounded Gossip and the one spent by the early version of the protocol. Because we changed the selection of the gossip peer with whom to exchange view samples, biasing each node of the final solution to select peers in the view corresponding to their role, the number of core gossip exchanges will be different from the early version. We present

the measurements in Figure 4.3.

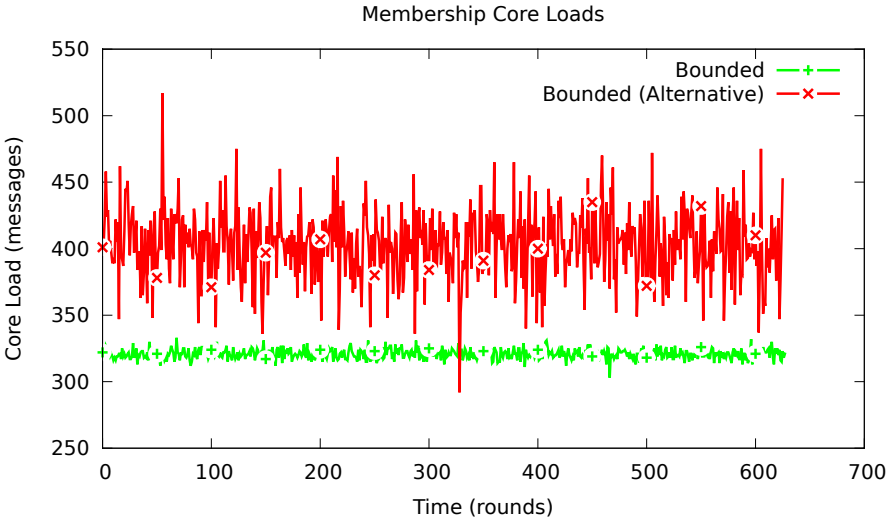
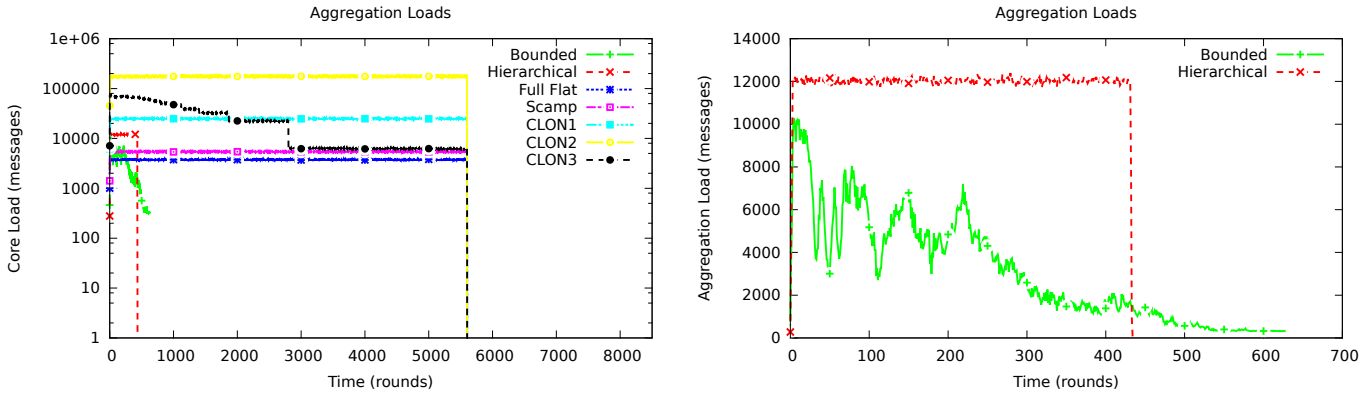


Figure 4.3: Core membership messages sent by two versions of Bounded Gossip

While the early version of Bounded Gossip presents a higher number of core membership transmissions and a higher variance each gossip round, the final version of our protocol is able to stabilize the number of contacts made in each round at a much lower number. This is to be expected given the algorithm and the simulation configuration. In both protocols we used values of $R = 2$, $PV_2 = 3$ and $PV_1 = 4$, and the topology has 80 different clusters, each of 32 nodes. We can then estimate the average number of transmissions for the old version of our protocol: the probability of a given node selecting a core contact is $\frac{PV_2}{32-1+PV_2+PV_1} = \frac{3}{38}$. Because we have a total of 2560 nodes, the number of core membership contacts per round, on average, is $\frac{3}{38} \times 2560$. We have to account for the answers, so we can estimate the number of core membership transmissions to be $\frac{3}{38} \times 2560 \times 2 = 404.21$. On the other hand, the number of core membership contacts in the final version of the protocol can be estimated in the following manner: there are 2 nodes in each of the 80 clusters with a core role, contacting a core neighbor with probability of 0.99. Therefore, the number of such contacts is $0.99 \times 80 \times 2$. We have to add that to the core contacts that happen at random when the nodes choose not to contact their role peers (*i.e.*, 1% of the time), which is equal to 0.1×2560 multiplied by the probability of selecting a core neighbor which is $\frac{3}{38}$ as we have shown before. The estimated number of contacts is thus $0.99 \times 160 + 0.01 \times 2560 \times \frac{3}{38}$, which multiplied by two to account for the replies yields 320.84.



(a) All protocols

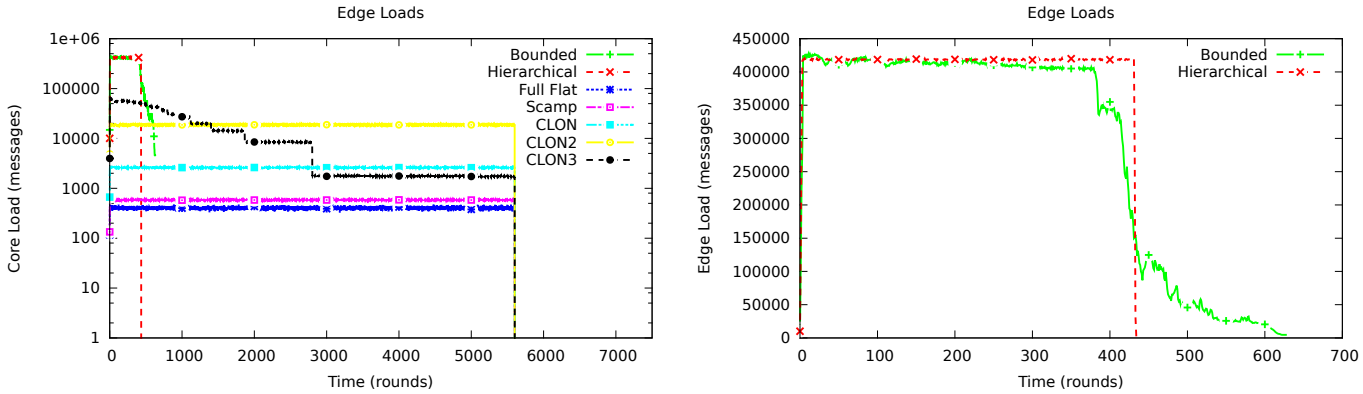
(b) Hierarchical and Bounded Gossip

Figure 4.4: Aggregation switch load

While it is clear that our solution greatly reduces the core load on architectures with a single core switch, we argue that such results are also extensible to architectures with multiple core switches. In these topologies, the core switches typically offer multiple redundant paths between every pair of aggregation switches. Such architectures typically employ a load distribution scheme similar to the *Equal-Cost Multi-Path* routing (ECMP) (Vahdat, Al-Fares, Farrington, Mysore, Porter, & Radhakrishnan 2010). ECMP operates by selecting the core switch used to route each message by leveraging consistent hashing over the message to be routed This usually translates into a uniform load distribution across these core switches. As our solution in no way affects the uniformity provided by consistent hashing over messages, it is expected that Bounded Gossip, when compared with competing solutions, yields lower load over each available core switch. Therefore, Bounded Gossip promotes a more efficient utilization of all available core switches in these types of network architectures, as effectively as it does in networks with a single core switch.

4.4 Aggregation Switch Load

We also measured the load in the aggregation-level switches (without counting the load in these switches induced by the messages sent at the core level) in the virtual datacenter. Results are shown in Figure 4.4(a). In this case, the topology-oblivious solutions have a reduced load only because the number of connections to core nodes greatly exceeds that of connections to aggregation nodes. Still, Bounded Gossip is on par with these solutions and outperforms



(a) All protocols

(b) Hierarchical and Bounded Gossip

Figure 4.5: Edge switch load

Hierarchical Gossip and the different CLON configurations.

Naturally, the missing distinction between edge links and aggregation links in CLON generates a high traffic in the aggregation layer of the topology, whereas Hierarchical Gossip suffers from having only one configuration parameter to tune the probability of gossiping to all the hierarchical levels. A detail of the comparison between Bounded Gossip and Hierarchical Gossip can be seen in Figure 4.4(b).

4.5 Edge Switch Load

We measured the load created by edge-level messages as well, excluding the previously analyzed traffic (remember however that all the above messages induce load in the edge layer as well). The results are summarized in Figure 4.5(a). Again, the topology-oblivious protocols create a small number of edge links in the overlay, which causes the uniform peer selection to produce few edge messages. Unfortunately, edge switches have to process the messages sent in the upper layers as well, being therefore just as busy. The peer selection bias in Hierarchical Gossip and Bounded Gossip is clearly visible, followed by all the CLON configurations. We show a more clear comparison between Hierarchical Gossip and Bounded Gossip in Figure 4.5(b).

It is important to note that despite reducing the load in the higher levels of the topology, Bounded Gossip does not compensate for that with more overhead than Hierarchical Gossip's in the edge switches, instead maintaining a close (slightly lower) traffic in edge messages. Also

important is the fact that the load values include the messages exchanged by the shuffles executed by the peer sampling service.

4.6 Latency

We also measured the latency distribution (in rounds) of all the application-level messages generated in the scenario above, to compare the overall latency of the dissemination process between the various solutions. The results can be seen in Figure 4.6.

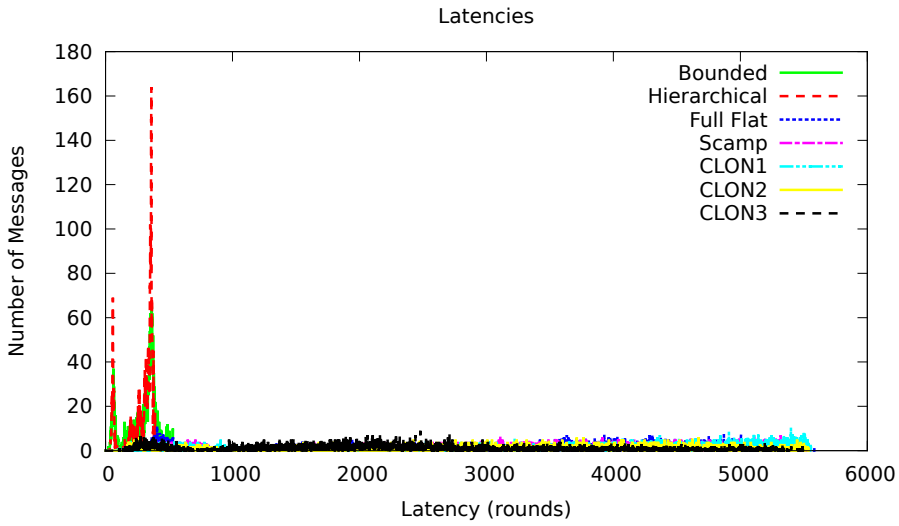


Figure 4.6: Latency results

Both topology-oblivious approaches show that due to the quota limitation with the objective of reducing the core load, the small number of messages processed in each round penalizes latency in an unfeasible way, achieving latency values for some messages of over $5,000$ gossip rounds. While the same is true for the two first configurations of CLON, the CLON3 setting is able to reduce the average latency of the messages, having fewer occasions where messages take more than $3,000$ gossip rounds to infect all nodes. The protocols with lower overall latency are Hierarchical Gossip and Bounded Gossip, delivering messages with an average latency of around 300 gossip rounds.

4.7 Throughput

For the throughput experiments, instead of limiting the core switch load per round, we limited the total core switch load induced during the entire dissemination process, to simulate the expected throughput of the different solutions when there is a limit of the core switch load in a given time frame. To simulate this scenario, we configured our message transport to drop all core messages sent after the limit was reached. Then, we periodically generated messages in the system, such that no two messages were disseminated at the same time. For each protocol, we then observed the number of application-level messages they were able to deliver to every participant. Figure 4.7 illustrates the results.

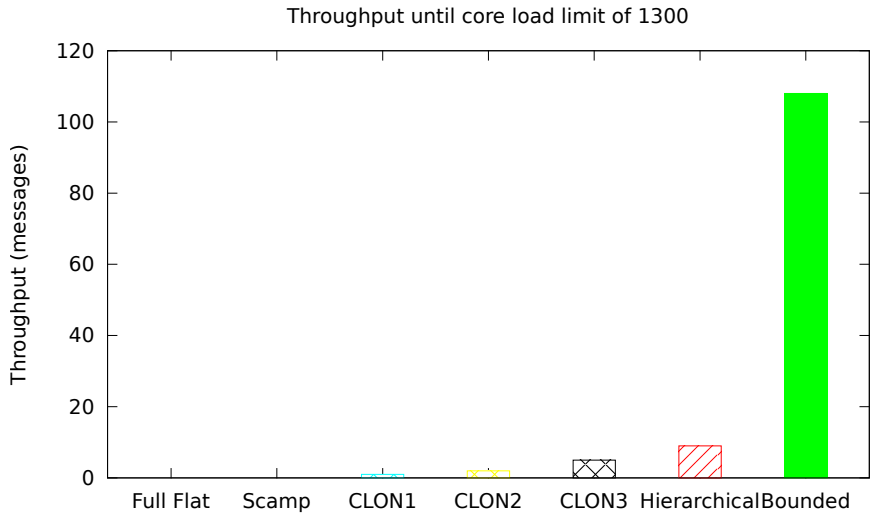


Figure 4.7: Throughput before core limit is reached

It is visible that Bounded Gossip offers the best throughput when we limit the total core load to $1,300$ messages, by a factor greater than 10 over the second best protocol. As expected, considering the results presented above, the topology-oblivious solutions, Flat Gossip and Scamp, cannot effectively disseminate any messages to all participants with such a small core load. Although the first CLON configuration tries to send more core messages for a single application-level message than the ones allowed in total, the $1,300$ that were not dropped allowed that message to reach every node in the network. The second and third CLON configurations achieved slightly better results, due to the core round limits and the fewer core links, respectively. The poor resource utilization of Hierarchical Gossip also induces a high core load even when such messages are not needed, drastically reducing the throughput that the solution can achieve when

the core switch load is limited in a given amount of time.

4.8 Reliability

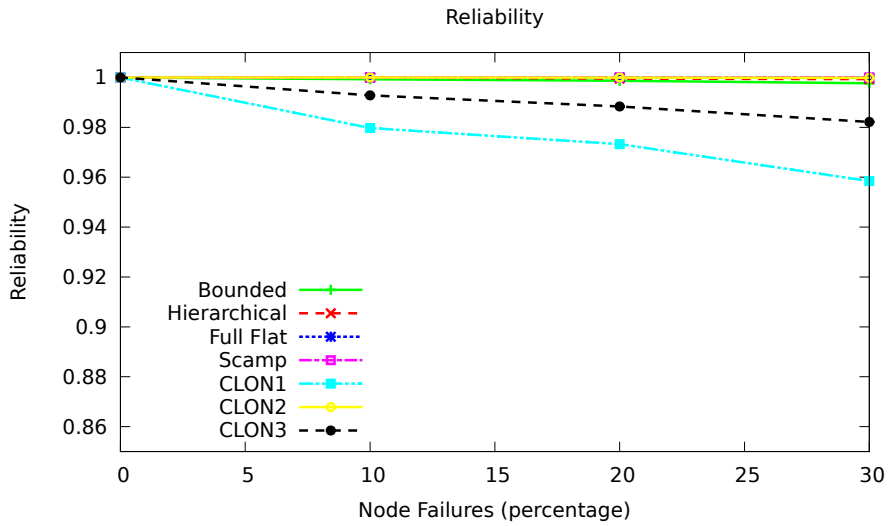


Figure 4.8: Reliability results

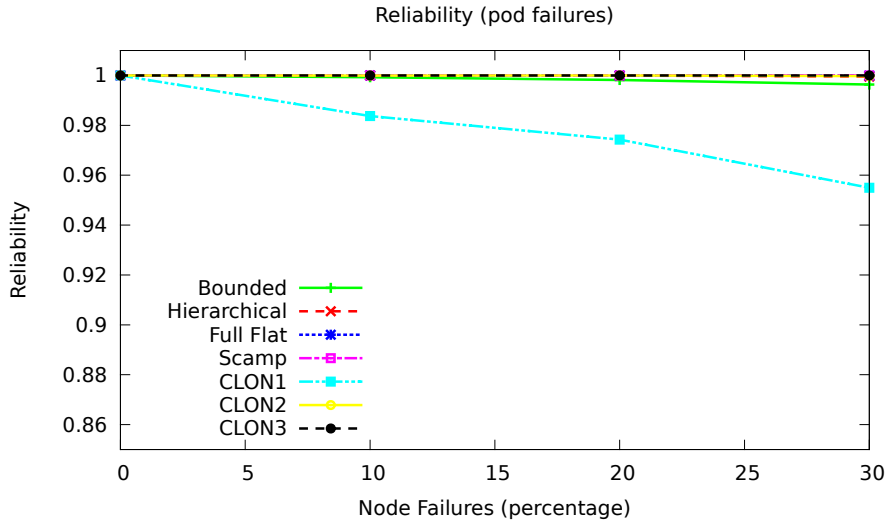


Figure 4.9: Reliability results (pod failures)

In the reliability experiments, we simulated three different scenarios where nodes fail concurrently with the generation of new messages and their dissemination over the system, with the goal of evaluating the robustness of Bounded Gossip as well as to compare it with the robustness of other competing protocols.

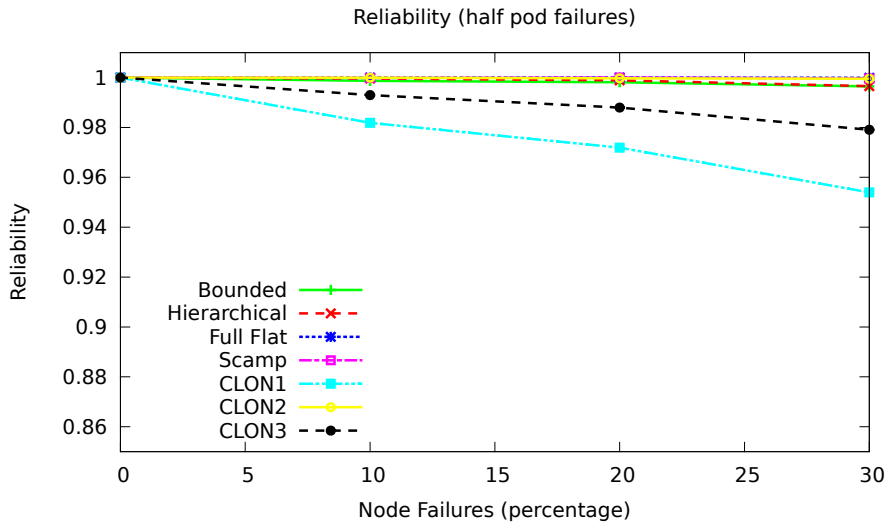


Figure 4.10: Reliability results (half pod failures)

In the first scenario, we continuously generate messages up to a total of 5,600 messages, failing a node uniformly at random in each gossip round. We executed various simulations with different percentages of nodes failing, up to 30% of all nodes in the system. Failed nodes did not join the system again. In the end of the simulation, we counted the number of active nodes that received each message and divided it by the total of active nodes still in the system. We then averaged that number to find the reliability of the protocol in that experiment. To exclude messages that had little chance of being disseminated, we only count messages that are known by at least one active node. Finally, we plotted the results which are provided in Figure 4.8.

All the protocols were able to achieve similar results in these conditions, although two of the three CLON configurations had weaker results (but still with reliability values of at least 96%). The other protocols are able to achieve reliabilities close to 100% even when 30% of all nodes fail. This demonstrates the inherent robustness of epidemic protocols, and one of the reasons their redundancy properties are important to keep in a system that requires reliability. Another important aspect to consider is that our solution is able to maintain those properties despite the membership and dissemination bias employed, that allowed for a better resource utilization and less core and aggregation switch load as seen in previous experiments.

To assess if the above is true when other failure models are considered, we simulated similar settings for failure of 30% nodes not chosen at random. Due to the hierarchical structure of datacenters, it is feasible to consider a scenario where a switch malfunction causes its closest

nodes to stop receiving messages. Therefore, we devised simulations where all the nodes in a given cluster became disconnected. Since in our topology there are 32 nodes in each cluster, to maintain the same average rate of failure than our previous tests we fail an entire cluster every 32 gossip rounds. The results are visible in Figure 4.9.

Although some of the protocols are negatively affected by the non-uniform failures and present a decrease in reliability, other protocols such as the Hierarchical Gossip protocol and the third CLON configuration have better reliability results under these circumstances. After a careful analysis of the results, we came to the conclusion that such behaviors can be explained by the inherent difficulty of those protocols in infecting all the clusters in the system with each message. Because we are just counting the number of active nodes that were infected with the message, we are reducing the number of clusters that need to be infected and thus reducing the difficulty in infecting all the clusters.

To consider another analysis on non-uniform failures that difficult the infection of all the clusters in the system, we simulated a different scenario where we still failed 30% of all the nodes but where only half of the nodes in a cluster failed. In this case, for a message to have 100% reliability it needs to infect the remaining half of every cluster where nodes failed. We present the results in Figure 4.10. As expected, all the protocols are negatively affected in this scenario, and present slightly lower reliability results than in the uniform failures simulation. Still, Bounded Gossip maintains its strong reliability guarantees, despite the controlled form of determinism present in the solution. Also, because of the parameters used by Bounded Gossip, it would be easy to overcome scenarios with such a type of failures by increasing the round limit in the aggregation-level dissemination (π_1). Similarly, to compensate for a scenario where entire core zones are expected to fail, the system administrator can configure a higher round limit for core-level dissemination (π_2).

Finally, we also extracted results from the preliminary version of the protocol, described in the previous chapter. The reliability values obtained by it can be seen in comparison to the final solution in Figure 4.11. It is clear that our changes in the membership and dissemination schemes have resulted in a more robust system, that handles node failures with exactly the same behavior as the more robust epidemic protocols. Our previous version, while obtaining good results, presents a lower percentage of messages that reach 100% of active nodes than competing protocols. As we have seen before, these changes in reliability do not penalize the number of

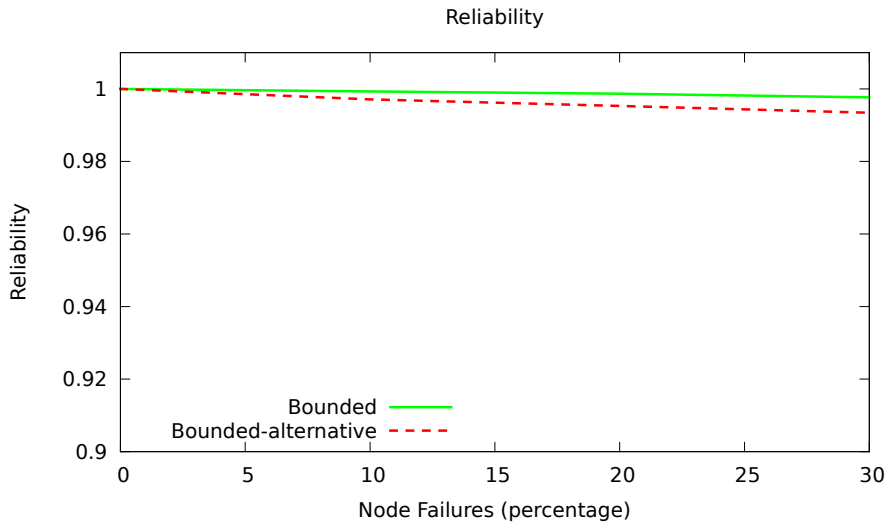


Figure 4.11: Reliability results (Bounded Gossip alternatives)

core membership contacts, instead achieving a lower bound of core membership load.

4.9 Load Distribution per Node

One important feature of epidemic protocols is their ability to distribute the load across all participants in the system. This means that every node makes on average the same amount of effort during the dissemination process, by processing and sending the same number of messages as its peers. To assess if our solution maintains this desirable property of gossip protocols, we measured the number of messages that each node sent during the dissemination process. We divided that number by the total of application-level messages generated in that time, to get the average number of messages sent by each node per application-level message. The values can be seen in Figure 4.12.

Instead of a fully uniform load distribution, we induced a load distribution divided in two groups, where one group sent an average of 13 messages per application-level message (the value configured for our fanout parameter) and the other group sent an average of 7 messages per application-level message.

To understand the disparity of the values and the creation of both groups, we measured and grouped the load distribution per node according to each node's role in the dissemination scheme. We obtained the results present in Figure 4.13.

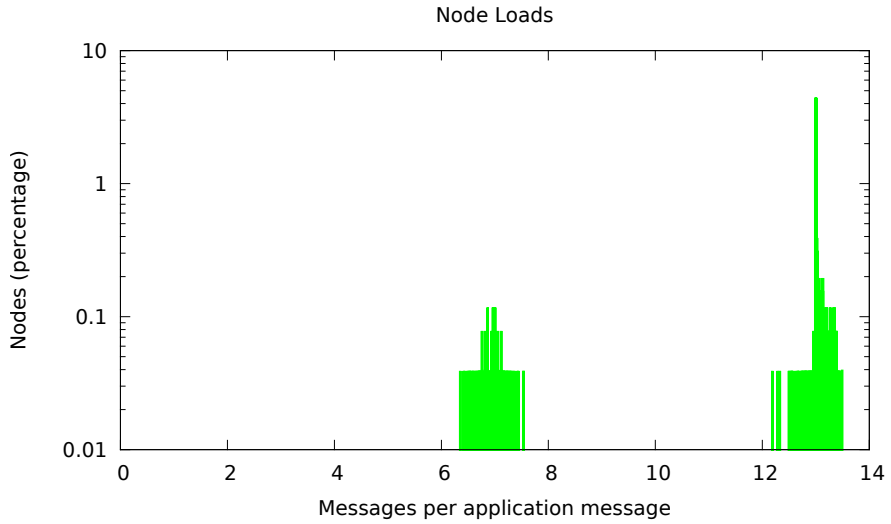


Figure 4.12: Load distribution per node

It is now understandable that the difference in load is directly correlated to the role of each node in the dissemination process. In this execution, we configured a value of $\pi_2 = 2$ and a value of $\pi_1 = 4$. Due to those values and the topology hierarchy, it is very likely that aggregation links receive messages in the middle stages of the aggregation dissemination. That means nodes with that role will synchronize between the replicas so that only one of them processes the message. On the other hand, core role nodes are more likely to receive messages during the last transmission of the core dissemination stage. Those nodes will not synchronize the processing of such messages because they do not need to be transmitted through core links. All the replicas will therefore process the message and distribute it merely at a local level, inside the cluster.

To compensate for this fact and show the capabilities of achieving a uniform load distribution, we reconfigured the protocol to have a closer number of rounds for each dissemination stage. Naturally, we compensated the lower number of aggregation dissemination rounds with a higher value for the aggregation partial view (PV_1) of each node. The results are shown in Figure 4.9.

While not exactly the same, the loads per node are much more similar. An alternative behavior that would not produce a disparity in load distribution per node would be relying on a different synchronization process in which role replicas, instead of removing the identified message from their queues, reconfigured their copy to be disseminated only at the local level. This would improve the protocol's resilience but could introduce some latency degradation in

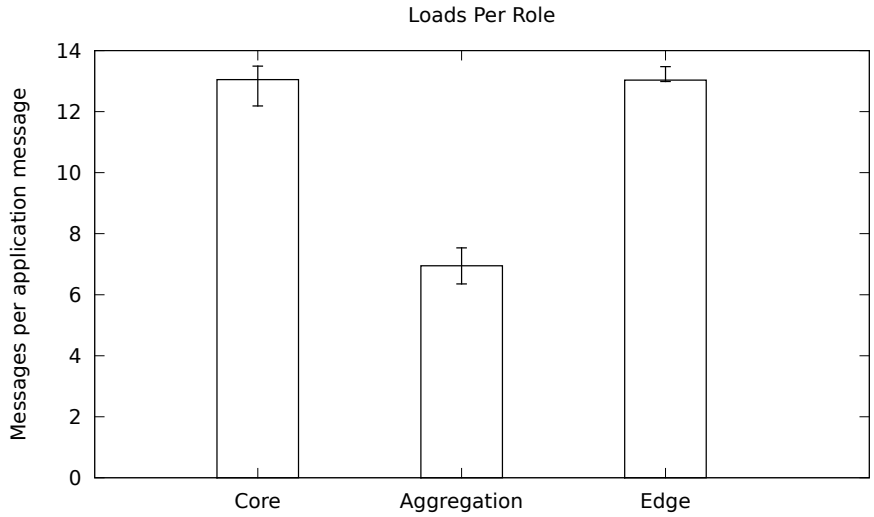
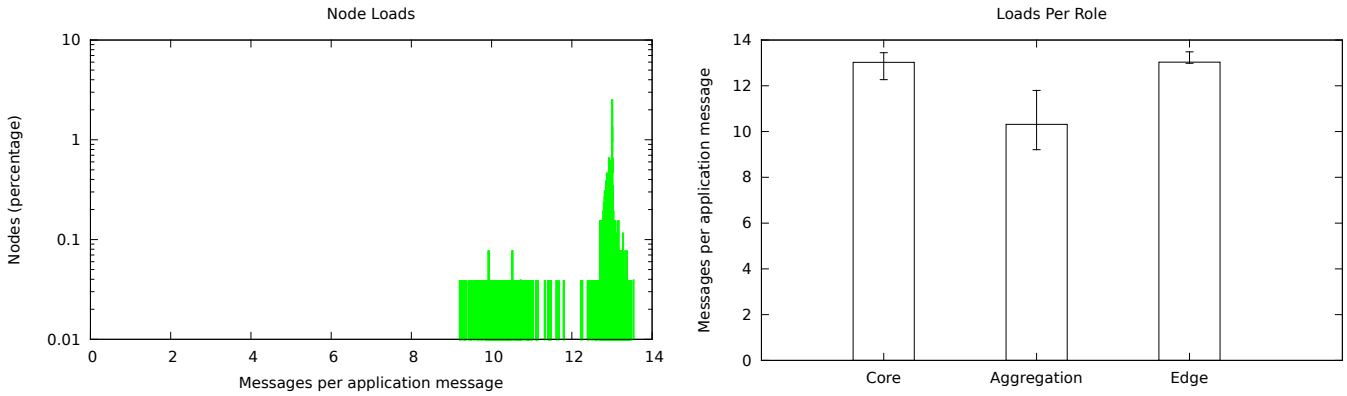


Figure 4.13: Load distribution per node, according to role



(a) Per node

(b) According to role

Figure 4.14: Load distribution

cases where the involved nodes would exaust quota on transmitting such messages and keep the other messages (that required dissemination at the level for which the node is responsible) in their queues for longer.

Summary

This chapter evaluated our solution against other competing protocols, such as CLON and Hierarchical Gossip, while also showing the advantages of topology-aware protocols over topology-oblivious ones. We showed how Bounded Gossip minimizes load in core routing equip-

ment in a three-tier architecture, achieving a better resource utilization that offers a throughput 10 times better than previous solutions in scenarios where core load is limited. We compared the various protocols' latency and reliability and proved that Bounded Gossip does not suffer from penalties in those values. Finally we presented the load distribution per node in the system and offered possible ways to make it more uniform.

The next chapter concludes the thesis and offers some points for future research.

5 Conclusions

5.1 Conclusions

This thesis proposed Bounded Gossip, a gossip protocol for large-scale data centers. We introduced the most important concepts of epidemic protocols and their operation in datacenters, defining three different approaches for imbuing topology-awareness in such solutions. We showed the benefits of adding determinism to epidemic broadcast, creating a protocol that relies on three topology-aware components: a membership service, a dissemination scheme and a rate-based flow control mechanism. We evaluated the performance of our solution against previous works found in the literature and achieved better resource utilization that translates in 10 times message throughput with less switch load per round in the higher levels of the hierarchy and no penalty to overall dissemination latency or reliability. We also presented an analysis of the load induced in the nodes in the system and compared some results to a previous version of our work.

5.2 Future Work

As future work, we plan on extending our solution to operate efficiently across multiple datacenters, leveraging the architecture described in this thesis that offers support for an arbitrary number of hierarchy levels. Additionally, it would be important to adapt and evaluate Bounded Gossip in more recent datacenter architectures, proposed to improve the conditions set by the current three tier architecture. These proposals include the use of modular switches and redundant links between servers (Vahdat, Al-Fares, Farrington, Mysore, Porter, & Radhakrishnan 2010) or more significant changes such as deploying the servers in a way that physically translates a structured network (Costa, Donnelly, O'Shea, & Rowstron 2010; Abu-Libdeh, Costa, Rowstron, O'Shea, & Donnelly 2010).

References

Abu-Libdeh, H., P. Costa, A. Rowstron, G. O'Shea, & A. Donnelly (2010, August). Symbiotic routing in future data centers. *SIGCOMM Comput. Commun. Rev. 41*, 51–62.

Agrawal, D., A. El Abbadi, & R. C. Steinke (1997). Epidemic algorithms in replicated databases (extended abstract). In *Proc. of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '97, New York, NY, USA, pp. 161–172. ACM.

Benson, T., A. Akella, & D. A. Maltz (2010). Network traffic characteristics of data centers in the wild. In *IMC 2010*, pp. 267–280.

Birman, K. P., M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, & Y. Minsky (1999, May). Bimodal multicast. *ACM TOCS 17*, 41–88.

Branco, M., J. Leitão, & L. Rodrigues (2012). PEC: Protocolo epidémico para centros de dados. In *INForum - Simpósio de Informática*, Portugal.

Carvalho, N., J. Pereira, R. Oliveira, & L. Rodrigues (2007). Emergent structure in unstructured epidemic multicast. In *Proc. of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '07, Edinburgh, pp. 481–490. IEEE Computer Society.

Costa, P., A. Donnelly, G. O'Shea, & A. Rowstron (2010). CamCube: A key-based data center. *Technical Report MSR TR-2010-74*.

DeCandia, G., D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, & W. Vogels (2007, October). Dynamo: Amazon's highly available key-value store. *SIGOPS OSR 41*, 205–220.

Demers, A., D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, & D. Terry (1987). Epidemic algorithms for replicated database maintenance. In *Proc. of the sixth annual ACM Symposium on Principles of distributed computing*, PODC '87, New York, NY, USA, pp. 1–12. ACM.

Eugster, P., R. Guerraoui, A.-M. Kermarrec, & L. Massoulie (2004, May). From epidemics to distributed computing. *IEEE Computer 37*(5), 60 – 67.

Eugster, P. T., R. Guerraoui, S. B. Handurukande, P. Kouznetsov, & A.-M. Kermarrec (2003, November). Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst. 21*, 341–374.

Eugster, P. T., R. Guerraoui, A.-M. Kermarrec, & L. Massoulié (2004, May). Epidemic information dissemination in distributed systems. *Computer 37*, 60–67.

Ganesh, A. J., A.-M. Kermarrec, & L. Massoulié (2002). HiScamp: self-organizing hierarchical membership protocol. In *ACM SIGOPS EW 2002*, Saint-Emilion, France, pp. 133–139.

Ganesh, A. J., A.-M. Kermarrec, & L. Massoulié (2001). SCAMP: Peer-to-peer lightweight membership service for large-scale group communication. In *Networked Group Communication Workshop (NGC)*, Volume 2233 of *Lecture Notes in Computer Science*, pp. 44–55. London, UK: Springer Verlag.

Gupta, I., A.-M. Kermarrec, & A. J. Ganesh (2006, July). Efficient and adaptive epidemic-style protocols for reliable and scalable multicast. *IEEE TPDS 17*(7), 593–605.

Gupta, I., R. v. Renesse, & K. P. Birman (2001). Scalable fault-tolerant aggregation in large process groups. In *Proc. of the 2001 International Conference on Dependable Systems and Networks (formerly: FTCS)*, DSN '01, Goteborg, Sweden, pp. 433–442. IEEE Computer Society.

Jelasity, M., R. Guerraoui, A.-M. Kermarrec, & M. v. Steen (2004). The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In *Proc. of the 5th ACM/IFIP/USENIX international conference on Middleware*, Middleware '04, New York, NY, USA, pp. 79–98. Springer Verlag.

Karp, R., C. Schindelhauer, S. Shenker, & B. Vocking (2000). Randomized rumor spreading. In *Proc. of the 41st Annual Symposium on Foundations of Computer Science*, Redondo Beach, CA, pp. 565–. IEEE Computer Society.

Kermarrec, A.-M. & M. v. Steen (2007, October). Gossiping in distributed systems. *SIGOPS Oper. Syst. Rev. 41*, 2–7.

Lakshman, A. & P. Malik (2010, April). Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev. 44*, 35–40.

Leitão, J., J. Marques, J. Pereira, & L. Rodrigues (2009). X-BOT: A protocol for resilient

optimization of unstructured overlays. In *Proc of the 2009 28th IEEE International Symposium on Reliable Distributed Systems*, Niagara Falls, NY, pp. 236–245. IEEE Computer Society.

Leitão, J., J. Pereira, & L. Rodrigues (2007a). Epidemic broadcast trees. In *Proc. of the 26th IEEE International Symposium on Reliable Distributed Systems*, SRDS '07, Beijing, pp. 301–310. IEEE Computer Society.

Leitão, J., J. Pereira, & L. Rodrigues (2007b). HyParView: A membership protocol for reliable gossip-based broadcast. In *Proc. of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '07, Edinburgh, pp. 419–429. IEEE Computer Society.

Lin, M.-J. & K. Marzullo (1999). Directional gossip: Gossip in a wide area network. In *Proc. of the Third European Dependable Computing Conference on Dependable Computing*, EDCC-3, London, UK, pp. 364–379. Springer Verlag.

Matos, M., A. Sousa, J. Pereira, R. Oliveira, E. Deliot, & P. Murray (2009). CLON: Overlay networks and gossip protocols for cloud environments. In *On the Move to Meaningful Internet Systems, International Symposium on Distributed Objects, Middleware, and Applications (DOA)*, Volume 5870 of *Lecture Notes in Computer Science*, pp. 549–566. Springer Verlag.

Montresor, A. & M. Jelasity. PeerSim: A scalable P2P simulator. In *P2P 2009*, Seattle, WA, pp. 99–100.

Renesse, R. v., K. P. Birman, & W. Vogels (2003, May). Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM TOCS 21*, 164–206.

Renesse, R. v., D. Dumitriu, V. Gough, & C. Thomas (2008). Efficient reconciliation and flow control for anti-entropy protocols. In *Proc. of the 2nd Workshop on Large-Scale Distributed Systems and Middleware*, LADIS '08, New York, NY, USA, pp. 6:1–6:7. ACM.

Renesse, R. v., Y. Minsky, & M. Hayden (1998). A gossip-style failure detection service. Technical report, Cornell U.

Rhea, S., D. Geels, T. Roscoe, & J. Kubiatowicz (2004). Handling churn in a DHT. In *Proc. of the annual conference on USENIX Annual Technical Conference*, ATEC '04, Berkeley,

CA, USA, pp. 10–10. USENIX Association.

Tang, C. & C. Ward (2005). GoCast: Gossip-enhanced overlay multicast for fast and dependable group communication. In *Proc. of the 2005 International Conference on Dependable Systems and Networks*, DSN '05, Yokohama, Japan, pp. 140–149. IEEE Computer Society.

Vahdat, A., M. Al-Fares, N. Farrington, R. N. Mysore, G. Porter, & S. Radhakrishnan (2010, July). Scale-out networking in the data center. *IEEE Micro 30*, 29–41.

Voulgaris, S., D. Gavidia, & M. v. Steen (2005). CYCLON: Inexpensive membership management for unstructured P2P overlays. *Journal of Net. and Syst. Man. 13*, 2005.

Voulgaris, S., E. Rivière, A.-M. Kermarrec, & M. Van Steen (2005). Sub-2-Sub: Self-Organizing Content-Based Publish and Subscribe for Dynamic and Large Scale Collaborative Networks. Rapport de recherche RR-5772, INRIA.