Universidade do Minho
Escola de Engenharia

Miguel Ângelo Marques de Matos

Network-Aware Epidemic Broadcast

Tese de Mestrado
Mestrado em Engenharia Informática
Trabalho efectuado sob a orientação do
Professor Doutor Rui Carlos Oliveira

Maio 2009

Acknowledgements

To my advisor Prof. Rui Oliveira for his unwavering support since I joined the distributed systems laboratory and for the enlightened guidance. To Prof. António Sousa for our weekly meetings on the context of the DC2MS project whose insights proved so fruitful. To Prof. José Pereira for all the insightful brainstorms we had that provided invaluable research directions in all the moments. Without the kind support of the three this dissertation will be impossible.

To my colleagues and friends at the laboratory, Nuno Carvalho and Ricardo Vilaça, for their support, critics and ultimately for the affable working environment.

To Ana, for everything only we know.

To my parents and nephew, for allowing me to stand in the shoulders of giants.

And finally to the Great Unknown for providing so many mysteries to keep us thinking.

Resumo

Os protocolos de disseminação fiável baseados na abordagem epidémica têm ganho popularidade nos últimos anos dada a sua escalabilidade e resiliência na entrega de mensagens em sistemas distribuídos de larga escala. Contudo, esta resiliência e escalabilidade são obtidas através de elevados níveis de redundância na propagação das mensagens que conduzem inevitavelmente ao consumo substancial de recursos nos nodos e respectivos canais de comunicação. Em cenários que apresentam canais com recursos restritos, como o modelo emergente da Computação em Nuvem em que vários *data centers* estão interligados numa federação global, esta característica impede a utilização efectiva desta classe de protocolos.

O objectivo desta tese é, portanto, aumentar a aplicabilidade dos protocolos de disseminação epidémicos, através da redução da carga imposta nos canais com recursos restritos. Isto é alcançado construindo uma rede sobreposta que tem em conta as características individuais dos canais de comunicação, e através de um protocolo de disseminação que considera a localidade dos nodos aquando da propagação das mensagens. Através de experimentação exaustiva, observa-se que os protocolos propostos reduzem a carga imposta nos canais de comunicação com recursos restritos, sem contudo afectar a escalabilidade e resiliência que tornam os protocolos de disseminação epidémica tão atractivos.

Abstract

Epidemic multicast is an emerging resilient and scalable approach to the reliable dissemination of application data in the context of very large scale distributed systems. Unfortunately, the resilience and scalability come at the cost of considerable redundancy which led to high resource consumption on both links and nodes. In environments with resource constrained links, such as in Cloud Computing infrastructure composed by data centers organized in a federation around the globe, the high resource consumption precludes the use of this class of protocols. The goal of this dissertation is therefore to cope with the constraints of these scenarios, by reducing the network load imposed on the constrained long distance links. This is achieved by constructing an overlay that reflects the characteristics of the links, and by using a dissemination protocol that takes into account locality when transmitting the message payloads. We conducted an extensive experimental evaluation that presents an improvement over an order of magnitude in the number of messages that traverse the costlier links, without endangering the resilience and scalability properties that make epidemic based protocols so attractive.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Begin - to begin is half the work, let half still remain; again begin this, and thou wilt have finished.

Decimus Magnus Ausonius

This introductory chapter presents the motivation that led to this dissertation, its relevance to the problems faced in the actual IT scenario, the main results obtained and an outline of the chapters of the dissertation.

1.1 Motivation

With the popularization of the modern desktop computer, and its ever growing processing and storage capabilities, we have been assisting through the last two decades to a massive decentralization of computing power. The common user can now do most of its everyday tasks from spreadsheets to text processing with the computer in its desk, without needing to login in the old mainframe. On the other hand, the advent of the World Wide Web and its exponential growth in the end of the last century led to the publication of content and services through well known providers that rely on the client-server paradigm. The services and contents are hosted in the provider central servers and the client accesses them by means of its Internet connection.

The standardization and commoditization of hardware enabled the construction and assembly of infrastructures composed by thousands to hundreds individual computers that when internetworked form a platform more powerful than the simple sum of all parts, the well-known data centers. Todays data centers effectively support our Information Society ranging from government agencies that manage citizen information to private companies that provide a wide variety of services.

The necessity, and possibility, of building those large infrastructures incited practitioners to develop mechanisms that effectively harness the power they provide leading to the grid computing initiative. The rationale behind those grid infrastructures is to use a divide and conquer strategy in order to parallelize applications. With the massive parallelization, is then possible to solve technical or scientific problems that would otherwise not be computable in acceptable time frames. The nodes composing the grid act in a concerted manner by splitting the task in several segments and working over each one of them individually. The typical usage of the grid includes the processing of long running batch tasks controlled by one entity, by dedicating parts of the grid infrastructure to that particular computation. Examples of such applications include statistical analysis and inference over large amounts of data or processing intensive tasks, such as weather forecasting or protein folding. The individual nodes in the grid are loosely coupled entities.

In the grid, the characteristics of the jobs requires the pre-allocation of considerable parts of the infrastructure, which inhibits the use of completely automated resource allocation, such as that done in a single computer, and prevents tasks from different entities to run concurrently on the nodes of the infrastructure. In the latter 90's we assisted to the evolution of this concept with the introduction of Utility Computing. In Utility Computing the focus is on the business model that by means of metering and billing allows customers to access the computational resources of the provider. The term utility comes from the idea that computational resources should be offered as a public service, like electricity, and therefore billed according to consumption. This business model allows customers to access vast quantities of resources for the amount of time desired, without having to setup up a complex IT infrastructure with the implicit cost it carries and benefits the providers as it allows them to rent the excess capacity of their infrastructure, which is frequently over provisioned in order to meet peak demands.

In parallel to this business model, we have been assisting to the emergence of the Software as a Service paradigm. In this business model, applications are delivered through the Internet as a service to its customers. The administrative burden of managing the low level infrastructure, deploying and updating the system's hardware and software is offloaded to the service provider, allowing the customers to focus on the details of their particular businesses. Recently this Software as a Service model has been expanded to offer programming platforms and low level IT infrastructures within the same business model in what is now known under the Cloud Computing moniker. While at first sight it may look similar to Utility Computing, there are some key differences that justify a new designation. The first and perhaps more important differentiator is the concept of elasticity. Elasticity allows the automatic up- and down-scaling of allocated resources in a transparent way, and guarantees that failures are concealed from the customer by quickly replacing the failed node with a spare replica, a property known as self-healing. Whereas in Utility Computing the customer rents a 'grid' for its own use and discards it when the work is done, Cloud Computing has a broader scope. The goal is to completely offload the infrastructure of a given customer to the cloud provider, leveraging on its expertise in managing those large infrastructures and relying on well defined Service Level Agreements that guarantee the reliability of the service and data confidentiality. The services provided by a cloud environment could range from the low level infrastructure where the customer only 'sees' bare-metal machines, to the offering of a programming platform where the customer is able to deploy its application in the cloud and do not worry about the low level management details, up to the already known software as a service model. Examples of providers offering solutions at those different levels include, respectively, Amazon EC2 [4], Google App Engine [18], and Salesforce [39].

The availability of these platforms is inducing a shift from the completely decentralized philosophy of nowadays to a centralization of computational capabilities by a couple of service providers. History repeats itself and it seems that the pendulum is swinging back to the centralization of application platforms, [13].

To enable the worldwide delivery of those services over the network and guarantee that they can still be provided despite natural disasters such as earthquakes, floods and civil turmoils, and because of problems of scale itself, the current infrastructure consists of geographically dispersed data centers aggregated by means of federation. To this end the different data centers are connected among them by means of expensive, possibly inter-continental and hopefully redundant links in order to mitigate the problems pointed above. Furthermore the intra-data centers links are expected to be more reliable than the inter-data center ones, with high bandwidth and low latency, and be pervasively deployed in the data center infrastructure, in order to support the communication needs of the hundreds to thousand individual nodes that power the data center.

Despite the different offerings and the inner details that power each one of them, those cloud platforms are distributed systems which happen to be composed of hundreds to thousands of nodes. This underlying infrastructure could be seen from the customer point of view as a nearly infinite pool of computing resources available on demand. On the other hand, from the service provider's point of view, it is of paramount importance to effectively manage those nodes in order to enable efficient resource usage, provide accounting mechanisms that can be used to bill the customer and increase the reliability of the system as a whole in order to meet stringent Service Level Agreements. As in any distributed system, there are two fundamental building blocks that leverage the reliability and proper coordination of the system and enable its proper management: reliable multicast and distributed agreement. Reliable multicast provides trustworthy communication primitives to the system, guarantying that messages reach their intended recipients. Distributed agreement offers an abstraction to the voting problem, ensuring that all correct processes (eventually) decide the same value upon a set of valid proposals. With these strong primitives it is possible to build a reliable management framework to properly administer the infrastructure. For instance, the administrator could provide data aggregation services to account for the global state of the system, and deploy the billing mechanisms on top of it. The problems raised by the management of such very large scale infrastructures and the need to provide reliable mechanisms in order to do so are the core of an undergoing project at our lab, Dependable Cloud Computing Management Services [2]. This thesis pretends to give a satisfactory answer to one of those problems: reliable multicast in the context of the very large distributed systems that power todays' cloud infrastructures.

1.2 Brief Problem Presentation

As outlined in the previous Section, todays' cloud infrastructures consist of geographical dispersed data centers, organized in a federated fashion and connected by long distance expensive links. In order to provide a reliable management service that spawns this federated organization, a scalable and reliable communication service is fundamental. Unfortunately, the communication demands intra-data center and inter-data center are very different, both in terms of latency and bandwidth required to provide a reliable service, and in the need of timeliness of information available across the federated infrastructure. In a smaller scope, this can be also observed in the architecture of a single data center, as collections of nodes are also grouped in a federated manner that reflects the networking technology available today. This is evident in the individual clusters that compose the data center and are deployed in a hierarchical fashion, inter-connected by more expensive network devices as we move up in the networking tree that composes the data center. This problem is alleviated, but not solved, by using a fat tree network layout [3], where leaf nodes are grouped in a way to mitigate the load imposed on the individual network devices such as routers and switches that interconnect them, while at the same time providing transparent load balancing and failover among those devices. Further details of the fat tree network deployment model in a data center can be found in [3].

The objective of this thesis is therefore to provide a reliable communication service that improves the matching between the amount of traffic handled by each component on the infrastructure and the running application semantics and needs. However, this is not straightforward in a very large environment composed of thousands to hundreds of thousands of nodes, each one with a more or less unpredictable life cycle. The life cycle of each node is very important in a infrastructure of such scale, as nodes may arbitrarily join and leave the network, either due to failures of both links and nodes or due to business needs related, for example, to maintenance and updates of the individual components. As such, the proposed communication service provides reliable dissemination mechanisms, to all nodes in the infrastructure while at the same time seamlessly coping with the inherent churn rates, the rate at which nodes leave and enter the system.

This is achieved by leveraging on the resilience of unstructured network overlays, more details will follow in the subsequent chapters, and carefully biasing the overlay links in order to take into account the underlying network topology.

With this approach, we are able to reduce the load imposed on the long distance links, for instance those connecting geographically dispersed data centers, by an order of magnitude while at the same time tolerating considerable failures of the whole infrastructure. Furthermore, our reliable multicast service constantly adapts to changes on the infrastructure that happen, for instance, when a considerable amount of nodes join or leave the system.

1.3 Dissertation Outline

The rest of this dissertation is organized as follow: Chapter 2 provides background information in order to familiarize the reader with the key concepts of reliable dissemination in very large scale distributed systems, and offer a state of the art review of the current approaches to address this problem; Chapter 3 presents the problem we are addressing, its worthiness and why the current approaches do not provide satisfactory answers; Chapter 4 presents the rational behind the developed protocol, carefully describing it and why it addresses the problem presented in Chapter 3; Chapter 5 assesses the quality of the proposed service by means of extensive simulations and the discussion of the obtained results; and finally Chapter 6 concludes the dissertation, presenting the main insights obtained during its elaboration, how successfully it achieves the proposed objectives and presenting directions for future research on the subject of reliable dissemination services on very large scale distributed systems.

Chapter 2

Related Work

Great work is done by people who are not afraid to be great.

Fernando Flores

This chapter is divided in two main sections: Section 2.1 introduces the two main approaches to address the problem of building the supporting infrastructure to reliable multicast, and Section 2.2 presents the state of the art in unstructured network protocols.

2.1 Background

This section, introduces the background concepts necessary to familiarize the reader with the technical details present in the rest of the dissertation. A brief review of the core concepts of structured overlay networks is presented in Subsection 2.1.1, and a thoughtfully analysis of unstructured overlay networks principles is presented in Subsection 2.1.2, as the latter will be the approach taken in this thesis.

Before dwelling in the details of each approach, it is important to grasp a pervasive concept used by both proposals: the overlay. The overlay is a virtual computer network built atop another network, for instance a physical one. The overlay could be visualised as a graph, where nodes, or peers, are connected by a virtual or logical links in order to form a path. Each node communicates with the others using those links, which are mapped to the underlying network as appropriate.

2.1.1 Structured Overlay Networks

The term structured overlay network comes directly from the fact that in this class of protocols the overlay is judiciously controlled, and information is placed on specific peers according to the rules defined in the particular algorithm used. Due to this very fact, structured overlay networks are extremely efficient in routing requests to the appropriate node, as the location of those nodes could be calculated in a deterministic fashion. Therefore, this class of protocols is extremely popular to store and retrieve arbitrary data and build distributed hash tables (DHT). The DHT algorithms define a topology by assigning identifiers to each node, and a function that determines the distance, in number of hops, between any two identifiers in the space. Nonetheless, the inherent overlay structure can also be used to provide reliable multicast primitives to applications [8,19,35,44].

The mechanisms to construct the structured overlay networks are roughly divided in two main classes [9]: hypercube algorithms and Cartesian hyperspaces.

In the hypercube mechanism, the space of identifiers, the keys, is populated by peers in a circular fashion, in a ring-like formation. Each peer is assigned a unique identifier, the *nodeId*, chosen at random. The *nodeId* is used to assign the node to a deterministic position on the ring, by means of a uniform hashing function. Therefore, nodes are distributed nearly evenly on the ring, thus achieving uniform data partitioning among the nodes in the overlay. With this structure established, each peer maintains a routing table to its neighbours in the key space, and is responsible for maintaining part of the key space between it and its predecessor and successor, the node(s) immediately before and after it in the ring, respectively. Upon a request, the peer either responds to the client, if it is the manager of that key, or forwards the request to a neighbour that is numerically closer in the key space to the requested key, by consulting its routing table. The number of hops that a request must take in the ring before being successfully answered depends, therefore, on the number of entries in the routing table. With bigger routing tables, the request could be answered in less hops but, larger tables are costlier to maintain as the state of more peers needs to be taken into account. Upon failure of a node in the ring, its closest neighbours perform some calculations, dependent on the particular algorithm used, and the peer numerically close to the failed one takes over its key space. Examples of such protocols include [38,41,43].

On the other hand, in Cartesian hyperspace routing mechanism, nodes are organized in a d-dimensional cube. Each node in the system is assigned to a hyper-space region, and is responsible for managing the keys in that region. When a request from a client is received, the node responds to the client if it is responsible for the region of the request's key, or forwards the request in a greedy fashion to a neighbour whose region is closer to the request's key. As there are multiple paths between any two points in the space, the algorithm is capable of routing around failed regions in a straightforward fashion. Upon join, the new peer contacts a random node in the hyperspace, the key space is split in two halves, one of those parts is assigned to the new peer and the appropriate routing information is updated. The joining process could be optimized by splitting the key space in a more pondered manner, such as forwarding the joining node to a region whose key space is larger than the one initially chosen. Upon detection of a neighbour failure, nodes initiate a takeover procedure, to ensure that one of the neighbours becomes responsible for the region of the failed peer. After that, the neighbours send soft updates among them in order to update the respective routing tables, and ensure that the failed node is correctly pruned from the tables. The state necessary to maintain the routing information to the neighbours is of the $2d$ order, where d is the number of dimensions. The [34] structured overlay protocol is an example of Cartesian hyperspace routing.

A reliable multicast service could be deployed on those overlays by following two different approaches: flooding and tree-based dissemination.

As the name implies, in flooding [35] the application level messages received are relayed to all neighbours in the Cartesian hyperspace or in the hypercube. The flooding protocol leverages on the routing information already maintained by the overlay, and creates separate multicast groups on top of it, according to the interest of the peers. As expected, flooding is very demanding in bandwidth and as such, several optimizations to this naive strategy exist that take advantage of the location of nodes in the space in order to reduce the number of duplicates received by each node. In one of those strategies flooding is only done in the same 'direction' as the received message, as nodes on the opposite direction are already expected to have received the message [35].

In the tree-based approach [8], the dissemination of application level messages uses a reverse-forwarding mechanism to construct and maintain the multicast group that encompasses all the nodes interested in the dissemination process. For each multicast group, the dissemination protocol creates a multicast tree with a unique identifier, and uses it to relay messages to the relevant peers. To join the group, a peer uses the underlying overlay to send a message to the multicast group. As the joining request traverses the underlying overlay, each node checks whether it is already part of the desired multicast group, and if it is, it stops forwarding the message and adds the joining node as a child in the tree, if not the request is forwarded to the parent until it is adopted by a node or it reaches the root of the tree. In the latter case, the root will adopt the joining node as a direct children. The protocol carefully balances the dissemination tree in order to ensure an evenly load distribution among the participating nodes. To further prevent bottlenecks in certain nodes of the tree, the protocol provides mechanisms to demote a node's child to a grandchild, thus transferring some of the dissemination effort to its children.

Further details of the deployment of these protocols on top of the structured overlay construction mechanisms available, and a thoughtful comparison of the trade-offs between each one can be found in [9].

2.1.2 Unstructured Overlay Networks

A completely different approach from the one presented previously relies on the mathematical foundations of epidemic disease spreading [5]. Due to this, this class of algorithms is also known as epidemic-based reliable multicast and even gossip-based due to the similarities to rumor spreading. The underlying principle is astonishingly simple: if each member of the population infects a minimum number of neighbours drawn randomly across the universe, then the entire population will be infected after a known period of time, or rounds. The probability that the whole population becomes infected, or atomic infection, is therefore affected by two model parameters: the number of neighbours that each infected node tries to contaminate in each round, also known as the fanout, and the duration of the infection spreading, or number of rounds, modeled as discrete steps. Furthermore, two opposite infectious behaviours could be considered: infect and die, where an infected node contaminates a *fanout* number of neighbours and stops permanently, and the infect forever alternative, where an infected node will always try to infect *fanout* neighbours during the entire time span of the epidemic.

For a given population, the model parameters can be adjusted to ensure that all members are infected with high probability. In fact, slightly below those values the infection will reach almost none of the population, and above them the infection will reach almost all members, a property known as bimodal dissemination guarantee that has been studied in [6]. Due to the probabilistic guarantee that is possible to offer, this protocols are also known as probabilistic dissemination protocols.

Applying these principles to the dissemination of information in a computer network is however not trivial, and raises several interesting challenges. An essential requirement for epidemic based algorithms to work is the knowledge of the whole population because the targets selected for infection are expected to be drawn randomly across the entire population. Furthermore, in [12] the authors have identified key challenges when deploying those algorithms: membership, network awareness, buffer management and message filtering. The membership is related to the necessity of knowing the whole population, as explained previously, how nodes get to know each other, and how many of them they need to know to achieve successful dissemination. The second challenge, network awareness is concerned with the problem of reflecting the network topology in the connections established between nodes. These two challenges will be the core of this dissertation and will be addressed with further detail in the next sections. The buffer management problem is concerned with the handling of multiple messages by the same process simultaneously. When dissemination of different messages occurs concurrently, processes may have to temporally store messages in order to do adequate processing and, possibly, forward it to other nodes for a given number of rounds, which implies that processes may have to hold messages for considerable amounts of time. Furthermore, processes need to known the history of messages in order to avoid delivering duplicates to the application. As memory is not an infinite resource, these requirements and constraints demand that buffer management protocols ensure the timely pruning of spurious messages, without dropping unwanted ones, which could impact reliability [21]. Several solutions exist for this problem, such as dropping messages according to age [11], defining an obsolescence relation between messages [31] or calculating the overall average buffer capacity in a distributed fashion [37]. In reliable dissemination, the goal is to deliver every message injected into the system to every participant. Message filtering pushes this forward and attempts to reduce the number of uninteresting messages that a given process receives, by using the concept of interest groups, and ensuring the reliable dissemination only among the members of each group [10].

After the overview of the epidemic foundations presented above, we will now focus on the problems that arise from the construction and maintenance of the membership, and the properties that a protocol must abide by, to ensure reliable dissemination of information. As stated previously, for the epidemic model to work properly, the potential targets for infection should be chosen randomly across the universe of nodes. To be able to randomly choose across all the nodes, any given node must have, therefore, global knowledge. In fact, initial protocols such as [6], clearly rely on having global knowledge of the membership at each node to successfully guarantee the bimodal dissemination property. While this global knowledge could be attained for small to medium sized clusters with a relatively stable membership, it is not suitable, or even feasible, for large scale systems composed of hundreds to thousands of nodes. This comes directly from scale itself, as the knowledge necessary to maintain at each node requires vast amounts of memory [11] and from a natural phenomenon in distributed systems, churn. Churn is closely related to the dynamics of the environment, and measures the rate at which nodes enter and leave the system. If the churn rate is considerable, the cost of updating the global membership knowledge of all nodes in a large scale system, becomes unbearable or even unattainable.

To overcome this problem that effectively limits the applicability and scale of epidemic based solutions, researchers have developed several protocols that rely on epidemic mechanisms to build and maintain the membership information [11, 15, 16, 24–27, 42]. The rationale behind these algorithms relies on each process knowing only a small number of other processes, the view, instead of the global knowledge required before. The resulting 'who knows who' relationship could be modeled as a graph where the edges are the nodes, and the vertices represent the 'knows' relation, which can be symmetric in case of undirected graphs or asymmetric, if the graph is directed. In this representation the view corresponds then, to the set of graph neighbours of a given node. It has been proved [11] that constructing the right 'who knows whom' relationship with partial views of the system is a reliable approach to the construction of unstructured overlay networks without requiring global knowledge at each node.

When switching from global to partial knowledge, the uniformity of the random sampling process that chooses potential infection targets is affected, as nodes cannot select targets randomly across the universe but only in the restricted set of its neighbours [12]. The problem of choosing a random peer from the universe when global knowledge is not available or attainable, could be addressed by means of random walks [17]. A random walk is a procedure that consists of successively taking random paths while traversing a graph, for a given number of times. This has been deeply studied in [17], in the context of information searching and overlay construction mechanisms, and one of the important outcomes is that a random walk on a graph with 'enough' length is equivalent to choosing a node randomly across the universe of nodes, which effectively solves the problem of the random selections pointed previously.

Furthermore, as the systems evolves, namely with respect to its size, it is necessary to tune the dissemination parameters, the fanout and number of rounds, as well as the view size, a property which is known as adaptability. If a given protocol fails to constantly adapt to changing systems sizes the reliability and/or performance of the protocol will be seriously compromised. If the system size grows considerably, the failure to adapt the protocol parameters will inevitably lead to the loss of the reliability guarantees, as the overlay will partition and/or messages may not reach all the nodes.

On the other hand, if the system size shrinks below the pre-defined protocol parameters there will be an unnecessary waste of resources on nodes and links, as the protocol is configured to infect more nodes than the existing ones. The relationship of those parameters with reliability and the impact they have on each other has been studied in [6,23]. An important result of the previous works shows that for a given system size N, bimodal dissemination guarantees are obtained if each node infects around $log(N) + c$ nodes, where N is the system size and c a protocol parameter related to the desired reliability in the presence of faults. The final requirement to build a fully distributed membership service, is the bootstrapping itself, which consists on the initial steps that a process must effectuate in order to discover at least a node belonging to the overlay and establish a connection to it, a process that is known as joining. To the best of our knowledge, there are currently no solutions to address this problem in a fully decentralized fashion, as nodes joining the overlay are expected to know, a priori, a subset of 'well-known nodes' to which they can connect to.

The other fundamental aspect of building a fully scalable membership service is related to network awareness, or locality. In fact, if an epidemic protocol that does not take into account locality is deployed on a network where the cost of links may vary greatly, for instance a Wide Area Network, its reliability and usefulness is limited [12]. This comes directly from the fact that links are established with equal probability despite their cost, and therefore close neighbours may only be able to communicate among them by means of costlier, long distance links, for instance two nodes in the same LAN may not know each other and be able to communicate only by means of a common neighbour on the WAN. If the amount of messages exchanged between them is considerable, then the costlier links could easily become a bottleneck, for instance in terms of bandwidth or latency, precluding a reliable and scalable dissemination. It is important to note that the 'cost' function is abstracted out of the model and should only indicate an abstract distance between two nodes and/or the preference that should be given to a link over another. The traditional solution to the network awareness problem relies on hierarchical organizations: special processes establish links according to the cost function leading to an hierarchical or tree-like organization that reflects the network topology. Several well-known protocols, [10,25] use this principle to offer dissemination guarantees while mimicking the organization imposed by the cost function.

Despite the inherent details of each protocol the overlay a pervasive concept across all of them that abstracts the links established between any given pair of nodes. As the overlay could be seen as a graph, it is therefore of the utmost importance to understand the graph properties that are important to obtain a quality overlay upon which message dissemination could take place. Those properties have been identified in [20] and are the following: connectivity, average path length, degree distribution and clustering coefficient. Connectivity indicates whether there is at least one path from each node to every other node. Failure to maintain connectivity will result in partitions and therefore failure to infect all nodes. The average path length measures the number of hops that separate any two nodes in the graph, and is closely related to the overlay diameter. Low average path lengths are desirable as they represent a lower bound on the latency necessary to disseminate a given message, and thus tighten the vulnerability window to node and network faults. Degree distribution represents the probabilistic distribution of the neighbours of each node, its degree, and is related to node reachability and its proneness to disconnection from the rest of the overlay. Nodes with low degrees are prone to become disconnected in the presence of failures, whereas high degrees degrade the quality of the overlay as the dissemination effort becomes too dependent of those nodes. Therefore, a normal distribution with low deviation is essential to ensure a high quality overlay, and consequently, an effective and reliable dissemination. Clustering coefficient measures the closeness of neighbour relations, it is the ratio between the number of links established among the neighbours of a given node by the total of possible links among those neighbours. The numeric value of this property should be as small as possible because high clustering coefficients lead to an increased redundancy in message transmission, and the consequent waste of resources, and it also increases the probability of partitions as neighbour nodes tend to be highly connected around the cluster and poorly connected to the rest of the overlay.

So far we have analysed the requirements and theoretical properties necessary to obtain a fully decentralized and reliable membership service, known in the literature as the Peer Sampling Service [20]. This service offers abstract primitives to obtain a certain number of potential gossip targets. Although that service and the actual dissemination protocols are usually used together to provide a decentralized reliable multicast abstraction, we clearly separate them in this dissertation, as different requirements and assumptions are made on each one of them and, therefore, different improvements could be done on each one. We will now address the different dissemination strategies available, and the trade-offs offered by each one of them [22].

Gossiping strategies follow two major approaches: pushing and pulling. In a push strategy, each peer forwards a message as soon as received to its neighbours for a given number of rounds. If the payload is transmitted instantly them we are in the presence of the eager variant. If the payload is omitted and only an advertisement of the message is sent, then we are on the lazy variant. In the latter, a node that received the advertisement of a known message could then ask the source for the payload and lazily push the payload. Assuming that the message payload tends to be much larger than an advertisement with the message identifier, the lazy variant allows for a drastic reduction on bandwidth consumption at the cost of increased latency as three communication steps are needed to obtain the actual message content. In fact, if a pure lazy push strategy is used, it is possible to achieve exactly once payload delivery for every destination, at the cost of a considerable penalty in latency. Furthermore, the impact on reliability must also be taken into account, as the additional round trips widen the time window to network and node faults. Oppositely, in the eager variant the latency is minimal, but comes at the cost of higher bandwidth consumption, as nodes tend to receive multiple copies of the same message through different paths. The eager push strategy is the most common dissemination

```
proc send(destination,message)
```

Listing 2.1: Send primitive

strategy in nowadays protocols, such as [11, 21, 32], to cite a few.

In the pulling strategy nodes periodically ask neighbours for new messages. When a node receives a request for new messages, it will send all new known messages to the requester, if acting on the eager variant. Oppositely, in the lazy approach, also known as two-phase pull, the receiver of the request will send only a digest of the new known messages, allowing the requester to selectively pull the desired messages. As in the push approach, the lazy variant imposes less constraints on the bandwidth, while the eager variant decreases the latency necessary to disseminate the updates. However, as in pull gossiping updates are only asked periodically, the impact on latency of the lazy variant may be negligible if that period is considerable greater than three times the average network latency.

While the choice between an eager versus a lazy variant is clearly a trade-off between bandwidth and latency, the difference between a push versus a pull scheme is more subtle. In pull, nodes proactively ask for new messages where in push nodes behave in a reactive fashion to message exchanges. Therefore, in an environment where messages are sparingly injected into the system, a push strategy has no communication overhead, while the pull approach presents a constant noise due to the periodically check for new messages.

Before dwelling into the details of each protocol we will define the semantics used in the pseudo-code listings, which we will use for the rest of the document. The *send* primitive, depicted in Listing 2.1 is a low level operating system primitive that abstracts the transmission of a message on the underlying network. The first parameter, *destination*, identifies the receiver of the message, and the second the actual message to be sent.

The message is then handled on the receiver side by defining a procedure $handleMessageName$, where $MessageName$ is the message initially sent.

2.2 State of the Art of Unstructured Networks

This section carefully presents a review of the state of the art of membership management and dissemination protocols. The membership management protocols are divide according to their awareness, or not, to locality. The dissemination protocols subsection only describes one protocol. While several other well known protocols [6,11] may have been included they focus on aspects that are not central to this dissertation, such as buffer management and message filtering, and their underlying principles are based on the different dissemination strategies already presented in previously and as such they will not be covered. The described dissemination protocol uses different strategies to obtain a wide-range of latency versus bandwidth trade-offs. The reason to include just this dissemination protocol, comes from the fact that it will be latter improved in this dissertation in order to accomplish our goals.

2.2.1 Flat Protocols

This Subsection covers the state of the art in membership construction protocols that do not take into account locality, and therefore result in flat overlays.

Scamp

Scamp [15] is a peer-to-peer decentralized membership protocol with the interesting property that the average degree distribution converges automatically to the desirable value of $log(N) + c$, where N is the number of peers in the system, and c is a protocol parameter related to the amount of faults that can be reliably tolerated. The protocol is presented in pseudo-code in Listing 2.2.

Upon boot, lines 2 to 5, a node obtains a contact node by an external mechanism, adds it to its view and sends it a subscription request, enabling nodes to know about the joining node. Upon reception of a subscription request, the receiver forwards the request to all its neighbours and create additional c copies that will be sent to randomly chosen nodes in its view, as can be seen in lines 7 to 13.

The protocol foundation relies on a probabilistic function that integrates joining nodes into the view with a given probability that is inversely proportional to the view size. In short, the smaller the

```
1
2  upon init
3    contact = getContactNode()
4    view.Add(contact)
5    send(contact,Subscription(myId))
6
7  proc handleSubscription(nodeId)
8    for n ∈ view
9       send(n,Join(nodeId))
10
11   for i=0; i < c; i++
12      n = randomNode(view)
13      send(n,Join(nodeId))
14
15 proc handleJoin(nodeId)
16   keep = randomFloat(0,1)
17   keep = Math.Floor((viewSize + 1 ∗ keep)
18
19   if (keep == 0) and nodeId ∉ view
20    view.Add(nodeId)
21   else
22      n = randomNode(view)
23      send(n,Join(nodeId))
```

Listing 2.2: Scamp protocol

view size the greater the likelihood of a successful integration and vice-versa, as can be observed on lines 16 and 17. If the subscription is not accepted at a given node, then it is forwarded continuously to one of that node's neighbours, until it becomes eventually accepted, as is possible to observe in lines 19 and 20. This is important as it preserves the amount of subscriptions on the system and therefore ensures that a subscribing node is known by a minimum amount of nodes. It is also important to note that the views are asymmetric, which means that a node who knows another does not necessarily means that the latter knows the former. In a graph, this could be modeled as directed edges, whose origin is the node that knows the other and the end on the latter. By always forwarding subscriptions until they are accepted and emitting, on average, $viewSize + c$ subscriptions for each joining node, combined with the probabilistic integration function, Scamp ensures that the overlay average degree converges to the right value, providing adaptability to changing system sizes in a completely distributed fashion and without requiring global knowledge.

Scamp is a reactive protocol in the sense that it does not try to make further optimizations to the underlying overlay. In fact, in a stable environment the protocol does not induce any overhead on the network, as no messages need to be exchanged to preserve the overlay.

Cyclon

Cyclon [42] is a membership management protocols that uses a completely different approach from that found in Scamp. It relies on a shuffling mechanism where links are changed among the peers, to continuously improve the overlay and quickly detect and remove links pointing to nodes that leaved the overlay, either due to failures or to the normal life cycle. The shuffling operations is performed periodically by each node on the system and consists of several steps which we will describe below.

Each node periodically chooses a set of its neighbours of size c, which is the minimum of the known number of neighbours and C, a protocol parameter that specifies the maximum size of the shuffle set. From this set, a node X is randomly chosen to initiate a shuffle operation. The initiator sends the shuffle set to X, adding its own identifier to the set and removing X from it. Upon reception, X chooses a random set of its known neighbours with the same size of the received set and sends it to the initiator. After, both nodes integrate the nodes in the received set into its own view according to the following rules:

- Already known neighbours are discarded from the received set;
- If the integration of the received nodes into the view exceeds a given threshold, then already known nodes are discarded accordingly to the following rules:
  - Entries sent to the other node are discarded first;
  - If this is not enough the remaining neighbours are randomly discarded, until there is enough room to accommodate the received entries.

Cyclon improves over the original shuffling mechanism proposed in [40], by attributing an age notion to each link, and exchanging and discarding links accordingly to that metric from the oldest to the newer ones. With this improvement over the classical shuffling mechanism, Cyclon is able to quickly detect and remove links pointing to nodes that have leaved the system, promoting the healthy renewal of links according to its age.

HyParView

HyParView [24] also relies on a shuffling mechanism to manage the overlay. Its distinguishable characteristic is the maintenance of two different views with different goals and requirements: a larger passive view and a smaller active view. The active view is of size $fanout + 1$ and is used to disseminate application level messages, by flooding the graph defined by the relationships of that view and is maintained using a reactive strategy. When a node detects that a peer in its active view has left the overlay, due to a failure or a disconnect operation, it randomly chooses a peer in its passive view and adds it to the active view, therefore enabling a quick healing of the dissemination graph in presence of high rates of failures. The passive view is much larger than the active one and is used to find valid targets to heal the active view, as explained previously. The passive view is maintained by a shuffling mechanism similar to that of Cyclon, but instead of exchanging peers directly with its neighbours, the shuffle request is propagated through the overlay by means of a random walk, parametrized with a given time-to-live, a protocol parameter. By promoting shuffle exchanges with distant neighbours (according to the overlay neighbourhood relations and not necessarily related to any other distance metric, such as network distance), the quality of the overlay is further improved as it becomes more resilient to partitioning. This resilience comes directly from the maintenance of a large passive view and from the random walk that avoids the passive view to cluster among a set of neighbours.

The join mechanism assumes the existence of a well-known contact node and is depicted on Listing 2.3. The joining node sends a *Join* request to that contact node, lines 1 and 2, and it will be integrated into that node's active view as can be seen in lines 4 to 9, even if an existing node in the active view must be dropped. Additionally, the contact node will send a *ForwardJoin* request to all the nodes in its active view, in order to ensure that the joining node is known by enough nodes in the overlay. The *ForwardJoin* procedure is a random walk across the overlay parametrized by the $ActiveRandomWalkLenght(ARWL)$, a protocol parameter, and is depicted in lines 11 through 18. There is another protocol parameter $PassiveRandomWalkLenght(PRWL)$ that indicates at which point in the random walk the joining node should be integrated into the passive view. Upon expiration of the random walk, lines 12 and 13, the node is integrated into the active view, even if an existing nodes has to be dropped, which happens if the active view is full. The same applies to the integration on the passive view. When a node is removed from another node active's view, lines 6 and 29, the formed is informed via a *Disconnect* message, removes the sender from its active view and integrates it on the passive view, as it is

```
 1 upon init do
 2   send(contact,Join(myself))
 3
 4 proc handleJoin(newNode)
 5   if isFull(activeView)
 6       dropRandomElementFromActiveView()
 7   activeView $\leftarrow$ activeView $\cup$ newNode
 8   foreach n $\in$ activeView and n $\neq$ newNode
 9       send(n,ForwardJoin(newNode,ARWL,myself))
10
11 proc handleForwardJoin(newNode,timeToLive,sender)
12   if timeToLive == 0 || #activeView == 0
13       addNodeActiveVew(newNode)
14   else
15       if timeToLive == PRWL
16           addNodePassiveView(newNode)
17       n $\leftarrow$ n $\in$ activeView and n $\neq$ sender
18           send(n,ForwardJoin(newNode, timeToLive$-$1, myself))
19
20 proc dropRandomElementFromActiveView()
21   n $\leftarrow$ n $\in$ activeView
22   send(n, Disconnect(myself))
23   activeView $\leftarrow$ activeView $\setminus$ n
24   passiveView $\leftarrow$ passiveView $\cup$ n
25
26 proc addNodeActiveVew()
27   if node $\neq$ myself and node $\in$ activeView
28       if isFull(activeView)
29          dropRandomElementFromActiveView()
30       activeView $\leftarrow$ activeView $\cup$ node
31
32 proc addNodePassiveView(node)
33   if node $\neq$ myself and node $\notin$ activeView and node $\notin$ passiveView
34       if isFull(passiveView)
35           n $\leftarrow$ n $\in$ passiveView
36   passiveView $\leftarrow$ passiveView $\setminus$ node
37
38 proc handleDisconnect( peer)
39   if peer $\in$ passiveView
40       activeView $\leftarrow$ activeView $\setminus$ peer
41       addNodePassiveView(peer)
```

Listing 2.3: HyParView Protocol

possible to observe in lines 38 to 41.

2.2.2 Hierarchical/Locality-aware Protocols

This subsection covers the state of the art in membership construction protocols that take into account locality, and therefore result in overlays that mimic the underlying network topology according to a cost function. This function is abstracted out of the models and should provide information to the protocol about the willingness to establish remote links.

Directional Gossip

Directional Gossip [25] aims at providing a gossip-based reliable multicast service in a Wide Area Network (WAN) scenario. This is achieved by using two different gossip levels: one that runs on the Local Area Networks (LAN), and the other that is deployed in the WAN, and encompasses the composing LANs. At the LAN level, a standard gossip mechanism is used to disseminate application level messages within that LAN. For each LAN, one or more nodes are elected as gossip servers and serve as the gateway for the inter-LAN communication. Upon reception of a new message from its LAN, the gossip server disseminates that message to the other LANs via the WAN links. On reception of a message from a remote location, the gossip server is responsible to disseminate that message within its LAN, using the standard gossip protocol deployed there. By using the notion of gossip servers to handle the traffic that crosses the WAN links, the authors are able to effectively reduce the load imposed on those constrained, long-distance links.

Gossip servers get to know each other by means of an external mechanism provided by the administrator. As the state maintained by each gossip server is probably small, it consists of the information about the other gossip servers, the authors suggest the possibility of using replication to handle the failures of the gossip servers.

Localizer

The Localizer [27] protocol defines a mechanism to refine overlays built by Scamp, based on a cost function. With this refinement, it is possible to define an adequate cost function, in order to bias the overlay to the desired network topology, mitigating the network mismatch problem. Additionally, the refinement improves the degree balancing of the original protocol to achieve better quality overlays. The protocol periodically proceeds to links exchanges in order to bias the overlay, in a series of steps described below:

- Each node chooses two random nodes from its neighbourhood, calculates the link cost to each one, according to the defined cost function and sends those values to both;

- The receivers reply with their respective degrees and additionally, one of them sends to the initiator the cost of establishing a link with the other node;

- The initiator evaluates locally the gain of exchanging one of its links to the other nodes with a link between them, taking into account the calculation done in the previous step;
- If the gain is desirable, the initiator instructs the other nodes, with a given probability, to establish a link between them. The probability of the transition specifies a trade-off between the speed of convergence and the closeness to a optimal configuration;
- If the transition is successful, then the initiator drops one of its links, behaving in a self-sacrificing manner.

Additionally, to promote the healthy renewal of links, each nodes has a lease time. Upon expiration of the lease, the nodes connected by it simply drop the link. Nodes who get disconnected by this procedure rejoin the overlay.

With this procedure, Localizer is able to effectively bias the overlay accordingly to the cost function, thus mimicking the network topology while at the same time improving the resilience to faults.

Low Link Costs and Short Paths Overlay Networks

In [26], the authors build on top of the Localizer protocol that approximates the overlay to the network topology, and attempt to obtain an overlay with low link costs and short paths. According to the authors, in this protocol, a link exchange only requires two participating nodes, while on Localizer it requires three. Furthermore, the initiator does not loose one of its links which eliminates the self-sacrificing behaviour of Localizer.

To achieve this, a node is selected with a given probability as a special node. If selected as a special, a node randomly picks one of its links and manages it as a special link.

After this initial step that determinates whose links are to be considered special, each non-special nodes periodically performs the following actions:

- The node selects one of its links that is not a special link managed by other nodes, and sends a message to the node connected to that link;
- The receiver sends to the initiator the set of all its neighbours;

- The initiator removes all its neighbours and itself from the received set. If the resulting set is empty the procedure ends here, otherwise it continues;
- The initiator communicates with all nodes in the resulting set, in order to calculate the cost of each link;
- After, the initiator chooses the link that provides the greatest gain, if any, and establishes a connection to that link, removing the one pointing to the selected target.

Special nodes execute the same procedure, with the exception that a link is replaced by a long distance link only if the chosen link is the special link managed by that node, as chosen initially.

As pointed by its authors, this protocol has not been evaluated in the presence of node leaves, either due to failures or disconnection.

HiScamp

HiScamp [16] is a hierarchical overlay construction and management protocol that leverages on the previous work done in Scamp. It uses a distance function to cluster close nodes, therefore defining a hierarchy of clusters that could span multiple levels. Each level runs an instance of Scamp in order to provide the reliable dissemination service. Each cluster is seen at the next level as a single abstract entity, represented by one or more nodes. With this hierarchy it is possible to reduce the load imposed on costlier links, as messages are targeted almost within each cluster. The protocol uses two views: an $inView$ to handle subscriptions, and a $hView$ used in the dissemination of application level messages. The $hView$ has as many levels as the hierarchy, where the lowest level contains gossip targets in the same cluster, and the other levels contain targets on the same hierarchy level. The $inView$ has one lesser level than the hierarchy that is common to all nodes in the same level, and contains all nodes belonging to that level.

The joining process involves several steps, and works as follow:

- A joining node sends a subscription request to a pre-determined well-known close node, where this closeness is given by the cost function;
- If the distance of the joining node is below a preset value, the node is included into the cluster as follow:

  - As in Scamp, the contact node creates several copies of the subscription and forwards it to its neighbours in the level one $hView$;
  - The forwarded requests are handled just as in Scamp, and eventually integrated into the receivers level one $hView$;
  - Finally, the views of the joining node are initialized as follow: the level one $hView$ contains just the initially chosen contact node, and the other levels of $hView$ are empty, and the $iView$ becomes the same as the contact node, by having it send a message with this information.

- If the distance exceeds the preset value, the joining node creates a new cluster and its subscription is thus handled at the second level of the $hView$ as follow:

  - The contact node uses its $iView$ that contains the identifiers of the other clusters to forward several copies of the joining request;
  - The subscription is handled as in a normal Scamp instance, and eventually integrated in the $iView$ and level two $hView$ of the receiver;
  - The nodes who integrate the subscription gossip the addition of the joining node to their $iView$, in order to update the $iView$ of the nodes in its cluster;
  - Finally, the level one $hView$ of the joining node is set to empty and its level two $hView$ and $inView$ are initialized to contain only the contact node.

To overcome the single point of failure that comes from the inter-cluster links only connecting the nodes which created each one of the clusters, HiScamp periodically runs a routine to balance the $hView$ levels higher than one and therefore, ensure that inter-cluster messages are handled by more than one node.

As inter-cluster messages are only handled by few nodes, HiScamp effectively reduces the stress imposed on long distance links, but at the cost of decrease reliability. For instance, as pointed by the authors, with more than 20% node failures the number of reachable nodes drops below 90%.

2.2.3 Dissemination Protocols

Emergent Structure in Unstructured Epidemic Multicast

The Emergent [7] dissemination protocol foundation stems from the observation that by combining the eager and lazy push strategies it is possible to obtain a wide range of latency versus bandwidth trade-offs. The challenge therefore is to do so without impairing the reliability guarantees that characterize gossip-based dissemination protocols.

This is achieved by delegating the choice of the particular strategy to use to an oracle. The oracle is abstracted out of the model used to prove correctness and instructs the protocol about the dissemination strategy to use for a given node. The authors are then able to prove the protocol's correctness and liveliness despite the strategy chosen by any particular node. In fact, different nodes could choose different dissemination strategies, i.e. eager or lazy push, based on local knowledge only, to provide several trade-offs suited to a wide range of scenarios.

By configuring oracles out of model used to prove correctness, relying only on local knowledge, and allowing different nodes to used different, independent strategies, the protocol is able to adapt progressively and with low latency to different scenarios, which are essential properties to build confident and self tunning protocols, as been argued in [28].

The Emergent protocol is divided in two distinct layers, a basic gossip protocol depicted in Listing 2.4 and the actual point-to-point communication, shown in Listing 2.5

The layer presented in Listing 2.4 is the one offered to the application via its *Multicast* primitive and the *Deliver* upcall. Upon injection of a new message on the system by the application, by invoking the *Multicast* primitive, the protocol creates a unique identifier, the message round is initiated to zero and the message payload is forwarded, as can be seen on lines 4 and 5. In *Forward* the message is delivered to the application (line 8), its identifier is added to the set of known messages to avoid the delivery of duplicates (line 9) and, if the current round number is inferior to the maximum round number t, a protocol parameter, the peer sampling service is consulted to obtain $fanout$ communication targets, another protocol parameter (lines 11 and 12). After obtaining the peer identifiers, the $L - Send$ primitive of the point-to-point communication layer is invoked for each one of them (lines 13 and 14). Upon reception of a message, its

```
1  initially
2   K = ∅ /*known messages*/
3
4  proc Multicast(d)
5   Forward(mkdId(),d,0)
6
7  proc Forward(i,d,r)
8   Deliver(d)
9   K = K ∪ {i}
10  P = ∅
11  if r < t
12     P = PeerSample(fanout)
13  for each p ∈ P
14     L−Send(i,d,r+1,p)
15
16 proc L−Receive(i,d,r,s)
17    if i ∉ K
18       Forward(i,d,r)
```

Listing 2.4: Basic Gossip Protocol: Peer Selection

identifier is checked against the known identifiers and, if the message is new, it is forwarded, as depicted in lines 16 to 18.

We will now analyse the point-to-point communication protocol, depicted in Listing 2.5. In this layer two sets are maintained, one that holds the message payloads, used when nodes lazily request the payload, and other which holds the identifiers of known messages. Upon call of the $L - Send$ primitive by the previous layer, the oracle, abstracted by the *isEager* primitive, is consulted to infer whether the message payload shall be sent eagerly or lazily (line 6). In the latter case the message payload is stored to allow for a future retrieval by lazy pushing nodes. Additionally, the protocol sends an advertisement message to the target, the $IHAVE$ message on lines 9 and 10 and the message identifier is added to the set of known messages.

Upon the reception of a message payload, on line 17, the identifier is checked against the known set of messages. If the message is not known by the protocol, its identifier is added to the set of known messages (line 19) and any pending request on the payload are cleared (line 20). Nonetheless, and at first sight counter-intuitive, the payload is delivered to the higher level via the $L - Receive$ upcall, even if it has already been delivered. The reception of a $IHAVE$ message on line 13 indicates that the sender has a copy of the message payload. If the message is not known, its payload is queued for retrieval in a point in the future. The details of the scheduling policy are abstracted in the protocol by means of the $ScheduleNext$ primitive on lines 27 to 29. This procedure runs continuously and is responsible to lazily push advertised message payloads.

```
1  initially
2    $\forall$i: C[i] = $\perp$ /*cached data*/
3    R = $\emptyset$ /* known messages*/
4
5  proc L−Send(i,d,r,p)
6    if isEager(i,d,r,p)
7        send(p,MSG(i,d,r,myself))
8    else
9        C[i] = (d,r)
10       send(p,IHAVE(i,myself))
11   R = R $\cup$ {i}
12
13 proc handleIHAVE(i,s)
14   if i $\notin$ R
15      QueueMsg(i,s)
16
17 proc handleMSG(i,d,r,s)
18    if i $\notin$ R
19       R = R $\cup$ {i}
20       Clear(i)
21    L−Receive(i,d,r,s)
22
23 proc handleIWANT(i,s)
24    (d,r) = C[i]
25    send(s,MSG(i,d,r,myself))
26
27 forever
28   (i,s) = ScheduleNext()
29    send(s,IWANT(i,myself))
```

Listing 2.5: Point-to-Point Communication

The design decision to deliver a message to the peer selection layer even if the message is already known (lines 17 to 21) stems from the well-known best practice 'premature optimization is the root of all evil'. In fact, by choosing not to deliver the payload to the upper level the applicability of the protocol to new unpredicted scenarios will be restricted. For instance, the basic gossip protocol layer could be replaced by a version where receiving duplicate payloads is important and as such not be feasible if the point-to-point communication layer does not provide this. In [30] the authors reason about the impact of premature simplifying assumptions with different studies and argue that simplifications may reduce the applicability scenario of several well-known protocols. Nonetheless, in this setting, duplicates are filtered in the peer selection layer, as can be observed in lines 16 to 18 of Listing 2.4.

Chapter 3

Problem Statement

Fixed formation is bad. Study this well.

Miyamoto Musashi

This Chapter presents the problem addressed in this dissertation, discuss its worthiness in todays IT world, and argues why the proposals reviewed in the State of the Art in Section 2.2 do not satisfactorily solve the presented problem.

As outlined in Chapter 1, the current trend in the IT ecosystem is to move again to centralized platforms that offer a given service to its customers by means of multi tenancy mechanisms.

To support the global and reliable delivery of those services in a worldwide, previously unseen, very large scale, service providers have to solve a variety of challenging issues in order to fully realize the Cloud Computing model. These challenges range from the low level infrastructure management to the higher level billing mechanisms, passing by proper isolation among customers in order to support multi tenancy and adequate delivery of services. The supporting infrastructure for all the stack of services is based around the computational power present in the worldwide deployed data centers of the service providers. Those data centers are composed of thousands to hundreds of thousands of individual nodes organized in a tree-like fashion. This organization comes directly from the actual networking technology that aggregates nodes in the order of several dozens around multiplexer network devices, such as switches and routers. Those devices are then aggregated behind other higher capacity, and more expensive devices in a hierarchical fashion, forming a tree-like structure containing several branches and roots to cope with scalability demands and fault tolerance. This organization is also extrapolated to inter-data centers connections linked together by expensive high-bandwidth links. Whereas communication among nodes connected to a single network device is relatively cheap, both in terms of bandwidth available and latency due to several optimizations that could be done in the networking stack, as we move up in the networking tree, the communication cost increases progressively with respect to latency and bandwidth, and ultimately in the financial burden too as internetworking devices on the top of the tree are more expensive. This is because networking devices close to the top of the tree have to handle all the traffic among the different branches and thus the aggregate bandwidth requirements become quite high [3]. As such, these scenarios are composed by a wide range of links and networking devices with different capabilities and characteristics that need to be considered when deploying a global communication service.

The reliability of such service is paramount to the global management of the infrastructure, as nodes unreachable by the communication service can be considered non existing nodes, as there is no mechanism to manage parts of the system which are not accessible. Administrators could then leverage on a reliable communication service in order to deploy on the infrastructure the essential building blocks for a proper management framework, such as data aggregation and distributed agreement. Distributed agreement [29] is related to the necessity of making decisions in a distributed system, such as on which node to place a given customer, or decide about the outcome of a distributed transaction. Aggregation is a powerful tool to infrastructure management, as it provides mechanisms to query, combine, data mine and present the information made available by individual nodes in a scalable fashion [36]. Both building blocks have clear advantages on relying on a reliable communication service, focusing instead on the concrete problems they are aimed at solving.

From the points stressed above, it is now clear that a reliable communication service is key to enable the reliable construction and provisioning of very large scale service platforms. However, those emerging platforms have particular needs and semantic requirements due to the inherent organization of its underlying infrastructure. In fact, the hierarchic organization is not neglectful to an equal handling of the network links because, as pointed above, they present different characteristics and typical loads and therefore the reliable communication service must take this individual characteristics into account.

Additionally, on systems of this very large scale, the dynamics should not be left out off the equation, or the reliability of the communication service will become severely endangered. This comes from the fact that change is a natural part of those systems, due to the large scale itself, as nodes will constantly join and leave the system due to failures or periodic maintenance operations. The larger the system, the greater the impact of this dynamics as it is highly likely that at any given time some nodes, somewhere, will be joining or leaving the system due to an arbitrary, maybe unknown reason. Furthermore, these unpredictable dynamics may lead to the *physical* disconnection of parts of the infrastructure, for instance due to the failure of an intercontinental link connecting two separate data centers. As such, the reliable multicast service must be robust enough to tolerate considerable amounts of failures, and resilient to the churn phenomena, ensuring that it will continue to function as expected in such harsh conditions.

With the constraints and requirements presented above, we intend to build a multicast service that:

1. Reliably delivers the application messages to the correct participants;
2. Differentiates links according to their characteristics;
3. Adapts to ever changing system sizes;
4. Tolerates considerable amounts of failures of both nodes and links;
5. Mitigates the churn effects.

The first requirement is the most important in a reliable communication service, as non-delivered messages could compromise the semantics and correctness of the application. Although a wide range of applications could tolerate message omissions, our service is aimed at applications with more stringent requirements and as such it should deliver all messages to all correct participants. The second requirement is related to network awareness and is essential in the context of a cloud scenario. Failure to take into account the network topology will seriously compromise the performance and reliability of the service as the links inter-connecting the branches close to the top of the tree will easily become a bottleneck. With those links overloaded the performance degrades, as both the effective bandwidth available decreases and the latency of message transmission increases, up to a point where the reliability of the network could become compromised, as there is too much load imposed on it. The third requirement is important to the long term reliability and performance of the communication service. Despite adding or removing some nodes on systems of this scale may be negligible, in the long run the system must cope with the addition or removal of considerable amounts of nodes, such as when adding or removing a data center to the federation, due to administrative or business reasons and prolonged failures that may physical disconnect substantial parts of the system. The fourth and fifth requirements are also a consequence of the targeted very large scale scenario. In it, faults are a natural part of the system, and consequently churn, and therefore the developed reliable multicast service must be resilient and well performing in the presence of this phenomena.

To summarize, we intend to design a very large scale communication service that focus on two of the problems identified in [12]: network awareness and adaptability while offering strong reliability guarantees and ideally performing well.

Chapter 4

Network-Aware Epidemic Broadcast

Management of many is the same as management of few. It is a matter of organization.

Sun Tzu

This Chapter carefully describes the developed protocols and present the intuition and justification of the design choices taken. The first Section justifies the approach taken, weighting the advantages and disadvantages of each proposal available. The structure of the remaining chapter reflects the clear differentiation made between the two different but related problems that arise when building a reliable dissemination protocol on top of an unstructured overlay network. The first problem deals with the construction and maintenance of the overlay, taking into account all the requirements outlined in Chapter 3, and is presented in Section 4.2. Section 4.3 describes the design decisions made to build a reliable dissemination protocol on top of the previously presented overlay.

4.1 Approach

This preliminary Section provides the rationale for the research direction taken to address the requirements presented in the previous Chapter, by recurring to the concepts and state of the art review presented in Chapter 2.

As the reader may remember, there are two main approaches when building the supporting infrastructure upon which a reliable multicast service can be deployed: structured and unstructured overlay networks. Thus, this is the natural first choice to make when designing such service. Structured overlay networks are very effective in resource usage of both links and nodes due to the explicit knowledge they impose on the construction of the supporting overlay. The dissemination tree is pre-built taking into account this structure, and application level messages are relayed on top of it. Furthermore the dissemination tree could be optimized to a given performance criteria, such as bandwidth or latency, and take advantage of links and nodes with higher capacity by placing them closer to the root of the tree. Thus, structured overlay networks are an attractive approach to handle links and nodes with different characteristics. Unfortunately, upon failures and network reconfigurations, the dissemination tree needs to be rebuilt, which makes this class of protocols considerably sensitive to churn. On the other hand, on unstructured overlay network protocols, the dissemination effort is evenly spread among all the nodes in the overlay, which enables their natural scalability and resilience. As such, we have the efficient structured approach versus the resilient unstructured one. Due to the very large scale and churn of the scenarios our communication service is aimed at, we will rely on the resilience of the unstructured approach and improve it to approximate the desirable performance metrics.

With this preliminary decision set, we still have to decide which one of the two unstructured overlay construction approaches, flat or hierarchical, is best suited to fulfil our goals. In the flat approach, nodes and links are treated equally, and as such are not suited to handle our requirement of taking into account the link characteristics when building the overlay. On the other hand, hierarchical approaches clearly differentiate desirable and undesirable links enabling the construction of a network aware overlay. Unfortunately, the proposals presented in the state of the art review, rest on post optimizations to the overlay, such as [26,27], on a selection of special nodes to handle the traffic that traverses costlier links, such as [25], or in having nodes behave in a self-sacrificing manner by loosing one of their links [27] in order to approximate the overlay to the desired network topology. Those proposals have serious drawbacks, as it is not clear how and when to choose those special nodes, or when to apply the biasing to the overlay. Furthermore, having nodes with special roles further inhibit the reliability of the overlay, as questions such as how to select those nodes in a distributed and automated fashion, how to handle their failures and how to make them known to each other must be answered in order to provide a truly resilient solution. Moreover, having nodes drop links to bias the overlay to a given criteria may impair its reliability as nodes that already have few links become prone to disconnection.

In our approach, we completely part away from these brittle design decisions, by refusing to rely on nodes with special roles, and focus on the locality awareness of the overlay at construction time, as locality is a natural characteristic of the systems where our proposal is intended to be deployed. The guiding principle is that if all the nodes could contribute to some extent to the locality awareness, as they contribute to the dissemination effort, a globally network aware overlay shall emerge naturally without compromising scalability, resilience and reliability. With this principle, we are able to reduce the load imposed on the undesirable links by an order of magnitude in a natural manner, while leveraging on the scalability and resilience to churn and faults of unstructured overlay networks. As such, we designed a novel hybrid proposal, where all nodes are treated equally as in the flat approach, but the establishment of links among them takes into account locality, as in the hierarchical approach. Furthermore, our proposal naturally adapts to changing system sizes, by transparently tunning the number of links that each node maintains with its neighbours.

Looking at the proposals available to address reliable multicast in very large scale systems in a top down manner, we successively discarded the proposals with the best performance to give preference to the reliable ones, as reliability is the most important metric in a reliable dissemination service. Then, we build up our mechanisms on the most reliable proposals available, flat unstructured overlay networks and improve their performance to achieve the remaining goals.

4.2 Peer Sampling Service

This Section carefully describes the Peer Sampling Service developed, starting up from a flat unstructured network protocols, as previously justified.

With the decision of addressing the reliable multicast problem with flat unstructured network protocols, the next natural step is to look at the available solutions and infer whether there is some previous work on which we could leverage some of our requirements. Looking at the proposals reviewed in 2.2.1 there is one requirement, adaptability, that is clearly addressed by one of the protocols, Scamp [15]. The requirement of adapting to ever changing system sizes in order to transparently scale without user intervention is addressed by the Scamp protocol, which is able to adjust the view size to the correct value. Nonetheless, Scamp is completely oblivious to locality and its hierarchical derivatives rely on specialized nodes, although chosen randomly, to address the network mismatch problem, which is conflicting with our previous decision of not relying on any kind of special nodes with particular roles.

This preliminary thoughts shown that no protocol available neither their approaches, is capable of addressing all the requirements we imposed. As such we depart way with the proposals done before by taking a novel approach that effectively address all the requirements presented in the previous chapter.

Due to its interesting convergence to the right node degree with respect to system size, our starting point will be the Scamp protocol. However, instead of building hierarchical strategies on top of it as done previously, we continue with a flat approach where all nodes are treated equally with respect to the roles they exhibit in the overlay, therefore not colliding with the 'no reliance on special nodes' assertion. Furthermore, the adjustment to the network topology is done at construction time instead of post-optimizations to the links established among the nodes. With the initial research path set, and the reasons that lead to it explained, the rest of this Section focus on the description of the developed protocol and the intuition behind it.

4.2.1 Network-awareness

If we focus on the Scamp protocol, we will observe that joining nodes are integrated into the view of a node with a given probability that is function of the actual view size of that node. By making the probability of integration inversely proportional to the view size, and always forwarding the subscription to other nodes if the integration is not successful, the nodes converge naturally, and in a completely decentralized fashion, to the adequate view size, on average. The full understanding of this behaviour is fundamental to the modifications we do in the original protocol, in order to make it cope with our locality awareness goals. By modifying the integration probability of joining nodes in order to take into account the locality, we are able to bias the overlay to mimic the network topology without endangering the properties of the original protocol. This is done

```
upon init
  contact = getContactNode()
  view.Add(contact)
  send(contact,Subscription(myId))

proc handleSubscription(nodeId)
  for n ∈ view
     send(n,Join(nodeId))

  for i=0; i < c; i++
     n = randomNode(view)
     send(n,Join(nodeId))

proc handleJoin(nodeId)
  keep = randomFloat(0,1)
  keep = Math.Floor( localityOracle(viewSize,nodeId) * keep)

  if (keep == 0) and nodeId ∉ view
   view.Add(nodeId)
  else
     n = randomNode(view)
     send(n,Join(nodeId))
```

Listing 4.1: Clon protocol

```
proc localityOracle(viewSize,nodeId)
  if isLocal(nodeId)
    return viewSize * 0.7
  else
    return viewSize + viewSize * 0.3
```

Listing 4.2: A possible *localityOracle*

indirectly, by manipulating the view size of the node receiving the subscription. In detail, if the joining node is considered local, with respect to an abstracted metric, the view size of the node is virtually decreased, which effectively augments the probability of integration of the joining node in the local area it belongs to. On the other hand, if the joining node is considered remote, the view size is virtually increased, reducing the probability of integration in foreign areas. This modification promotes the establishment of links among local nodes in detriment of remote ones, which adjusts the resulting overlay to the underlying network topology. The resulting protocol has been named Clon , which stands for Overlay Networks for Cloud Environments, as federated clouds are a common scenario where several highly intra-connected data centers are spread around the world and connected by costlier inter-continental links as pointed in the introductory chapter. The pseudo-code for the protocol is presented in Listing 4.1, and we will carefully describe it next.

The initial bootstrapping and joining mechanism remains the same of the original protocol. After the choice of the initial contact node, the joining node sends it a subscription request on lines 2 to 5. Then in lines 7 to 13, the receiver forwards the subscription to all its neighbours, creates c additional copies and forwards it to random neighbours in its view. c has the same impact as in the original protocol, and is related to the amount of faults tolerated.

Upon reception of a join request, line 15, the *keep* variable is initiated with a random value, and then the probability of integration is calculated taking into account the locality of the joining node. The adjustment of the view that takes into account locality is delegated to the *localityOracle*. This oracle is therefore responsible to access whether the node is local or not, and manipulate the view size according to that.

As an example we give a possible *localityOracle* on Listing 4.2. This oracle reduces the view size by 30% if the node is considered local, or increases it by 30% if the node is remote, effectively manipulating the probability of integration of remote and local nodes. The particular details of how to detect the locality of the node are abstracted out of the model by the *isLocal* primitive, and could be calculated by several mechanisms, such as observing and comparing the IP addresses of the joining and receiving nodes, and determine whether or not they are on the same local area network. If the node is considered local, the oracle should virtually decrease the view size in order to increase the probability of integration, or proceed otherwise if the node is remote. It is important to note that the oracle only returns the perceived view size, and should not manipulate the view, for instance by dropping nodes, or in any other way.

The impact of changing the probability of integration according to the localization of the joining node, effectively biases the overlay to the underlying network topology, without major impacts on the reliability of the protocol in face of failures. A detailed experimental assessment of the properties of the overlay obtained can be found in Section 5.2. Furthermore the experimental analysis of the impact of this alterations on the load imposed on the long distance links can be found in Section 5.3.

4.2.2 Degree Balancing

So far, we have focused on how to properly bias the overlay in order to mimic the underlying network organization. However, if we focus on the obtained protocol, we will observe that it still has some important limitations, as the original Scamp protocol: the distribution of the nodes' degree, and the bootstrapping process, which requires a set of well-known nodes to initiate the subscription.

The first problem, the distribution of the nodes' degrees, comes directly from Scamp being a reactive protocol, i.e. it only modifies the overlay in the presence of leaves or joins, and is further aggravated by the bootstrap mechanism. A node wishing to join the overlay must contact a well-known node, establish a link with it, and send the subscription request. As such, for a given period of time, or forever, if the membership remains stable, the last nodes to join the overlay only have one outgoing link, i.e. they only known one neighbour, which is the initial contact node. This clearly impact the reliability of the proposal, as this nodes are prone to disconnection because the inherent link redundancy of gossip based protocols is not present. On the other hand, older nodes tend to have much more neighbours than newer ones, particular the contact nodes and its closest neighbours. This happens because even though the probability of integration is probabilistic and based on the view size, those nodes received high amounts of subscriptions, almost from all the nodes in the overlay, and as such some of them will be eventually integrated, despite the low probability. As such, the convergence to the average degree that Scamp offers, is misleading, as certain groups of nodes tend to be much below or above the ideal degree and therefore impair the quality of the obtained overlay.

To overcome this deficiency in the protocol, we devised a degree balancing mechanism that normalizes the distribution of the nodes' degree in a distributed fashion. The basic idea behind the mechanism is to drop excessive links from nodes with high degrees and integrate them in the nodes with lower degrees. However, we intend to do so in a decentralized fashion, without direct node interaction and any kind of agreement, based only on local decisions, and without having nodes assume special roles in order to proceed with the link exchange. Furthermore, this exchange also needs to take into account locality, or the work developed to bias the overlay to the network topology will be lost after several runs of the degree balancing mechanism.

Based on those principles, we developed the mechanism depicted in Listing 4.3 and will discuss it next.

Periodically, a node initiates a random walk by choosing a random neighbour from its view, and sends it a request with the following information: a given TTL which will specify the length of the random walk, and the number of remote and local neighbours it knows, as can be observed in lines 1 to 3. The random walk then traverses the

```
1  every ΔT
2   target = getRandomNode(view)
3   send(target,NODEINFO(TTL,localNeighboursSize(),remoteNeighboursSize()))
4
5  proc handleNODEINFO(TTL,localNS,remoteNS):
6   if --TTL > 0
7      target = getRandomNode(view)
8      send(target,NODEINFO(source,TTL,localNS,remoteNS))
9   else
10       myLocalNS = localNeighboursSize()
11       myRemoteNS = remoteNeighboursSize()
12       if localNodeDifference(myLocalNS,localNS)
13          droppedNode = dropLocalNode()
14          n = randomNode()
15          send(n,Join(droppedNode))
16       if remoteNodeDifference(myRemoteNS,remoteNS)
17           droppedNode = dropRemoteNode()
18           n = randomNode(view)
19           send(n,Join(droppedNode))
20
21 #sample
22 proc localNodeDifference(myDegree,receivedDegree)
23    return (myDegree − receivedDegree) / 2 > receivedDegree
24
25 #sample
26 proc remoteNodeDifference(myDegree,receivedDegree)
27    return (myDegree − receivedDegree) / 2 > receivedDegree
```

Listing 4.3: CLON normalization algorithm

overlay by the number of hops specified by the TTL (lines 6 to 8) until it eventually expires. Upon termination, the receiving node calculates its number of local and remote neighbours and weights those values against the local and remote number of neighbours received via the random walk. If the difference between the receiving node's degree and the one obtained through the random walk is substantial, then the receiver drops one of its links (lines 14 and 18) as this hints that it has more links than the average and as such is reducing the quality of the overlay. The particular calculation to determine whether or not the difference between the degrees is relevant (lines 21 and 24), could be obtained in many different ways in an application dependent fashion. In our Listing we give an example of such calculation, by considering that the degrees differ too much if half of the difference between them is higher than the subtrahend, but more stringent or relaxed calculations could be taken in face of the scenario requirements. By adjusting the period T of this procedure and/or the oracles that determine the difference between the degrees the programmer could adjust the speed of convergence to the ideal node degree distribution, considering the application demands. Finally, the dropped link, if any, is forwarded as a join request, as if the node pointed by the link has just joined the overlay (lines 15 and

19). This allows us to take advantage of the integration mechanism already deployed and, therefore to continue to improve the overlay with respect to the network topology. This balancing procedure does not require direct intervention between the intervening nodes (the node who drops the link and the node whose link has been dropped) and as such is completely decentralized. It is important to note that the only decision nodes could take is to drop links. They are forbidden to ask for links as this will imply some coordination among them. Therefore, the nodes with lower degrees will improve their degree indirectly, by integrating the links dropped by the nodes with higher degrees, as the integration is likely to succeed due to the low degrees of those nodes. Furthermore, nodes only require local knowledge to decide whether or not to drop links and the re-integration of the dropped links is done recurring to the normal protocol integration mechanism, avoiding the use of special nodes to integrate those links or decide whether or not they should be dropped.

4.2.3 Bootstrapping mechanism

The last problem we *partially* address, is the bootstrapping mechanism that allows joining nodes to discover one or more peers already in the overlay. To the best of our knowledge, existing protocols solve this by assuming there is an external entity that provide the node identifier(s), in order to allow the joining node to contact peers already on the overlay. This is, in general, not satisfactory as it puts out of the model an important aspect of the overlay building protocol, and tends to be addressed by relying on static centralized solutions such as having one or more servers to provide the set of initial identifiers. With node churn this set is hard to maintain up to date and provides a brittle solution, even if we made the unrealistic assumption that those servers do not fail. In our proposal we address this problem by fully decentralizing this initial discovery mechanism making every node in the overlay a potential server in the true peer to peer spirit. The only requirement we impose is the availability of a broadcast primitive on the local area where the new node is physically connected. This requirement is virtually guaranteed to be satisfied in modern network architectures, due to the pervasiveness of the TCP/IP communication protocol. However, if there are no nodes in the local are, for instance this mechanism is unable to obtain contact nodes but we will elaborate on this issue latter.

In the following we explain the rationale behind the developed bootstrapping mechanism which is depicted in Listing 4.4

Recalling the integration mechanism by which joining nodes get added to the views of other nodes, it is possible to observe that the view size converges, on average, to the ideal value of $log(N) + c$, where N is the system size and c a protocol parameter related to fault tolerance [23]. However, in Scamp and in the previously presented version of Clon a node joining the overlay only establishes one link with the contact node it receives from the external mechanism, which effectively impairs the connectivity of the joiner. Notwithstanding, if instead of obtaining just one contact node, the joiner obtained some contacts, its connectivity will be improved from the beginning. Ideally, this value should be near the average node degree, because in this way the new node will be indistinguishable of nodes that have been on the overlay for more time. Next we will explain the operation of the protocol and then how the ideal results could be achieved.

Upon boot, the node uses the available broadcast primitive to request contacts from all the nodes in its local area, as can be seen on lines 3 and 4. If upon reception of the request all nodes replied to the originator, this could lead to problems, in a phenomena known as acknowledgement bomb. This phenomena stems from the fact that if every node in a large scale system replies to the originator of a broadcast, the network will become suddenly overloaded, and the requester may be overrun by the amount of replies and crash. To overcome this problem, we rely on an oracle that should instruct whether the node should reply to the request or not, with a given probability. Although the oracle may be configured in a naive manner, lets say reply only with a probability of 10%, the amount of replies generated will be effectively reduced, alleviating the problems of the acknowledgement bomb phenomena.

If the oracle instructs the node to reply, i.e. if it returns true (line 7), there is another decision that needs to be made: whether to provide a local or remote node as a contact. If this is not taken into account, and joining nodes are always provided with local contacts only, the reliability of the overlay will be compromised. This is because if joining nodes only get to known local nodes, over time the number or remote known nodes will decrease considerably and the overlay will partition around the local areas, which we certainly want to avoid. In Listing 4.4, we show a simplistic oracle in lines 42 to 44 which instructs the protocol to reply half the times with a remote nodes and half of the times with a local one, but naturally this could be configured to the application requirements. After this initial steps where the oracles decide about replying and the kind of node chosen,

```
myC = c

upon init()
  BCAST(CONTACT, myself)

proc CONTACT(nodeId)
  if (sendOracle())
    if(externalContactOracle(nodeId))
      contact = randomExternalNode(view)
    else
      contact = myself
      #or
      # node = randomLocalNode(view)
    send(nodeId,CONTACTREPLY( contact))

proc handleCONTACTREPLY(contact)
  keep = random()
  keep = Math.Floor((viewSize + 1 * keep)

  if (keep == 0)
    if view.size() > 0)
      schedullecCheck()
    view.Add(contact)
    send(contact,Join(myId))
    if (myC > 0)
      send(contact,Join(myId))
      myC = myC − 1

#possible send oracle
proc sendOracle()
  totalNodesEstimate = $10^{(viewSize-c)}$
  localNodesEstimate = totalNodesEstimate / NUMBER_OF_LOCAL_AREAS

  if localNodesEstimate < 1
    localNodesEstimate = $10^{viewSize}$

  reply = viewSize / localNodesEstimate
  seed = randomFloat(0,1)
  return seed < reply

#possible external oracle
proc externalContactOracle(nodeId)
  rand = randomFloat(0,1)
  return rand < 0.5

proc cCheck()
    while(myC != 0 )
      contact = randomNode(view)
        send(contact,Join(myId))
  myC = myC −1
```

Listing 4.4: CLON contact discovery protocol

the contact is sent to the requester, as can be seen in line 14.

Upon reception of a reply, on line 16, the joining node should decide whether or not to integrate the contact in its view, based on the same procedure of the original Scamp protocol, where the probability of integration is inversely proportional to the size of the view. As the node is bootstrapping this may seem counterintuitive but the motivation behind this conditional integration is to ensure that even if the oracle of the repliers is not well configured, the view size will still be within the normal bounds of the system. Without this restriction the view size of the joining nodes could integrate too much nodes which will therefore impact the quality of the obtained overlay. If the contact node is to be integrated by the joining node, the latter adds it to its view, i.e. establishes a link with it, and sends it a *Join*. With these changes the join mechanism is effectively decentralized and inverted. Instead of being the contact node to send the subscription requests as in the original protocol, it is now the responsibility of the joining node to send them. This change has an immediate impact on the protocol as the c additional join requests sent randomly must still be transmitted. As such the joining node becomes the responsible for sending those additional requests. However, instead of sending those additional join request to just one contact, lets say the first received, we decide to distribute them among the several contacts obtained. To this end, the joining node now has an additional variable myC which is initially set to the c protocol parameter, as seen in line 1. Then, for each received contact, the joining node sends the normal *Join* request plus one additional copy until c copies are sent, lines 25 to 27. To prevent the case where less than c contacts are received, and therefore not enough additional copies could be sent, upon the reception of the first contact the protocol schedules the execution of the *cCheck* procedure on a point in the future. This scheduling may be only approximate and should start when it is expected that all the contact replies have been received, which on a local area shall be pretty close to the first one. This procedure only checks if enough copies have been sent, and if not they are sent to randomly chosen nodes, as in the original protocol.

After describing the protocol it is now time to clarify how we could exploit the local knowledge available, in order to obtain optimal results in the bootstrapping mechanism. Optimal in this context means that the joining node establishes as much contacts as the average view size, therefore becoming indistinguishable from the nodes already on the overlay. To achieve this exact behaviour, we will need global knowledge in order to calculate the ideal node degree and reply exactly with that amount of contacts to the joiner. Of course this solution is not acceptable, as it would impair all the work done previously on decentralizing the entire protocol. Nonetheless, if we rely only on local knowledge it is still possible to approximate this behaviour, in a probabilistic fashion. In fact, all nodes have a powerful estimation tool of the total amount of nodes, the size of their view. In fact the view size converges to $log(N) + c$ where as N is the number of nodes in the system, therefore it is straightforward to estimate locally the total number of nodes. If we also known the number of local areas available, which probably is fairly well-known (for example the number of data centers of a cloud provider), it is possible to estimate the number of potential repliers to the contact request, i.e. the number of nodes in the local area, and thus reply with the adequate probability. An example of an oracle configured in this way is shown in lines 29 to 39. The oracle estimates the total number of nodes (line 31), calculates the number of local nodes based on this estimation (line 32) and replies with a probability based on this calculations (lines 37 to 39). Although the configuration presented should be well suited to a wide range of scenarios, we still abstract it with an oracle in order to not impair the applicability of the mechanism in other, at this time unpredicted, scenarios. It is important to notice that if the estimation of local nodes is inaccurate, which happens when the view size is inferior to c the probability of replying adequately will be compromised. This comes from the fact that if the $totalNodesEstimate$ becomes smaller than 1, in the case c is greater than the $viewSize$ then the calculation on line 37 will yield a value greater than 1 and therefore the node will always reply to the contact request. This is easy to observe as $viewSize/localNodesEstimate$ always yields a value greater than 1, when the $localNodesEstimate$ is strictly smaller than 1, which will impair the optimal behaviour we intend to achieve. Thus, this abnormality is corrected by ignoring the wrong local nodes estimation and making it simply $10^{viewSize}$ (lines 34 and 35).

The advantages of this new bootstrapping mechanism are many fold. First, we eliminate the need to maintain a list of well-known nodes somewhere out of the model, as contact nodes are drawn from all the local nodes on the overlay. As such this also has an impact on the quality of the overlay as contact nodes are chosen more uniformly and therefore the problem of the well-known nodes and its direct neighbours having high degrees is alleviated. Furthermore, a joining node now knowns several initial contact points instead of just one, which effectively improves its connectivity. Finally, the subscriptions along with the c additional copies are sent to different parts of the overlay instead of only the neighbours of the contact node, as in the original protocol, which effectively counters the clustering around those nodes.

Despite this advantages the proposed mechanism still has one important drawback, which is why we initially said that the problem is solved only *partially*. Although this mechanism works well when the local areas are established and there are known remote nodes, the initial bootstrap of a whole local area could not be addressed with this mechanism. This is due to the fact that initially no remote nodes are known on the starting local area and as such we still require a set of well-known remote nodes to bootstrap a whole local area, provided by the administrator. To solve this problem it is possible to rely on the traditional approach of a set of well-known servers when starting a new local area to provide contact nodes, and then switch to our proposal. As initially there are few nodes in the starting local area, the problems with the set of external servers are minimized, as the size of identifiers they need to maintain is still small. Nonetheless, after this initial step the external mechanism could be discarded and as such our proposal may be used for the rest of the life-cycle of the application.

To conclude, the Peer Sampling Service proposed exposes two primitives, *PeerSampleLocal* and *PeerSampleRemote*, which provide a set of local and remote peers respectively. These primitives will then be used by the dissemination protocol described on the next section.

4.3 Dissemination Protocol

This Section describes the dissemination protocol developed, which leverages on the previous work done in the Emergent protocol [7]. For details about Emergent please refer to the background Section 2.1.2 and 2.2.3.

The Emergent protocol offers to the programmer two different dissemination strategies: eager and lazy push. In eager push the latency to infect all the nodes is minimal, as every node eagerly transmits the message payload upon reception to its neighbours. With the lazy strategy, the payload transmission is delayed to a latter phase, and therefore the bandwidth requirements of this strategy are much lighter than in the eager approach, at the cost of increased latency in the dissemination process.

As one of the main goals of this thesis is to reduce the load imposed on the costlier long-distance links, it is possible to take advantage of the different dissemination strategies offered by the Emergent protocol in order to further reduce the number of message payloads that traverse those costlier links.

The rationale behind this is to lazily send messages to the remote nodes, in order to reduce the load imposed on the long distance links, while attaining a desirable latency trade-off. If we use an eager strategy while disseminating in local areas and a lazy strategy when disseminating to remote ones this is achieved in a seamless way, without compromising the reliability of the dissemination. Furthermore, if we tune the protocol parameters to send the messages in a eager fashion to the remote nodes and then, after a small number of rounds fall back to a lazy approach, we could further reduce the overall latency of the dissemination process. The intuition behind this is that that as soon as some nodes in a given local area have the payload of a given message, they could use a eager strategy to quickly disseminate the message in their local area, as bandwidth constraints are more relaxed on local areas than in the links that interconnect them.

The notion of local areas interconnected by expensive links is a pervasive concept across all the developed work, however it is unfortunately absent in the original protocol. As such, this section deals with the deployment of this concept in the dissemination protocol, in order to further reduce the load imposed on the costlier links that connect the different local areas.

The rest of this section describes the changes necessary to enable a locality aware dissemination protocol, which are the following:

- Introduction of two round types to reflect locality;
- Reorganization of the queue of pending message payload requests, to give precedence to local nodes.

4.3.1 Locality awareness on the selection of peers

The introduction of two round types is fundamental to enable the locality awareness of the dissemination protocol. This comes from the fact that if the protocol used only a single round type to disseminate messages, local and remote nodes could not be distinguished on the peer selection part of the protocol, which is shown in Listing 4.5. For instance, it would be impossible to build a dissemination strategy that disseminates only to local or remote nodes, precluding the effective use of the network awareness that the Peer Sampling Service offers by means of its $PeerSampleLocal$ and $PeerSampleRemote$ primitives. Apart from seriously reducing the protocol performance trade-offs achievable, the dissemination protocol would not take advantage of the network knowledge present on the overlay carefully built on the previous section. Thus, two distinct and independent rounds types are used, one to the intra-local area dissemination and other to the inter-local area dissemination. The independence of the rounds is due to the way each one is increased, the local round count is only increased when messages traverse local area links, whereas the remote round count is only increased when messages traverse inter-local area links. Furthermore, when a message is received through a remote area, its local area round count must be reset, because it is meaningless to the current area where the message is being disseminated. Failure to do so will seriously impact the reliability of the protocol, as the message will not be relayed enough times in the given local area, therefore failing to infect all the members of such area. For instance, suppose that the protocol is configured with a given maximum local area round count of $maxRLocal$. As the message is disseminated in the originating local area the local round count naturally increases. Eventually, the message payload will be received by another local area with a local round count of, say $maxRLocal - 2$. If the receiving local area does not reset this counter, the message will only be disseminated for two more rounds (as $maxRLocal - 2 + 2 < maxRLocal$ yields false, see line 14 of Listing 4.5), compromising the reliability guarantees we seek.

With the necessity of two round types, one for local dissemination, and other for remote dissemination of application level messages explained, we will now describe the impact of such changes in the protocol pseudo-code, in Listing 4.5, focusing only on the changes necessary to the original protocol. The introduction of two round types, naturally implies the addition of new parameters to the protocol. These are $maxRLocal$ and $maxRRemote$ which specify the maximum number of rounds a message is to be relayed locally and remotely, respectively, and $remoteFanout$ and $localFanout$, which indicate the number of gossip targets that must be drawn from each set of neighbours. These two last parameters could be expressed in a single $fanout$ parameter and use some sort of weighting to choose between remote and local neighbours, similar to what is done in the

```
 1
 2 initially
 3  K = $\emptyset$ /*known messages*/
 4
 5 proc Multicast(d)
 6  Forward(mkdId(),d,0,0)
 7
 8 proc Forward(i,d,rl,rr)
 9  Deliver(d)
10  K = K $\cup$ {i}
11  P = $\emptyset$
12  if rr < maxRRemote
13    P = P $\cup$ PeerSampleRemote(remoteFanout)
14    for each p $\in$ P
15      L−Send(i,d,rl,rr+1,p)
16  if rl < maxRLocal
17    P = P $\cup$ PeerSampleLocal(localFanout)
18    for each p $\in$ P
19      L−Send(i,d,rl+1,rr,p)
20
21 upon L−Receive(i,d,rl,rr,s)
22   if i $\notin$ K
23     if not isLocal(s)
24        rl = 0
25     Forward(i,d,rl,rr)
```

Listing 4.5: Dissemination Protocol: Peer Selection

peer sampling service, but for the sake of clarity and simplicity we decided to clearly separate them. As such, in the *Forward* procedure, each round count is compared to their respective maximums (lines 12 and 16), and if the maximums have not been reached, the given number of peers is drawn from the respective set (lines 13 and 17), if available. Then the $L-Send$ procedure of the next layer is invoked for each one of the chosen peers. To finalize, in the $L-Receive$ procedure the local round count is reset if the message comes from an remote node (lines 23 and 24). The *isLocal* oracle abstracts the problem of identifying the origin, in terms of locality, of a node and can be built as pointed in the previous section.

4.3.2 Lazy push optimization

While the introduction of two distinct round types is crucial in the dissemination protocol in order to make it locality-aware, the next contribution is a improvement that stems naturally from the observation of the protocol's behaviour when dealing with lazily sent messages. The pseudo-code is presented in Listing 4.6. When the *isEager* oracle that controls the strategy to use when relaying a message to a given node decides to sent a message lazily, two things happen (lines 9 and 10): the message payload is stored in a temporary buffer in order to answer future requests, and an advertisement of the message is sent to the target. When a node receives the advertisement of a message (lines 13 to 15), if the message is not known it is queued for retrieval in a point in the future. The actual scheduling policy is abstracted by the *ScheduleNext* procedure, which is application dependent. Nonetheless, if we observe the pattern of message advertisements/payloads transmitted, it is possible to further reduce the number of message payloads that traverse the costlier links by rescheduling the requests to give precedence to local nodes. In this way, the payloads are lazily pushed over the local area links whenever possible, instead of the long-distance links that connect the different local areas. In fact, if the dissemination strategy is chosen carefully, for instance acting in a pure eager fashion in local areas and also eagerly to remote areas for a small number or rounds and then fall back to a lazy approach, few transmissions are actually made lazily over the long distance links. This is because the initial payloads sent eagerly to remote areas, and gossiped eagerly within that areas, will quickly overrun the necessity to ask for the payload transmission over the long distance links. Nonetheless, this measure is important as a way to reduce the payload transmissions over those undesirable links, whenever such strategy is not feasible or applicable.

To implement this, we modified the original protocol in the following way. When an advertisement of a message is received (line 13), instead of promptly scheduling the request as in the original protocol, the request queue is rearranged (lines 31 to 38) in order to give precedence to request on the local area. If the newly received advertisement source *isCloser* than the already scheduled request, their order is swapped. The *isCloser* relation is abstracted by means of the *isCloser* oracle, which calculates an application level distance between the available message payload sources, and can be built over the *isLocal* oracles defined above.

For the sake of completeness an example of such oracle is given in Listing 4.7. As it is possible to observe, the oracle returns false if and only if the new source for the message is from a remote node and the already known source is from a local node. This oracle configuration has an interesting side effect, if both nodes are at the same distance, i.e. either both are local or remote, the oracle returns true, which effectively swaps the older entry with the new one. As fresh entries are given precedence in the queue, this tightens the time window to node faults, as nodes tend to fail as times passes, therefore improving the confidence that the node who has the required message payload is still alive.

```
initially
  ∀i: C[i] = ⊥
  R = ∅

proc L−Send(i,d,rl,rr,p)
  if isEager(i,d,rl,rr,p)
      send(p,MSG(i,d,rl,rr))
  else
      C[i] = (d,rl,rr)
      send(p,IHAVE(i,myself))
  R = R ∪ {i}

proc handleIHAVE(i,s)
  if i ∉ R
    QueueMsg(i,s)

proc handleMSG(i,d,rl,rr,s)
   if i ∉ R
      R = R ∪ {i}
      Clear(i)
   L−Receive(i,d,rl,rr,s)

proc handleIWANT(i,s)
   (d,rl,rr) = C[i]
   send(s,MSG(i,d,rl,rr))

forever
  (i,s) = ScheduleNext()
   send(s,IWANT(i,myself))

proc QueueMsg(i,newSource)
  if i ∉ Queue
    Queue.add(i,newSource)
  else
    (i,oldSource) = Queue.get(i)
    Queue.add(i,newSource)
    if isCloser(newSource,oldSource)
      Queue.swap(newSource,oldSource)
```

Listing 4.6: Dissemination Protocol:P2P Communication

```

proc isCloser(newSource,oldSource)
  if isExternal(newSource) and (not isExternal(OldSource))
    return False
  else
    return True
```

Listing 4.7: A possible *isCloser* Oracle

Chapter 5

Experimental Evaluation

It doesn't matter how beautiful your theory is, it doesn't matter how smart you are. If it doesn't agree with experiment, it's wrong.

Richard Feynman

This chapter has three main sections. The first describes the experimental environment set up to analyse the impact of the protocol developed with respect to the goals outlined in Chapter 3. Section 5.2 presents the set of experiments conducted in order to assess the quality of the Peer Sampling Service with respect to several graph metrics, analyse the impact of the degree balancing mechanism and the bootstrapping algorithm. Finally, Section 5.3 compares the effectiveness of the Peer Sampling Service in the transmission of messages through long distance links. To this end, we use first a pure eager flooding gossip protocol, and then the improved version of the Emergent protocol in order to attest the improvements brought by a more carefully designed dissemination protocol. For each one of the experiments we present a explanation of the results obtained and discuss the rationale behind them.

As the Peer Sampling Protocol we designed is completely flat, i.e. it does not possess hierarchical characteristics, such as special nodes to handle locality, our proposal is compared against the Scamp protocol. To this end we implemented Scamp, Clon and the improved version of Emergent on the simulator.

5.1 Experimental Scenario Description

The experimental test bed consists of a custom made simulator written in the Python programming language [33]. Python was chosen over other languages due to our fluency with it, and due to its rapid prototyping capabilities, which enable the quick setup and modification of the experimental scenario, to fit the experimentation needs. The simulation is done in discrete time steps, and messages are handled by a global message queue that delivers them to their intended recipients in a First In First Out fashion. The overlay construction and management protocols have been implemented over graphs, by means of the NetworkX graph library. NetworkX is a python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks, modeled as graphs. Depending on the particular experiment, several data is gathered and logged, such as the total, local and remote number of messages received by each node. Due to the large amount of data generated, which amounts to over 60 gigabytes in some experiments, the data is logged to disk for latter post-processing. The processing is done by Python scripts using the R Programming Language [14] to extract the statistical properties from the logged data. R is further used to generated some of the presented graphics, along with gnuplot [1]. R is a programming language for statistical analysis that provides powerful mechanisms to infer the statistical properties of sets of data. The experiments have been run on a 8 core Intel Xeon CPU with 8 gigabytes of Ram and a 500 gigabytes hard drive running the GNU/Linux Ubuntu 8.10 operating system.

The experimental scenario, depicted in Figure 5.1, used in all the experiments consists of 1000 nodes divided in 5 local areas with 200 nodes. Furthermore, we assume that all the local ares are connected to each other by long distance links, in order to provide a federation-like scenario. With this particular setup we always ensure that the number of remote nodes is four times superior to the number of local ones, which is relevant to attest the biasing of the overlay to local nodes.

5.2 Peer Sampling Service Evaluation

In this Section we analyse the quality of the overlay built by our Peer Sampling Service and compare it to an implementation of the Scamp protocol in several relevant graph metrics such as connectivity, clus-

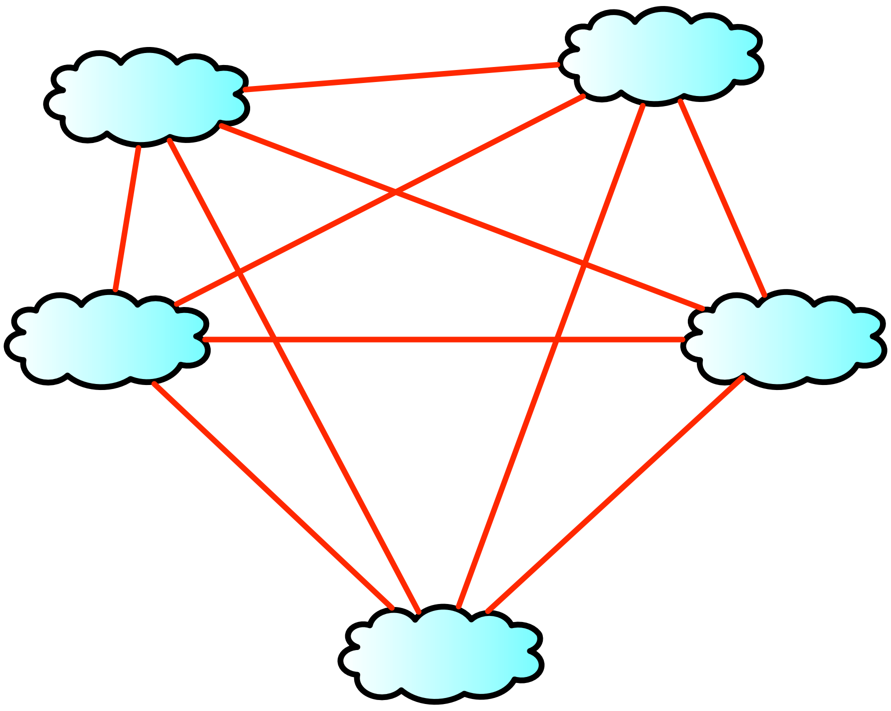

Figure 5.1: Network topology.

tering coefficient and average path length. To access the behaviour of both protocols in the presence of failures we devised three different strategies that randomly remove nodes from the generated overlays. Each strategy randomly drops nodes from the specified universe from 0 to 100% at increasing steps of 10%. The different strategies are: *UniformDrops*, which removes nodes from the overlay in a uniform fashion, considering all the existing nodes; *OneAreaDrops*, which drops nodes uniformly from a given local area; and *TwoAreaDrops*, which disconnects nodes uniformly from two pre-selected local areas. To apply each one of the strategies and their increasing drop rates we proceeded, for both protocols, as follows: first we generated the overlay using the particular protocol and keep a copy of it; after, we apply the given drop strategy for each drop rate and store the intermediate overlays, ensuring that each drop rate is applied to the initial overlay instead of the intermediate overlay generated just before it. For example, a given strategy with a drop rate of 20% is applied to the initial overlay instead of the overlay obtained by the dropping of 10% of the nodes. Furthermore, we do not allow the overlays to heal that is all nodes are removed at the same instant, at the beginning of each experiment. By not allowing the overlays to heal we precisely measure a lower bound on the resilience to massive failures, i.e. the results can be improved by means of healing, but not worsened (if we assume a random distribution of failures). For each one of the obtained overlays we then extract the properties to study which are: connectivity, clustering coefficient and average path length.

In the following experiments both protocols are parametrized with $c = 6$, which indicates the resilient to failures, as explained in [23]. Due to the value of the c parameter and the number of total nodes, the view size of each node converges, on average, to 9 which comes from $log(1000) + c = log(1000) + 6 = 9$. Nodes are created sequentially in the overlay and the contact is chosen randomly across the existing nodes. Furthermore, the locality oracle in CLON is configured in order to obtain, on average, 2 remote and 7 local nodes on the view of each node.

5.2.1 Overlay properties

When building an overlay that should encompass all the nodes in the system, the most important measure is the connectivity of the overlay. If connectivity is not guaranteed in the presence of high churn rates and/or massive failures, the overlay network will partition, isolating one or more parts of the overlay from the rest. Figure 5.2 depicts the results obtained when applying the different node dropping strategies presented above, at increasing rates, without applying the degree balancing mechanism. In the Y axis it is possible to read the amount of alive nodes globally reachable, and in the X axis the amount of dropped nodes for each strategy.

If we analyse the connectivity in the presence of faults in a global setting, by applying the UniformDrops strategy, it is possible to observe that the connectivity of CLON, green line, closely matches that of Scamp, red line, up to 60% global nodes dropped, only breaking up at above 70%. Nonetheless, the results of Scamp from those values up also drop well behind the desired connectivity ratio, thus making the results uninteresting. In fact, for up to 50% failures and without any type of healing, the connectivity is not perceivably affected for both protocols which attest their resilience. For drop rates up to 60%, which means 3 local areas out of 5, the connectivity stays in reasonable values above 90% which means that despite the massive failures, on average each node loses slightly more than half of its links, almost all nodes are still reachable.

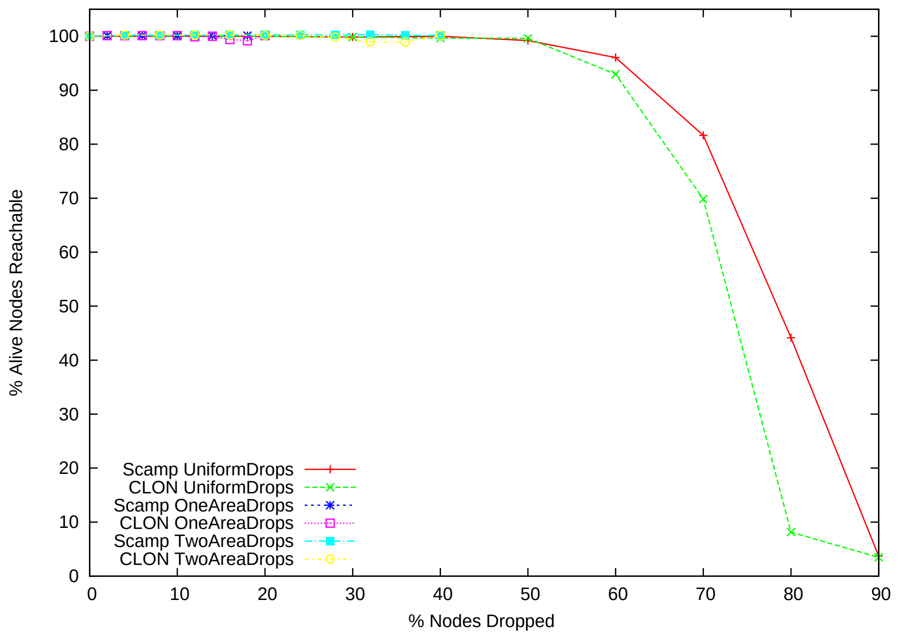

Figure 5.2: Overlay Connectivity.

If we now observe the results for localized drops on one local area, by applying the OneAreaDrops strategy, we observe that the connectivity is not affected for either CLON, pink line, or Scamp, blue line. This means that the complete failure of a whole local area does not affect the inter-connectivity among the others. In a real world scenario a complete failure of a whole data center/local area, could be externally perceived if, for instance, the links that connect it to the exterior go down, effectively precluding the physical network access to such data center. It is important to note that as the X axis is the percentage of nodes dropped globally, the values for this strategy end up at 20%, which corresponds to the complete removal of one local area out of the five we have in this scenario. For the same reason, the measurements in the TwoAreaDrops strategy end up at 40%, which corresponds to the complete removal of two local areas, as expected in this strategy. Finally, the light blue and yellow line correspond to Scamp and CLON respectively, in a TwoAreaDrops strategy. As it is possible to observe, the impact on connectivity of this localized drop strategy continues to not endanger the connectivity of the overlay.

For the two remaining graphics that depict the other graph properties we intend to analyse, we only plot the results obtained from the UniformDrops strategy in order to not clutter them up. Furthermore, the impact of the other strategies is not as relevant to the other metrics as it is for connectivity.

In Figure 5.3, we plotted the evolution of the clustering coefficient in face of the increasing global drop rates for both Clon, green line, and Scamp, red line. It is possible to observe an almost constant value that separates the higher clustering coefficient of Clon from the lower values of Scamp. This difference is easily explained by the goal of Clon itself, which gives preference to local nodes over remote ones and thus, the overlay tends to naturally cluster in order to reflect the clustered topology of the underlying network.

As the reader may remember from the background Section 2.1.2, overlays with high clustering coefficients tend to partition as the coefficient measures the closeness of neighbour relations and high values indicate that the neighbours are highly connected among them, but poorly connected to the exterior. As Clon tends to bias the overlay to a naturally clustered network, it is normal to observe an increase in the clustering coefficient. However, if we observe again Figure 5.2, it is possible to see that the connectivity is almost identical to that of Scamp, which allow us to conclude that the increase in the clustering coefficient is despicable with respect to the impact on connectivity. The other effect of higher clustering coefficients, is the increased redundancy of messages transmitted among neighbours, however to analyze the impact of this, we have to wait for Section 5.3.

The next metric related to the graph properties we analyse is the average path length, which is depicted in Figure 5.4. As it is possible to observe, the average path length increases steadily in both protocols until the 60%-70% rupture point where the overlay becomes disconnected. The discrepancy between Clon and Scamp is again related to the way links are established in the protocols. While in Scamp the probability of having a far away neighbour is the same as having a close one, in Clon it is much probable to have local neighbours and few remote neighbours. If a given node and its immediate neighbours do not have a link to all the other local areas, which is likely, then the average path length increases naturally as not all the local areas are reachable directly for any given node. As we intend to reduce the load imposed on the long-distance links, we will inevitably fall on the latency-bandwidth conundrum, which is reflected by the increase of the average path length and thus latency.

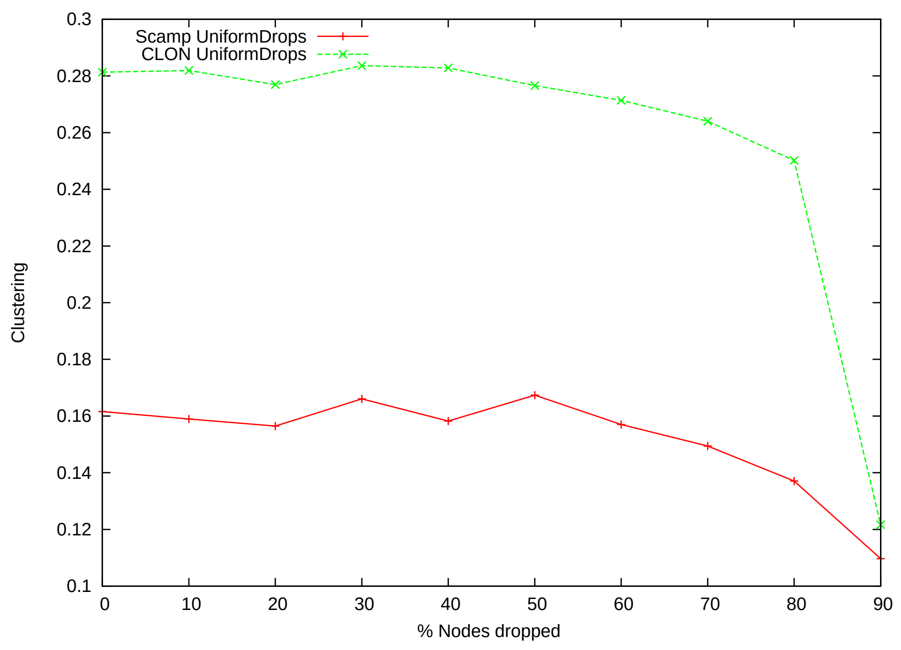

Figure 5.3: Overlay Clustering.

5.2.2 Degree balancing mechanism

In this Subsection we analyse the effectiveness of the degree balancing mechanism by observing its impact on the degree distribution and also on the graph metrics presented in the previous Subsection.

As stated in the protocol description Section 4.2, our proposal introduces a fully decentralized mechanism to balance the degree of the nodes in the overlay, in order to achieve a uniform overlay distribution and produce better quality overlays. We will now assess the quality of the degree balancing algorithm by showing an overlay before the algorithm is run, in Figure 5.5, and the same overlay after a hundred runs of the algorithm, in Figure 5.6.

Although the original Scamp algorithm guarantees that the *average* degree distribution will tend to the right value, the degree distribution in Figure 5.5 shows that the distribution is far from optimal. Considering that the ideal degree in this scenario is 9, it is possible to see that only slightly more than 15% of nodes have that degree, and around 45% of nodes are on the ideal degree value with a deviation of ± 1. Furthermore, a considerable amount of nodes has either very low or very high degrees, for instance some nodes have degrees above 25 which clearly inhibits the reliability and quality of the overlay. This is explained by the age of nodes. As nodes stay in

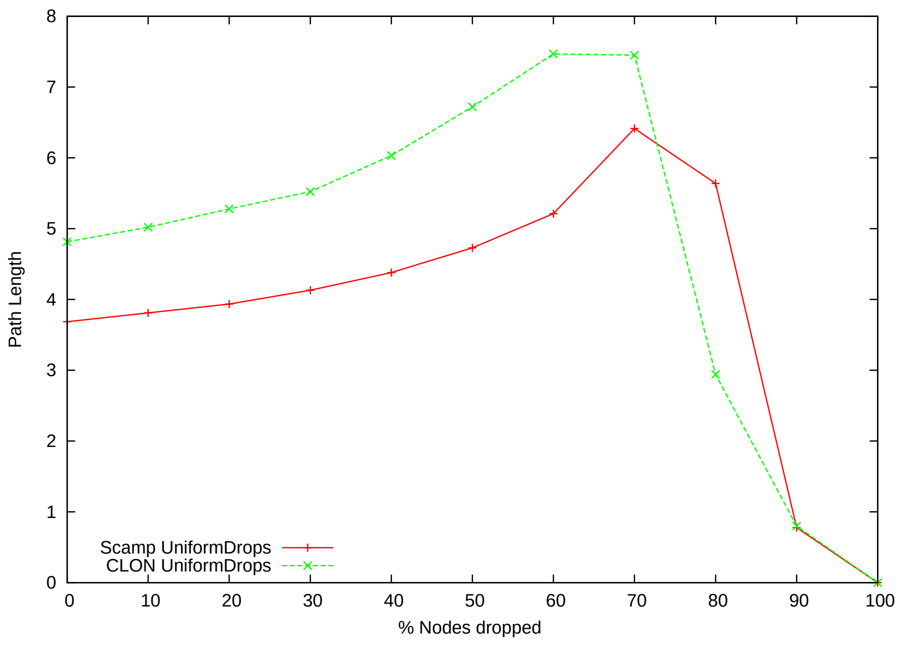

Figure 5.4: Overlay Average Path Length.

the overlay for more and more time, they will receive more and more subscription requests. Despite the probability of a request being integrated decreases with the increase of the node degree (or view size), some subscriptions will eventually get accepted, as the integration function has a probabilistic base, and thus those nodes will tend to have very large degrees. On the other hand, when the membership remains stable, i.e. no joins or leaves, the last nodes that joined the overlay will not receive new subscriptions and therefore their degrees will remain low.

With the degree balancing mechanism we proposed, the overlay effectively evolves by swapping links from nodes whose degrees are high to those whose degrees are low, therefore promoting an even degree distribution. If we observe Figure 5.6 which depicts the same overlay as above it is possible to see that degrees are more even distributed. For instance, more 50% nodes now are on the ideal degree distribution with a ± 1 deviation. Furthermore, the very high degree nodes have been eliminated, although some still have high degrees, and the same applies to the lower degree nodes. By running this optimization for the whole time of the dissemination process we will eventually obtain a narrow degree distribution around the ideal degree and thus contribute to better quality and more reliable overlays, as all nodes will tend to have similar degrees, and therefore the same

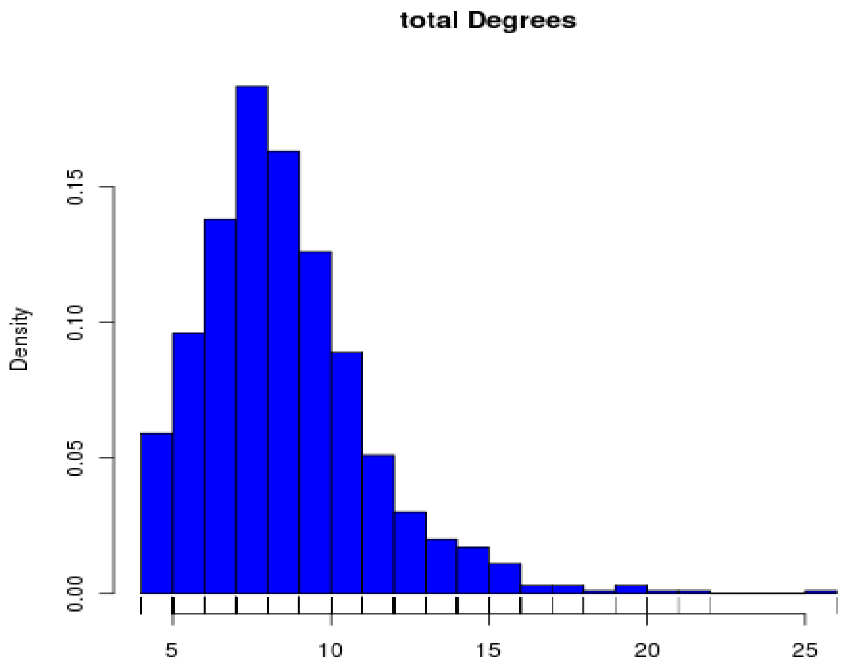

Figure 5.5: CLON Initial Overlay Degree Distribution

contribution to connectivity and to the message dissemination effort. This could be observed in the following experiments, where all the metrics presented tend to improve after the execution of the degree balancing mechanism.

To assess the impact of the degree balancing mechanism with respect to the previous analysed graph metrics, we plot them again after running the degree balancing procedure.

Figure 5.7 depicts the connectivity of the overlay after applying the degree balancing procedure. As it is possible to observe, the procedure effectively improves the connectivity of the overlay, by moving the links in the high degree nodes to the low degree ones. In fact, with this optimization CLON becomes more resilient than Scamp up to 60% and 70% drop rates, reaching nearly 100% of the alive nodes up to 60% global failures. As the impact of the optimization with respect to the connectivity for the other two drop strategies is negligible we do not plot them.

In Figure 5.8 it is possible to observe the impact of the degree balancing mechanism in the clustering of the overlay. The clustering of the underlying graph drop from the previous 0.28 of Figure 5.8 to

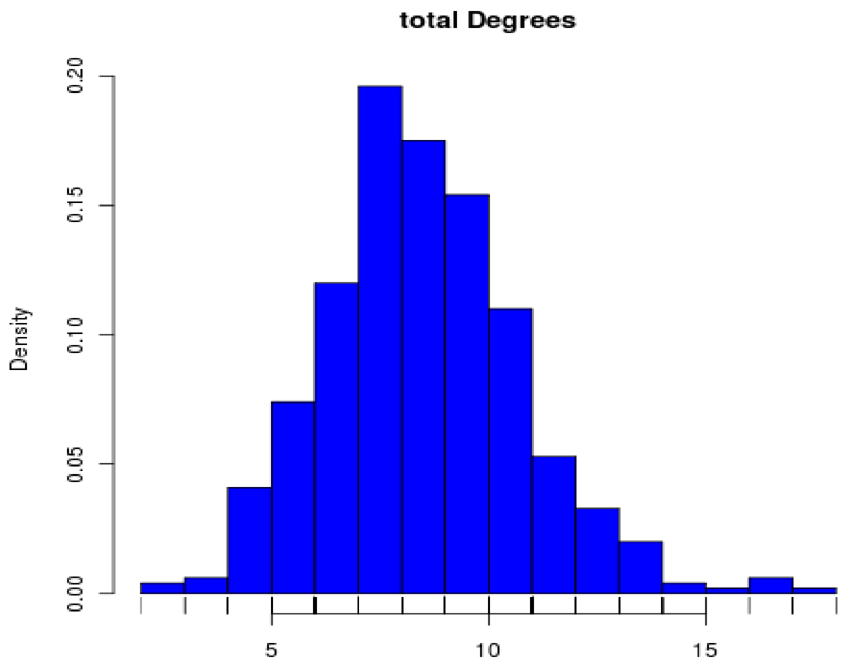

Figure 5.6: Clon Degree Distribution After 100 Runs of the Balancing Algorithm

below 0.22, approximating the values obtainable with Scamp.

The improvement in the path length that the degree balancing procedure brings is depicted in Figure 5.9. As it is possible to observe the exchange of links promoted by the degree balancing mechanism effectively improves the average path length, bringing it to values closer to Scamp.

In summary, and to finalize the evaluation of properties of the overlay built by Clon, we observe that it is possible to achieve the same tolerance to massive amounts of failures as in Scamp, while carefully building the overlay and establishing links among nodes in a way that reflects the underlying network topology. The cost to pay is a slightly increase in the clustering coefficient, due to the fact that the protocol tries to mimic the inherently clustered network topology, and an increase in the average path length, related again to the way links are established among nodes. Apart from those metrics, the overlay balancing mechanism proves to be effective, as it tends to normalize the degree distribution by reducing the degree of high degree nodes and consequently increasing the degree of nodes

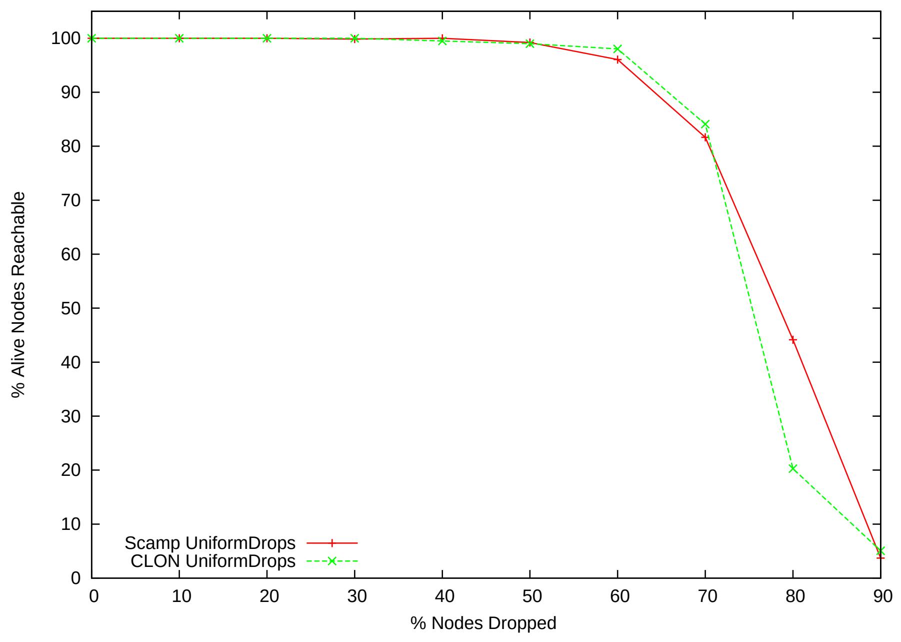

Figure 5.7: Overlay Connectivity After Degree Balancing.

with lower degrees. Furthermore, it improves the connectivity, the clustering coefficient and the average path length of the overlay, to levels closer of Scamp. Insofar, our experimental evaluation shows that CLON is by no means superior to Scamp, with the exception of the degree balancing mechanism. This is expected as Scamp builds a completely uniform overlay, in terms of links established between remote and local neighbours, and CLON disrupts this uniformity by biasing the overlay to take into account locality. However, the work done on the careful establishment of the links starts to give results in the next Section, when we evaluate the impact of the overlay in the dissemination process.

5.2.3 Bootstrapping mechanism

In this Subsection we analyse the proposed bootstrapping mechanism in order to assess if it satisfies the requisite of providing the joining nodes with several contact nodes.

To this end we used different *sendOracle* configurations, starting from a naive one and improving it to obtain the optimal configuration described in Section 4.2.3. The results obtained can be observed in Table 5.2.3. The table is organized as follows: the first two columns describe the configuration of the *sendOracle* and *externalContactOracle*,

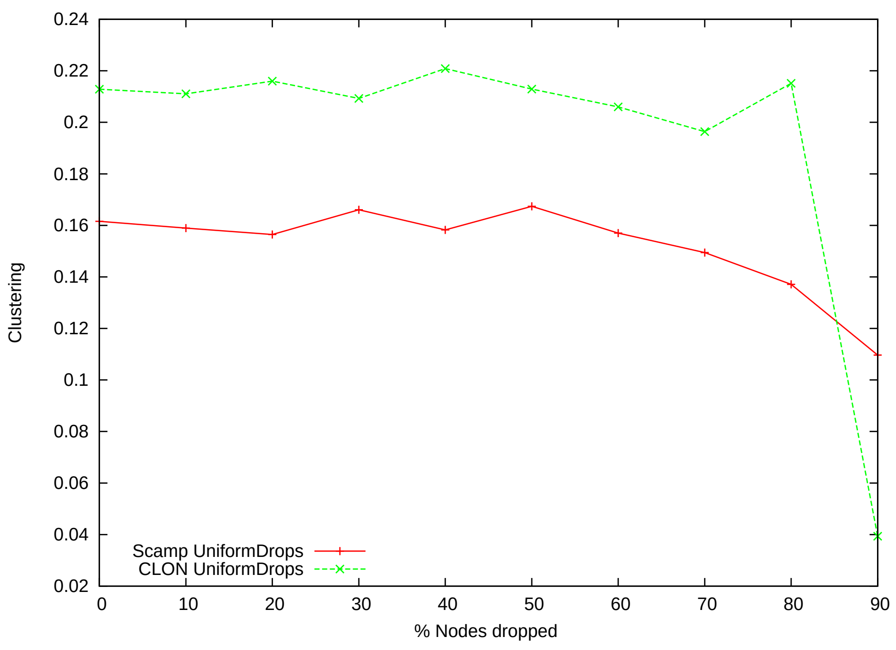

Figure 5.8: Overlay Clustering After Degree Balancing.

respectively; the third column presents the number of messages exchanged by the nodes in the runs of the bootstrapping mechanism, without considering the messages sent by the joiners after receiving the contact; in the fourth column it is possible to observe the total number of replies a given joiner obtained; finally the last three columns show the total, local and remote nodes effectively integrated in the view of the joiner. The run consists of having a new node join a random local area among the 5 available, initiate the bootstrapping protocol, and then extract the relevant measurements. For each configuration we run 5000 join operations and extracted the averages of the results obtained. Furthermore, each new run is independent of the previous, i.e. we run the bootstrapping mechanism for a joining node, and after the process ends, we proceed to the next experiment with a new overlay.

As it is possible to observe for all the configurations the *externalContactOracle* is configured to return true with a probability of 50%, which means that half of the replies will be with local nodes and the other half with remote ones.

On the first row of the table we show a naive configuration of the *sendOracle*, where it is configured to always return true. As such, the bootstrap generates 400 messages, 200 for the initial broadcast procedure and 200 replies to the joiner as each node always reply

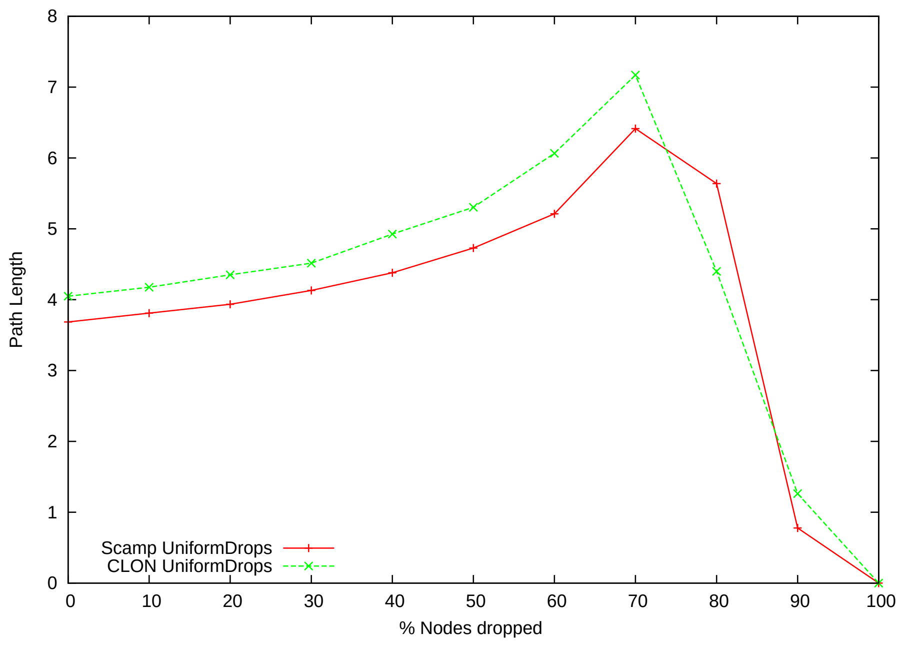

Figure 5.9: Overlay Average Path Length After Degree Balancing.

to the requests in this configuration. With this configuration the joiner obtains on 200 replies, one for each node on its local area, but only integrates 20 on its view. This is due to the probability of integration being restricted by the actual size of the view, as explained in Section 4.2.3. Of this 20 nodes integrated into the view 12 are local and 8 are remote.

On the next configuration, in the second row of the table, the *sendOracle* only replies to the contact requests 5% of the times, which results in sending to the joiner, on average, 10 replies ($0.05 * 200 = 10$), on averge. Of this 10 replies only 5 are effectively integrated into the view of the joiner, of which 3 are local and 2 remote. The configuration of 5% is just a arbitrarily small probability chosen to infer the behaviour of the bootstrapping mechanism.

On the next configuration we start to exploit the local knowledge available in order to approximate the desired optimal behaviour. In this configuration the oracle estimates the total number of nodes in the system, but does not known the number of local areas, and as such the probability of replying is only based on the global number of nodes it estimated. It is important to remember that the size of the view tends to be $c + log(N)$, where N is the number if nodes, and thus it is straightforward to calculate the number of total nodes, and reply with an accordingly probability. Despite not knowing the

Oracles Probabilities		Msgs	Contacts	Nodes Integrated		
sendOracle	externalOracle	Generated	Offered	Total	Local	Remote
1	0.5	400	200	20	12	8
0.05	0.5	210	10	5	3	2
based on global view estimation	0.5	286	86	13	8	5
based on local view estimation	0.5	238	38	9	6	3

Table 5.1: Different bootstrapping configurations.

number of local areas,the result is interesting as it gets closer to the ideal value of 9 links established by the joining node.

Finally, in the last configuration we exploit the knowledge of the previous configuration but assume that the number of local areas is known beforehand. As the number of local areas (or data centers) in our scenario is fairly static, it is reasonable to assume that that value is well known. This configuration corresponds then to the example oracle given in Listing 4.4. With this knowledge available, it is possible to achieve the optimal results in the bootstrapping mechanism. In fact, in this setting the joiner receives 38 contact replies and of those integrates 9 in its view, the value of the average degree of the overlay. Furthermore, the proportion of local and remote nodes is also closely approximated, as our biasing mechanism tends to build views with 7 local nodes and 2 remote ones.

To conclude the analysis of the bootstrapping mechanism, the above experiments show that is possible to achieve near optimal configurations with the local knowledge available at each node, as in the third configuration. Furthermore, if the number of local areas of the federation is known beforehand, it is possible to obtain an optimal result that makes joining nodes indistinguishable from the other nodes already present in the overlay.

5.3 Dissemination Protocol Evaluation

This section has two main objectives: first, analyse the impact of the overlays previously constructed on the dissemination of application level messages, and second, assess the impact of the developed dissemination protocol, based on Emergent. The experiments for both protocols run as follows: each node on the overlay injects a new message on the system, the simulator executes the dissemination protocol and when there are no more messages to be delivered, the data is analysed with the tools mentioned previously. For each experiment, the total, remote and local number of messages transmitted is logged. Furthermore, for the Emergent protocol, the number of total, remote and local advertisements exchanged is also logged. Unless otherwise stated, all the experiments are run on the overlays previously analysed without applying any drop strategy.

With respect to Clon we use two different overlays: one obtained without applying the degree balancing mechanism, and the other after applying the degree balancing mechanism as explained in the previous Section, which is identified as ClonBalance .

5.3.1 Flooding dissemination protocol

In this Subsection we conduct an experiment that consists of a flooding gossip protocol acting in a pure eager fashion. In this protocol, as soon as a new message is received, it is relayed to all known neighbours, following an infect and die model. This protocol is very bandwidth demanding as multiple copies of the same payload are received by each node, through its neighbours. The goal is to access the impact of the peer sampling services used, with respect to the total, remote and locally received messages by each node. The rationale is that if we are able to obtain significant results, i.e. reducing the number of messages that traverse long distance links, the results will be even more interesting with a locality aware dissemination protocol as the one presented in Section 4.3.

Figure 5.10 depicts the results obtained from the above experiment. As it is possible to observe, the number of total messages received, the sum of remote and local messages, by each node is the same in both protocols and is around 9000. This value is easily explained by the overlay characteristics and the dissemination protocol. Each node knows on average 9 neighbours and 1000 different messages are injected on the system, one for each node, thus accounting for the 9000 messages received, on average. However, if we now focus on the messages received remotely, the red bar, we start to see the benefits of using a peer sampling service that takes into account locality. Whereas in Scamp nearly 7000 messages are received remotely, in Clon this value drops slightly below 2000, an improvement of more than three times the value obtained in Scamp. The bulk of messages transmitted in Clon is therefore done locally, for the same reliability level, which effectively demonstrates the impact of a judi-

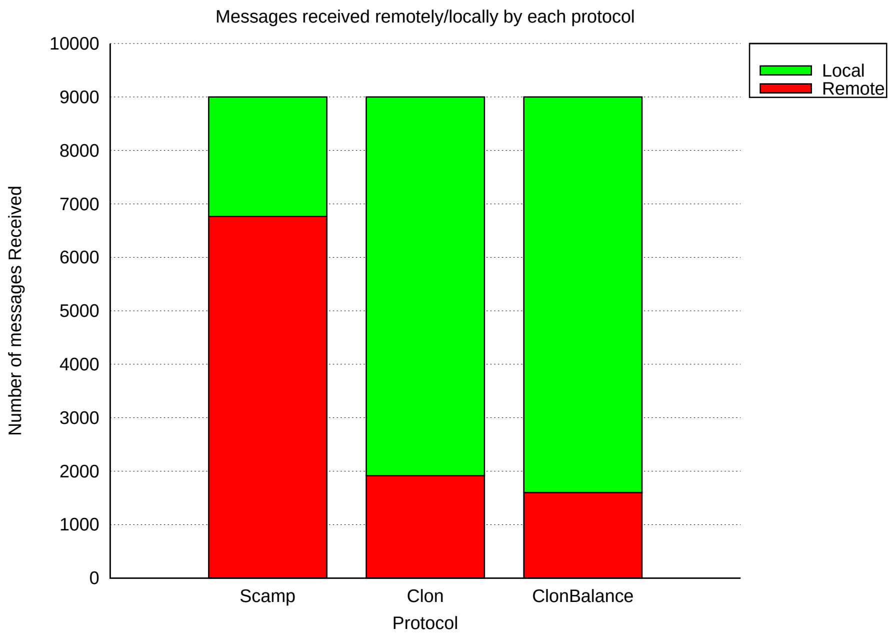

Figure 5.10: Messages received by each node using a flooding dissemination protocol.

ciously built overlay that takes into account the underlying network topology. The results obtained by ClonBalance show a minimal improvement in the number of remotely received messages with respect to Clon. While the difference is minimal to support substantial claims on the improvement bring by the balanced version of Clon, this result attests that the balancing mechanism preserves the biasing of the overlay, while enhancing the graph properties as shown in the previous Section. Nonetheless, if the degree balancing mechanism is run continuously, nodes with high degrees will be eventually eliminated and thus the unnecessary redundancy in messages transmission will be eliminated, further reducing the number of transmissions over the long distance links.

5.3.2 Improved Emergent Dissemination Protocol

In the next experiment, we used the improved version of the Emergent dissemination protocol with a simple policy: relay messages to local nodes using an eager strategy, and use the lazy strategy for all the remote nodes, using the same overlays as in the previous experiment. The results obtained are depicted in Figure 5.11 and we discuss them next.

The impact of using a locality aware dissemination protocol is perhaps the most interesting insight of Figure 5.11. In fact, by using the aforementioned dissemination strategy, the amount of message payloads transmitted over long distance links decreases considerably, both in Scamp and CLON. In Scamp this value dropped from around 7000 messages to slightly above 2000, which is similar to the values obtained solely by using CLON with a flooding dissemination strategy. The improvements of CLON is also considerably, going from around 1900 to about 600 payload transmissions. This results is quite important as it shows that by combining a locality-aware peer sampling service with a locality-aware dissemination protocol, it is possible to reduce the number of message payload transmissions over long distance links by an order of magnitude, when comparing protocols unaware of network locality. This can be observed by the results obtained by a combination of Scamp with a flooding protocol which yields around 7000 message payload transmissions on long distance links with a combination of CLON with the locality aware emergent, which achieves around 600 transmissions for the same dissemination scenario. Nonetheless, there are other interesting results that provides us with insights of the impact of combining the different dissemination and peer sampling protocols. The discrepancy of locally received messages, green bar, in Scamp and CLON could be explained as follows. The dissemination on local nodes uses a pure eager strategy, i.e. flooding, and as such a considerable amount of redundant message payloads will be transmitted in each local area. As in Scamp links are established without taking into account locality, each node knows, on average, more remote neighbours than local ones (remember that we have four times more remote nodes than local ones), and as such the number of locally redundant transmissions on Scamp is much lower than that of CLON. As nodes running CLON known more local nodes than remote ones the redundancy of locally received messages considerable increases. This could be mitigated by using a more meticulous dissemination strategy with respect to local nodes, such as transmitting eagerly for a certain number of rounds when local nodes are likely to not have the message, and then fall back to a lazy strategy to conservatively infect the remaining nodes. However, for the sake of simplicity we have not considered this approach in this scenario, as it is not our main goal. A detailed analysis of possible dissemination strategies can be found in the original Emergent paper [7]. The last result to analyse in this experiment is the number of message advertisements in both proto-

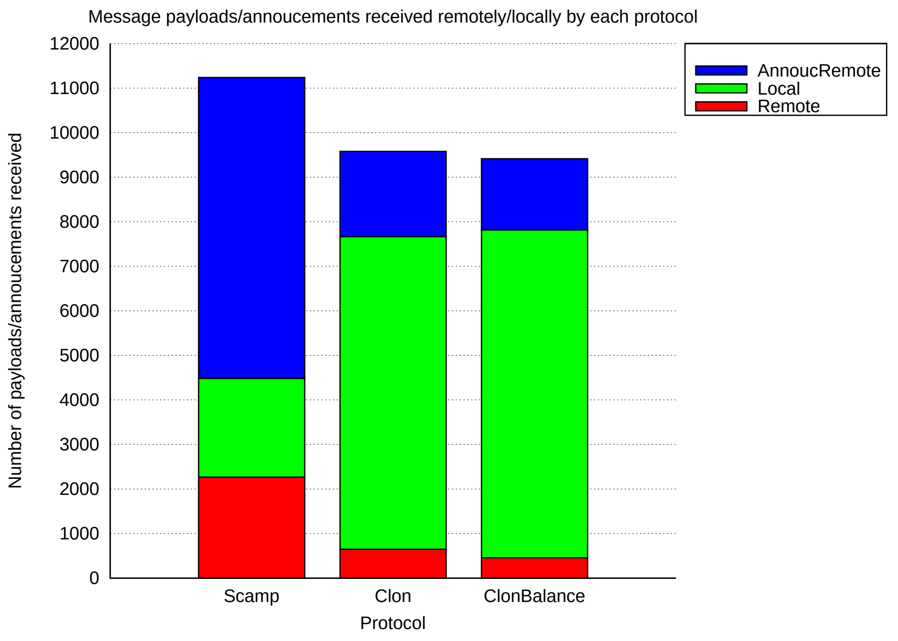

Figure 5.11: Messages/Advertisements Received using the improved Emergent dissemination protocol.

cols. The dissemination strategy used only sends advertisements to remote nodes and as such the blue bars reflects at the same time the total and remotely sent advertisements. The difference between Scamp and Clon is again substantial and draws from the differences in the overlay topologies. As in Scamp most nodes known are remote, a considerable amount of advertisements are sent over the long distance links and consequently the payloads are lazily pushed over those links, which explains the reduction of message payloads transmitted to remote nodes. In Clon, the amount of remote nodes known is smaller and as such the quantity of advertisements sent is smaller than that of Scamp. Once again the results obtained by relying on the balanced overlay ClonBalance show an almost imperceptible improvement as in the flooding dissemination protocol pointed above.

The next experiment, depicted in Figure 5.12 is completely different from the previous ones, and measures the impact in the latency/bandwidth trade off that the Emergent protocol offers. The goal is to observe the impact of the chosen payload transmission strategy (by means of the *isEager* oracle, see Listing 4.6) on the latency and bandwidth consumption of the dissemination process. To this end we run a set of experiments where the *isEager* oracle returns False reach

Figure 5.12: Bandwidth/Latency trade off of the different strategies using the improved Emergent dissemination protocol.

if and only if the target node is external and the external round is below a given threshold, as depicted in Listing 5.1. The rationale is to transmit the message payloads eagerly for a certain number of rounds and then fall back to lazy strategy. In the experiment we varied the TTL value from 0 to 9, and for each value we run the emergent dissemination protocol on top of the overlay built by Clon, without applying the degree balancing strategy. On the X axis it is possible to observe the different TTL used for each run. As such on the leftmost part of the axis we have a completely lazy strategy that becomes gradually eager as we move to the right. On the left Y axis we measure the bandwidth consumption, blue line, with respect to the number of message payloads transmitted over the long distance links. On the right Y axis we measure the latency of the dissemination, green line, in the number of hops necessary to infect all nodes in the overlay.

For instance, in the completely lazy strategy, i.e. when lazy after the round zero, nodes receive on average slightly more than 600 messages through remote links, which confirms the values obtained in Figure 5.11. With this configuration the latency to infect all nodes is 11 hops.

As expected, the bandwidth increases with the eagerness to transmit the payloads, as more redundant messages are sent, while the latency decreases, as messages reach all nodes quicker, without the additional roundtrips of a lazy strategy. It is interesting to notice that in this scenario the latency reaches its minimum after 4 eager rounds, when it becomes close to the overlay diameter. On the other hand, the bandwidth tends to stabilize only around the 7^{th} round. Therefore, in this scenario using a eager strategy for more than four rounds will only waste bandwidth without bringing any improvement on the latency of the dissemination process.

The point where the two lines intersect presents an interesting trade off as it is when the bandwidth required for the dissemination is small, with a moderate latency penalty.

```

proc isEager(i,d,ri,re,p)
  if isExternal(p)
    return re < TTL
  else
    return True
```

Listing 5.1: isEager oracle with a TTL configuration

Chapter 6

Conclusion

One must have a good memory to be able to keep the promises one makes.

Friedrich Nietzsche

In this final Chapter we present the main conclusions drawn for the work done, summarize the contributions this dissertation offers, and give pointers to future research questions.

6.1 Conclusions

By clearly addressing the problem of reliable multicast at two distinct levels: the peer sampling service and the dissemination protocol, we have been able to satisfactorily achieve the requirements presented in Chapter 3, as the extensive experimental evaluation conducted attests. Namely, the proposed set of protocols achieves reliable dissemination of messages to all the correct peers in the presence of massive rates of failures, while adapting to changing system sizes. The resilience of the protocols is assessed by the experimental evaluation conducted in Section 5.2. The advantages of the link differentiation promoted by Clon become evident in the dissemination of the application level messages, as there is a substantial reduction on the load imposed on the long distance links, as it is possible to observe in Section 5.3.

The key to the successful combination of this often adverse objectives relies on the overlay produced by Clon: by biasing the overlay to the network topology fewer remote links are established and therefore the load imposed on them is reduced; and by refusing to rely on special nodes to handle locality as in previous proposals, and using instead a flat unstructured approach, the inherent resilience and scalability of the latter protocols is preserved. Furthermore the natural resilience to churn that unstructured approaches present allows our protocol to cope with the requirement of mitigating the undesirable churn effects.

The continuous balancing of the node degrees proved to be an effective mechanism in the improvement of the overlay properties, which results in a superior overlay than the one obtained with the initial protocol, as shown in Section5.2. The mechanism improves the connectivity, clustering coefficient and average path length of the obtained overlay, approximating them to the values obtained with a protocol oblivious to locality, without disrupting the biasing previously established. In fact, by standardizing the degree of the nodes is is possible to tolerate more failures than a flat locality unaware protocol, such as Scamp.

The bootstrapping mechanism breaks some barriers by providing a more reliable and decentralized, yet not completely, way of providing the joining nodes with initial contact nodes. Using this mechanism joining nodes acquire several contact points to establish the initial links and as such the overall quality of the overlay is increased. Furthermore, it was been shown that by using an appropriate oracle it is possible to establish as many initial links as the average view size, which contributes to the indistinguishability between joining nodes and nodes already on the overlay.

By enabling the locality awareness on the dissemination protocol with the introduction of distinct rounds, and taking advantage of the overlay built by the peer sampling service we have been able to achieve an overall improvement of an order of magnitude on the number of messages that traverse the long distance links, when compared to protocols oblivious to locality.

Finally, it is important to stress the flexibility that the oracles confer to the proposed protocols. Instead of choosing a-priori configurations to each one of the oracles that 'should work on most scenarios', we defer that decision to the programmer who is able to adjust the parameters to his/her particular application environment. Therefore, we not restrict the protocols to the set scenarios we envision, widening its potential applicability to novel, maybe unforeseen settings.

6.2 Summary of Contributions

This Section briefly summarizes the contributions of this dissertation, which are the following:

- Design of a peer sampling service that establishes links among nodes, at construction time, taking into account locality;
- Development of a degree balancing mechanism that further increases the quality of the overlay obtained, without disrupting the locality properties of the overlay;
- Introduction of a decentralized bootstrapping mechanism that offers to the joining nodes several contact points instead of just one;
- Introduction of two distinct round types in the dissemination protocol, to handle separately local and remote nodes;
- Reordering of the queue of pending lazy pushes to give preference to local nodes over remote ones.

6.3 Future Work

After the work done on this thesis, we do believe that many pending and challenging issues still remain in the problem of reliable multicast in very large and dynamic distributed systems. An ambitious research direction will be to study the possibility of applying the knowledge obtained here in applications with different requirements, such as the ones we present below.

The overlays built by CLON mimic the network topology by establishing links with remote and local nodes with different probabilities. Although we have not studied it, it will be interesting to infer whether the current proposal addresses scenarios where remote nodes are at different distances. For example it may be desirable to establish links with a remote data center located in the country with more probability than establishing links with a remote data center in another continent. Possibly this can be achieved by configuring the oracles properly, but a full assessment of this scenarios will definitely widen the applicability scenarios of the presented protocols.

Additionally, it will be interesting to infer the possibility of using the rationale behind the biasing mechanism in order to provide reliable dissemination guarantees only to certain groups of nodes, based on their interests. This will imply a careful study of message filtering protocols and research on the possibility of biasing the overlay to approximate the interest groups. Thus the goal will be to reduce or even eliminate the number of messages that reach peers that are not particularly interested in them.

The developed set of protocols only guarantee the reliable dissemination of messages to peers. However, in certain scenarios this is not sufficient, as the application may require total ordering of the received messages. Inferring whether or not the proposed set of protocols is suitable, as a starting point, to provide total order guarantees will be surely a challenging and interesting research direction.

Bibliography

[1] gnuplot. http://www.gnuplot.info.

[2] DC2MS: Dependable Cloud Computing Management Services. http://gsd.di.uminho.pt/projects/projects/DC2MS, 2008.

[3] M. Al-Fares, A. Loukissas, and A.Vahdat. A scalable, commodity data center network architecture. *SIGCOMM Computer Communication Review*, 38(4):63–74, 2008.

[4] Amazon.com, Inc. Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2, 2009.

[5] N. Bailey. *The Mathematical Theory of Infectious Diseases and its Applications*. Hafner Press, second edition edition, 1975.

[6] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, M. Mihai, and Y. Minsky. Bimodal multicast. *ACM Transactions on Computer Systems.*, 17(2):41–88, 1999.

[7] N. Carvalho, J. Pereira, R. Oliveira, and L. Rodrigues. Emergent structure in unstructured epidemic multicast. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 481–490, Washington, DC, USA, 2007. IEEE Computer Society.

[8] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20:2002, 2002.

[9] M. Castro, M. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 1510–1520, 2003.

[10] P. Eugster and R. Guerraoui. Hierarchical probabilistic multicast. Technical Report LPD-REPORT-2001-005, Ecole Polytechnique Fédérale de Lausanne, 2001.

[11] P. Eugster, R. Guerraoui, S. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. *ACM Transactions on Computer Systems*, 21(4):341–374, 2003.

[12] P. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié. From epidemics to distributed computing. *IEEE Computer*, 37(5):60–67, May 2004.

[13] Y. Fang and D. Neufeld. The pendulum swings back: individual acceptance of re-centralized application platforms. *SIGMIS Database*, 37(2-3):33–41, 2006.

[14] R. Foundation. The r project for statistical computing. http://www.r-project.org/.

[15] A. Ganesh, A.-M. Kermarrec, and L. Massoulié. Scamp: Peer-to-peer lightweight membership service for large-scale group communication. In *Networked Group Communication*, pages 44–55, 2001.

[16] A. Ganesh, A.-M. Kermarrec, and L. Massoulié. Hiscamp: self-organizing hierarchical membership protocol. In *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pages 133–139. ACM, 2002.

[17] C. Gkantsidis, M. Mihail, , and A. Saberi. Random walks in peer-to-peer networks: algorithms and evaluation. *Performance Evaluation In P2P Computing Systems*, 63(3):241–263, 2006.

[18] Google. App Engine. http://code.google.com/appengine, 2009.

[19] J. Jannotti, D. Gifford, K. Johnson, F. Kaashoek, and J. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Usenix OSDI Symposium 2000*, pages 197–212, October 2000.

[20] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, pages 79–98, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[21] B. Kaldehofe. Buffer management in probabilistic peer-to-peer communication protocols. In *Proceedings of the 22nd International Symposium on Reliable Distributed Systems*, pages 76–85, Oct. 2003.

[22] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 565, Washington, DC, USA, 2000. IEEE Computer Society.

[23] A.-M. Kermarrec, L. Massoulié, and A. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14:248–258, 2001.

[24] J. Leitão, J. Pereira, and L. Rodrigues. Hyparview: A membership protocol for reliable gossip-based broadcast. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 419–428. IEEE Computer Society, 2007.

[25] M. Lin and K. Marzullo. Directional gossip: Gossip in a wide area network. In *Proceedings of Third European Dependable Computing Conference*, volume 1667 of *Lecture Notes in Computer Science*, pages 364–379. Springer, 1999.

[26] F. Makikawa, T. Matsuo, T. Tsuchiya, and T. Kikuno. Constructing overlay networks with low link costs and short paths. *Sixth IEEE International Symposium on Network Computing and Applications*, pages 299–304, July 2007.

[27] L. Massoulié, A.-M. Kermarrec, and A. Ganesh. Network awareness and failure resilience in self-organising overlay networks. In *Proceedings of the 22nd Symposium on Reliable Distributed Systems*, pages 47–55, 2003.

[28] M. Matos, J. Pereira, and R. Oliveira. Self tuning with self confidence. In *"Fast Abstract", Supplement of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2008.

[29] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of ACM*, 27(2):228–234, 1980.

[30] J. Pereira and R. Oliveira. Rewriting 'the turtle and the hare': Sleeping to get there faster. In *First Workshop on Hot Topics in System Dependability*, 2005.

[31] J. Pereira, L. Rodrigues, and R. Oliveira. Semantically reliable multicast: Definition, implementation, and performance evaluation. *IEEE Transactions on Computers*, 52(2):150–165, 2003.

[32] J. Pereira, L. Rodrigues, R. Oliveira, and A.-M. Kermarrec. Neem: Network-friendly epidemic multicast. In *Proceedings of the 22nd Symposium on Reliable Distributed Systems*, pages 15–24. IEEE, 2003.

[33] Python Software Foundation. Python programming language. http://python.org, 1990-2009.

[34] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172, New York, NY, USA, 2001. ACM.

[35] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Proceedings of the Third International COST264 Workshop on Networked Group Communication*, pages 14–29, London, UK, 2001. Springer-Verlag.

[36] R. V. Renesse, K. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2):164–206, May 2003.

[37] L. Rodrigues, S. Handurukande, J. Pereira, R. Guerraoui, and A.-M. Kermarrec. Adaptive gossip-based broadcast. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 47–56, 2003.

[38] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Lecture Notes in Computer Science*, volume 2218, pages 329–350, 2001.

[39] salesforce.com, inc. http://www.salesforce.com, 2000 - 2009.

[40] A. Stavrou, D. Rubenstein, and S. Sahu. A lightweight, robust p2p system to handle flash crowds. In *IEEE Journal on Selected Areas in Communications*, pages 6–17, 2002.

[41] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Networking Transactions*, 11(1):17–32, 2003.

[42] S. Voulgaris, D. Gavidia, and M. Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, June 2005.

[43] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, apr 2001.

[44] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 11–20. ACM Press, 2001.