Project Number 001907

DELIS
Dynamically Evolving, Large-scale Information Systems

Integrated Project
Member of the FET Proactive Initiative Complex Systems

DELIS-TR-469

Publish/subscribe in Unstructured Peer-to-Peer Systems

R. Baldoni, L. Querzoni,
S. Tucci Piergiovanni, A. Virgillito,
F. Morabito, F. Davide, G. Cortese

2006
Publish/subscribe for Unstructured Peer-to-Peer Systems - Deliverable 2

R. Baldoni, L. Querzoni, S. Tucci Piergiovanni and A. Virgillito
Dipartimento di Informatica e Sistemistica - Università di Roma “La Sapienza”
and G. Cortese and F. Davide and F. Morabito
Radiolabs

Contents

1 Introduction 2

2 Connectivity Maintenance in Dynamic Systems 3
   2.1 View Exchange-Based Overlay Maintenance Protocols 4
      2.1.1 Cyclon 5
      2.1.2 ADH 6
   2.2 Overlay Robustness Under Continuous Churn 7
      2.2.1 Simulation Settings and Measured Metrics 8
      2.2.2 Evaluation of OMPs 9
   2.3 Connection Recovery 10
      2.3.1 Evaluation 11

3 Event Distribution List Accuracy in Dynamic Systems 14
   3.1 Highly Dynamic Distributed Systems 15
      3.1.1 Dynamic Distributed System 15
      3.1.2 Unstructured Peer-to-peer Systems 15
   3.2 Events Distribution Lists for Publish/subscribe Systems 16
      3.2.1 Completeness and Accuracy of EDLs 16
      3.2.2 EDL Accuracy in a Dynamic Environment 17
   3.3 Evaluation 18
1 Introduction

The advent of large-scale distributed applications based on the peer-to-peer communication model, posed to system designers new problems. These problems are strongly connected to the dynamic behaviour of the environment in which such applications are run. Overlay management protocols have been introduced to guarantee peer-to-peer systems connectivity in dynamic large-scale scenarios.

Some of these protocols have been specifically designed to avoid the partitioning of the overlay network in large clusters (network breakage) despite massive node failures and the continuous arrivals/departures of nodes (churn). In the first section of this deliverable we identify a second effect connected to churn, namely network erosion. We show how erosion affects overlay network connectivity and point out that even a strongly connected overlay network, when exposed to continuous churn, can be disgregated in a relatively short time. Therefore we propose a connection recovery mechanism to be endowed at each node which is able to collaboratively detect node isolation and the presence of small clusters.

The dynamic behavior shown by these systems, poses problems even at higher applicative layers, that inhibit the usage of many techniques developed for quasi-static distributed systems. In the second section of this deliverable we analyze the behavior of a simple subscription-flooding based algorithm for publish/subscribe in a highly dynamic environment. Specifically we define the problem of keeping a list of processes matching a given event (Event Distribution List) complete and accurate. We propose a simple variant of the subscription flooding approach, introducing expiration of subscriptions and their periodic refresh, and show how it can maintain accurate and complete EDLs in a dynamic distributed system.
2 Connectivity Maintenance in Dynamic Systems

In the last decade the advent of peer-to-peer (p2p) computing introduced a new model of distributed computation where (i) the scale of the system can be very large, comprising up to millions of users (peers), (ii) each peer acts independently from all the others, actually precluding any form of centralized network-wide administration or management, (iii) each peer acts as a client of the service and cooperates with other peers to enable services for other participants, and (iv) the system, due to its size and the autonomy of each peer, is intrinsically dynamic as peers can join in or leave at any time.

In this context the basic problem that must be solved in order to build distributed applications, like publish/subscribe systems, is how to guarantee connectivity among participants. Connectivity is, in fact, the basic building block to enable network communications among peers. Modern p2p systems use, to this aim, an overlay network, i.e. a logical network connecting all the participants, whose maintenance is demanded to a specific protocol, namely an Overlay Management Protocol (OMP).

When p2p systems grow up to very large scales, phenomena connected to the dynamic behaviour of nodes gain importance: the continuous arrival and departure of nodes, usually known as churn, can cause, if not properly addressed, overlay network partitioning.

OMPs for unstructured p2p systems, based on gossip approaches [1, 2, 3, 4] revealed to be very effective in the prevention of major network breakages, i.e. the partitioning of the overlay network in two (or more) clusters of approximately the same size. Overlay network breakages can be considered as catastrophic events that affect the system with a large and abrupt reduction of the overlay network connectivity. These protocols aim at building and maintaining, through some lightweight mechanisms, an overlay network with a random topology; the random topology is used to guarantee a low probability of major overlay network breakage even when a very large portion of nodes is abruptly removed.

However, overlay network partitioning can also take the form of a second distinct effect (beside network breakage): network erosion. Network erosion is a phenomenon, caused by churn, concerning progressive isolation of single nodes or tiny clusters that lose connectivity with the main cluster of the overlay network.

Many OMPs underestimate this problem simply assuming that an isolated node, or nodes being part of tiny clusters, could eventually re-join the overlay [3] but without employing specific mechanisms. In this section we point out that fighting network erosion deserves the same attention as network breakages.

Erosion can be indeed so disruptive that a strongly connected overlay network exposed to continuous churn can be quickly disgregated. More specifically we show through an experimental study how badly network erosion affects view-exchange based overlay management protocols, like Cyclon [4] and ADH [3]. Overlay networks built through these protocols are, in fact, progressively eroded as long as the churn period lasts enough. Our study thus confirms that these OMPs were not designed to take erosion into account, and are thus not
able to face its effects.

To fight erosion we propose a connection recovery mechanism, whose goal is twofold: (i) increase robustness of the overlay network during long periods characterized by churn and (ii) recover connectivity during periods of stability. To reach these goals we exploit a node re-join method that locally detects a status isolation from the network’s main cluster. The presence of small clusters is also recognized and addressed by leveraging collaboration among nodes. Our experimental studies show how the connectivity recovery mechanism is able to reduce the effects of network erosion during long periods of time characterized by churn, improving the capacity of the OMPs to quickly react to topology changes; the experiments also show that, through this mechanism, the overlay network is able to quickly regain full connectivity when the system undergoes a stability period without churn.

The effects of network erosion have never emerged clearly in other works on OMPs for unstructured p2p systems [3, 4, 1, 2]. Voulgaris et al. in [4] and Ganesh et al. in [2] analyzed protocols’ behaviour only in a static setting without churn to check the ability of these OMPs to resist to massive node failures. Allavena et al. in [3] and Eugster et al. in [1] addressed the overlay network partition problem considering prevention and recovery from large network breakages. Specifically, the former provides a completely decentralized solution while the latter assumes the presence of a set of fixed nodes in the system. Nevertheless, both solutions have not been experimentally evaluated, thus the effects of overlay network erosion did not come out. Finally, the first analysis of the behaviour of two different OMPs, namely SCAMP [2] and Cyclon [4], under continuous churn has been presented in [5]. In that work some problems related to continuous churn have been pointed out, but the paper did not provide any solution to address them.

The remainder of this section is organized as follows. Section 2.1 introduces the reader to the details of two OMPs: Cyclon and ADH. Section 2.2 shows how overlays built and maintained through these two OMPs are affected by continuous churn. Section 2.3 introduces the connection recovery mechanism, while Section 2.3.1 shows its effectiveness through an experimental study.

2.1 View Exchange-Based Overlay Maintenance Protocols

An overlay network is a logical network built on top of a physical one (usually the Internet), by connecting a set of nodes through some links. A distributed algorithm running on nodes, known as the Overlay Maintenance Protocol (OMP), takes care of the overlay “healthiness” managing these logical links. The common characteristic of all OMPs is that each node maintains links to other nodes in the system. This set of links is limited in its size in order to favour system scalability and it is usually known as the view of the node. The construction and maintenance of the views should be such that the graph, obtained by interpreting links in views as arcs and nodes as vertexes, is connected, as this is a necessary condition to enable communication from each node to all the others. View maintenance can be realized through two main approaches: structured protocols use deterministic algorithms to update views’ content, while unstructured ones usually rely on probabilistic solutions. OMPs employing the
latter approach aims at building overlay topologies that closely resemble random graphs, and share with them nice properties like high connectivity and low network diameter. Thanks to these properties OMPs for unstructured p2p systems are usually considered as best candidates for highly dynamic settings like the ones we consider in this work.

OMPs for unstructured p2p systems differentiate among themselves with respect to the technique they employ to build and maintain views. They can be divided in two broad groups basing on the strategy used to manage node leaves:

Reactive Protocols - require each node to execute some algorithm before leaving the system [2].

Proactive Protocols - continuously adjust the network topology in order to allow nodes to leave without executing any specific algorithm [4, 3].

In this deliverable we focus the analysis of two proactive protocols, namely Cyclon [4] and ADH [3], due to the fact that there is nowadays common agreement in considering proactive protocols more suited than reactive ones to dynamic environments. ADH and Cyclon are both based on a technique known as view exchange that requires nodes to continuously exchange part of their views in order to keep the overlay topology as close as possible to a random graph. Random graphs are characterized by strong connectivity, a property that is exploited to avoid network partitioning: node departures or faults can, in fact, be simply ignored by the OMP as the random topology is supposed to remain connected despite node removals.

2.1.1 Cyclon

Cyclon [4] follows a proactive approach where nodes perform a continuous periodic view exchange activity with their neighbours in the overlay. The view exchange phase (named in this case “shuffle cycle”) aims at randomly mixing views between neighbour nodes. Joins are managed in a reactive manner, through a join procedure, while voluntary departures of nodes are handled like failures (no leave algorithm is provided). A simple failure detection mechanism is provided in order to clean views from failed nodes.

Data Structures and Parameters - Each node maintains only a single view of nodes it can exchange data with. The size of the view is fixed and can be set arbitrarily. Each node in the view is associated to a local age, indicating the number of shuffle cycles during which the node was present in the view. A predefined parameter $l$ defines the number of links exchanged during each view shuffle.

Join Algorithm - A node $A$ joins the overlay network starting from a node (bootstrap node) among those already present in the network. The protocol starts then a set of independent random walks from the bootstrap node. The number of random walks is equal to the view size, while the number of steps per each random walk is a parameter of the algorithm. When a random walk terminates, the last visited node, say $B$, adds $A$ to its view by replacing one node, say $C$, which is added to $A$’s view using an empty slot.

Shuffle Algorithm - The shuffle algorithm is executed periodically at each node. A shuffle cycle is composed of three phases. In the first phase a node $A$, after increasing the age of all the nodes in its view, chooses its shuffle target,
$B$, as the node with higher age among those in its view. Then, $A$ sends to $B$ a shuffle message containing $l-1$ nodes randomly chosen in $A$’s view, plus $A$ itself. In the second phase, when $B$ receives the shuffle message from $A$, it replaces $l-1$ nodes in its view (chosen at random) with the $l$ nodes received from $A$ and send them back to $A$. In the final phase $A$ replaces the nodes previously sent to $B$ with those received from it. Overall, the result of one shuffle cycle is an exchange of $l$ links between $A$ and $B$. The link previously connecting $A$ to $B$ is also reversed after the shuffle.

**Handling Concurrency** - In the specifications given in [4], no action was defined in the scenario of two (or more) concurrent shuffle cycles, e.g. when a node $A$, during a shuffle cycle in progress with $B$, is selected as a target node by receiving a shuffle message from $C$. If concurrency is considered, the nodes sent by $A$ to $B$ can be modified by the concurrent shuffle involving $A$ and $C$.

To analyze the behaviour of Cyclon in concurrent scenarios we extended the original specification in order to address this situation: when nodes that should be replaced by $A$ are no longer present in its cache, $A$ replaces some nodes chosen at random.

### 2.1.2 ADH

ADH employs a slightly different strategy to maintain views. Each node periodically substitutes its whole view with a new one, which is built basing on information collected since the last view exchange. Even in this case joins are managed in a reactive manner, through a join procedure, while voluntary departures of nodes are handled like failures. Failure detection techniques are not used because crashed nodes are automatically discarded by the view exchange algorithm as time passes by.

**Data Structures and Parameters** - Like Cyclon, also ADH employs a single view for each node. The size of the view $k$ is fixed and can be set arbitrarily. Two more parameters are considered: the fanout $f$ and the weight of reinforcement $w$. Both are detailed later in this section.

**Join Algorithm** - Nodes joining the overlay network fill their initially empty views with the view of one of the nodes already in the system. ADH does not prescribe any specific method to choose this bootstrap node, as the OMP should be always able to balance the network approximating a random topology.

**View Exchange Algorithm** - Each node updates its view periodically, at the end of every round. During a round each node collects:

- a list $L_1$ comprising the local views of $f$ nodes chosen at random from its view;
- a list $L_2$ comprising those nodes that requested its view during the round.

At the end of each round these two lists are used to create the local view that will be used in the next round. The new view is built by choosing $k$ nodes from both $L_1$ and $L_2$. The weight of reinforcement $w$ ($w \in [0, \infty]$) is used to decide from which list a node must be picked: if $w = 0$ then all nodes are selected

---

Footnote: In [3], the protocol is introduced in a synchronous environment where the notion of round is clearly defined. In an asynchronous setting, like the one we used to test the algorithm, the notion of round can be approximated with the time lapse between two consequent view exchange operations. In our setting rounds pertaining to different nodes are not synchronized.
in $L_1$, if $w = 1$ nodes are selected with equal probability in $L_1$ and $L_2$, and, finally, if $w = \infty$ then all nodes are selected in $L_2$. This mechanism is used to keep the network “clean” of crashed nodes (that surely will not appear in $L_2$), while mixing views. For these reasons the authors of [3], with respect to the value of $w$, suggest “Larger is better and will be either 1 or $\infty$ on a typical implementation”.

2.2 Overlay Robustness Under Continuous Churn

The protocols presented in Section 2.1 are able to build overlay networks whose topology approximates a random graph [4, 3]. Thanks to this property, systems built with these OMPs are supposed to be highly resilient to node removals. In a dynamic p2p scenario participants can enter or leave the system at their will, at any time. The global rate at which these actions occur is called the churn rate. Node removals happen continuously during some time periods, i.e. churn is not an instantaneous phenomenon but its effects are rather durable in time. During time periods characterized by sustained churn rates the time available to the OMP to repair the overlay network after a node departure, substituting dangling edges with valid ones, can become too short. Continuous churn can in these cases lead to lose overlay network connectivity an event that manifests itself in two ways: major network breakages and network erosion.

Network breakages are caused by the removal of one or more nodes which form the common frontier of two otherwise independent large clusters. When these nodes are removed no valid link exists that connects two clusters, thus nodes pertaining to distinct clusters cannot communicate. Figure 1 shows on its left side an example of a major network breakage. The net effect of a large network breakage is an abrupt and dramatic diminishment of the overlay network connectivity. OMPs based on gossip approaches like Cyclon and ADH revealed to be very effective in the prevention of such events. The random graphs they build and maintain, in fact, guarantee that, even when a very large number of nodes are removed, the probability of a major network breakage is extremely low. This result is clearly stated in [4], where it is actually tested only in static scenarios where nodes are removed all at once, and in [3].

Network erosion is a subtler phenomenon, endemic in systems affected by continuous churn, which progressively detaches single nodes or tiny clusters from the frontier of the main network cluster. This frontier is constituted by those nodes whose neighbors have been progressively removed, and whose views contain dangling edges. These nodes are indeed weakly connected to the main cluster and further neighbor removals can quickly bring them to a state of complete isolation. An example of progressive network erosion is depicted on the right side of Figure 1. It’s important to note that erosion, contrarily to network breakages, is a phenomenon which affects progressively the overlay network, but whose effects are nevertheless dramatic. If not properly addressed erosion can, in fact, lead to the disgregation of the overlay network, quickly reducing a strongly connected cluster to a “dust” of isolated nodes.

In the next sections we will show through the results of an experimental study how much network erosion can affect connectivity in an overlay network built and managed by the two OMPs previously introduced, but before delving into the results, let us introduce the environment used to conduct our tests.
2.2.1 Simulation Settings and Measured Metrics

Our test were aimed at the evaluation of two metrics:

**Churn Rate** - With *Churn Rate* $C$ we identify the global rate at which join and leave operations occur. In particular at each time unit, $C$ new nodes invoke the join operation and $C$ nodes in the overlay invoke the leave operation\(^2\).

**Reachability** - With *Reachability* $R$ we identify the average percentage of nodes that can be reached from any node in the overlay.

It’s important to note that $R$ is strictly related to the connectivity of the overlay network, as any value lower than 100% indicates that at least one node cannot be reached by at least one other node.

\(^2\)This rather simple churn model, brought from \([3]\), was chosen to maintain the system at a constant size during tests.
To analyze the network erosion effect and how OMPs behave when a sustained churn rate is present, we implemented and tested in a simulated environment (provided by Peersim [6]) both Cyclon and ADH. A run of each protocol was divided in three distinct periods: creation, churn and stability. During the creation period nodes join the system until a predefined network size $N$ is reached. Neither leaves nor overlapped join operations occur during this phase. During the churn period, nodes continuously join and leave the network at a given churn rate $C$. Leaving nodes are chosen uniformly at random from the network population. The churn period ends after 7000 time units. At the end of this period the number of nodes in the overlay network is still $N$, while the total number of nodes that joined/left the system depends on the specific churn rate $C$. $N$ was set to 1000 in all experiments. Message transmission delays vary uniformly at random between 1 and 10 time units. 10 independent runs were made for each experiment.

### 2.2.2 Evaluation of OMPs

Figures 2(a) and 2(b) report curves showing the evolutions of $R$’s value as time passes by for Cyclon and ADH respectively. Different curves in the same graph represent different churn rates ($C=2,4,8$).

The curves show that reachability is strongly affected during the churn period. At the beginning the curves undergo a steep descending slope that is

---

Further experiments, not reported here, show that, with the considered churn model, the total number $N$ of nodes in the system does not influence the final results as long as the ratio $C/N$ is kept constant.
mainly due to the join of new nodes that are immediately considered as part of the system even if their views are initially empty; these nodes will affect negatively $R$ until their join procedures end filling their views. During the churn period network erosion continuously affects the overlay network isolating single nodes: this effect causes the continuous diminishment of $R$. It is important to note that both Cyclon and ADH show in these graphs the same behaviour even if with distinct absolute values.

At the end of the churn period the system regains a small percentage of reachability: this is mainly due to join procedures that end correctly in the first part of the stability period. The interesting point is that, nevertheless, both OMPs are not able to regain full connectivity after the churn period ends: after a short period of time, used by the OMP to “heal” what remains of the original network, the system stabilizes to a constant $R$ value that is always below 100%. This problem can be imputed to isolated nodes, or to those residing in tiny clusters, whose messages are unable to reach many destinations.

Figures 2(c) and 2(d) confirm this result showing the evolution of node clustering for experiments conducted with $C = 4$. The curves represent percentage of nodes pertaining to the largest cluster (dark grey area), to clusters smaller than 6% of the whole network (light grey area), and isolated nodes (white area). These curves show that, even if ADH is more robust than Cyclon with respect to the effects of churn, the continuous arrival/departure of nodes negatively affects the main overlay cluster greatly reducing its size (with a consequent impact on reachability). From this point of view is fair to say that the analyzed OMPs are actually able to avoid large network breakages (we did not detect any massive network breakage during our tests), but node isolation, due to progressive network erosion, occurs very frequently.

2.3 Connection Recovery

As we showed in the previous section, node isolation occurs quite frequently when the overlay network experiences erosion due to continuous churn. The authors of [4] and [3] did not address explicitly this problem, but without any intervention isolated nodes will remain endlessly in this state.

In this section we introduce a connection recovery mechanism that can be added to both OMPs (and, generally speaking, to every OMP for unstructured overlay networks). Aim of this mechanism is to let nodes detect their isolation state and act consequently in order to regain connection to the main cluster of the overlay network. Moreover, our mechanism exploits cooperation among nodes to detect the presence of tiny clusters.

The basic idea of the connection recovery mechanism is simple: when a node detects that all the links in its partial view represent dangling edges, it triggers a new join procedure to regain connection to the system. In this way, isolated nodes will eventually re-join the system. Dead link detection is not done through some active mechanism but it is rather an indirect result of failed view exchanges (shuffles) with nodes that left the system.

To treat also nodes pertaining to small clusters (that will not satisfy the condition expressed above), we added a cooperative aspect to the basic mechanism. When a node $n$ tries to re-join the system it is assigned a bootstrap node

\footnote{This assignment can be realized exploiting the same method used for the plain Cyclon}
that is initially pinged to know the amount of links it has in its partial view. If this amount is under a predefined constant $P$ (a parameter of the connection recovery mechanism), $n$ rejects $n'$ as a bootstrap node, asking for a new one. Nevertheless, $n$ puts $n'$ in a low-connection list: as soon as $n$ encounters a node $n^*$ which is able to guarantee a number of links larger than $P$, $n$ warns all the nodes in its low-connection list that such a node exists in the overlay network. Note that $n$ does not inform nodes in the list about the identity of $n^*$, otherwise a local star-like topology would be created around $n^*$. Nevertheless, this sort of “signal” sent to some nodes will be interpreted by them as a clue that they are possibly part of either a small isolated cluster or a loosely connected part of the overlay: in consequence of this fact each node can independently decide to try a re-join even if its partial view is not empty. This decision is taken just looking at the current status of the view: if it contains a number of links still lower than $P$ then the node will try to re-join the system. The parameter $P$ actually influences the speed at which nodes re-join after detecting their isolation status. A possible solution to network partitioning based on connection recovery is also suggested in [3]. In this case the authors propose, to detect the presence of small clusters, a completely local approach where each node just check the variance of nodes in its view: if this variance is very low then the probability of the node being stuck in an isolated small cluster is high. This proposal was not evaluated in [3], is thus hard to compare its effectiveness versus our connection recovery mechanism.

2.3.1 Evaluation

In this section we evaluate the connection recovery mechanism applied to the Cyclon OMP, in the same setting used for the previous test on $C = 4$.

Figure 3 reports the value of $R$ for a network employing our connection recovery mechanism; the same figure reports, for comparison purposes, the values obtained from experiments ran with the plain Cyclon OMP. The curve shows that, thanks to our mechanism, reachability $R$ remains consistently higher with respect to the equivalent without re-joins. These curves point out a duplex effect caused by the connection recovery mechanism: a constant reachability value is
maintained during churn periods and, as soon as a stability period starts, full connectivity is quickly regained, regardless of the $R$’s values previously reached. It is worth noting that reachability values remain almost constant (or raise) as soon as the connection recovery mechanism starts to work at its full potential. The time needed for this to happen is proportional to both $P$ and the view size: the mechanism will, in fact, take up to $\text{viewsize}$ shuffle intervals before starting the re-join for an isolated node.

The same results are confirmed by Figure 4 where the evolution of overlay clustering is shown. The curves highlight how the connection recovery mechanism quickly pushes isolated nodes and tiny clusters to rejoin the system, increasing the size of the main cluster. It is important to note that the amount of isolated nodes shown by this figure is heavily affected by newly joined nodes that are still waiting to complete their join procedures, and thus have empty views.

Two aspects that can not be underestimated are the load, in term of number of join operations, that is introduced from the re-join mechanism and the impact of this mechanism on the node’s view.

We first measured the evolution of the number of join operations induced by connection recovery. Tests were limited to 1500 time units as the algorithm
did not show significant differences in its behaviour for longer tests in the same settings. As Figure 5 shows, the rate of join operations remains almost constant during the churn period, then immediately experiences a peak that is mostly due to the remaining isolated nodes that altogether try to re-join the system. As soon as the number of isolated nodes falls to zero the join rate drops. This actually means that the connection recovery mechanism remains active only for a limited time frame that is mainly linked to the length of the churn period, without causing further overhead when the system is stable.

Finally, we wanted to test if and how the addition of the connection recovery mechanism can alter the behaviour of the original OMP in terms of the type of network topology built. Our mechanism actually only trigger automatically node leave and join procedures, thus we expected no differences on the “quality” of the network built. This idea is confirmed by the curves shown in figure 6 where we report the in-degree distribution of nodes at the end of the simulation, for Cyclon with and without our connection recovery mechanism. The tests for this latter version were conducted in a scenario with \( C = 4 \). The curves clearly show that both implementations are able to build overlay networks with the same in-degree distribution, proving that connection recovery does not alter in any way the fundamental characteristics of the overlay.

Even though these results are encouraging there are still various aspects that deserve further investigation. Our algorithm currently relies on a parameter \( P \) that is considered fixed and predefined: it would be interesting to devise approaches to let nodes independently evaluate at run-time the correct value for this parameter.
3 Event Distribution List Accuracy in Dynamic Systems

In the last decade the advent of peer-to-peer (p2p) computing introduced a new model of distributed computation where:

- the scale of the system is very large, comprising up to million of users (peers);
- each peer acts independently from all the others, actually precluding any form of centralized network-wide administration or management;
- each peer acts as a client of the service and cooperates with other peers to enable services for other participants;
- given the size of the system and the autonomy of each peer, the system is intrinsically dynamic as peers leave at any time while others join in.
- the computation is realized by a continuously running system in which peers can join and leave at any moment.

The problem of selective information diffusion in this context is still an open issue and a hot topic for research. The publish/subscribe communication paradigm seems a promising approach in this sense. A publish/subscribe communication system is made up of processes, called publishers, that produce information in the form of events, and processes, called subscribers, that consume that information. Each subscriber declares its interest in a subset of all the events produced by issuing subscriptions, i.e. filters on events. A further set of processes is considered that cooperate with publishers and subscribers in order to deliver to each subscriber the events that correctly match its subscriptions.

The problem of determining the set of interested subscribers for each published event can be modelled as the problem of building on-the-fly an abstract list, called Event Distribution List (EDL), containing this set of subscribers. It is worth to note that each published event has an associated EDL which can be built only using previously collected subscriptions. Various implementations of the pub/sub communication paradigm differ according to how subscriptions are collected and maintained and how EDLs are built. For example, with the filter-based routing approach used in SIENA, each subscription is maintained in a subset of intermediate nodes, called brokers, and EDLs can be built routing information stored at each broker. The implementation of a publish/subscribe communication system on top of highly dynamic p2p systems is currently an open issue. The intrinsic dynamic behavior of p2p systems is a great obstacle to the building and maintenance of any logical structure, as nodes continuously entering and leaving the system disrupt this structure thus increasing the effort needed to maintain it. The usage of well-known approaches based on data structures that enable selective routing of events is unfeasible, due to the high maintenance overhead in a continuously changing environment. In fact, in such scenario, subscribers can frequently change their subscriptions or abruptly leave the system, leading to EDLs that are populated with subscribers that are either no more interested in those events or no longer in the system. That is,

\footnote{Peers leave the system due to many reasons: explicit actions, selfish behavior due to, for example, unfair shared workload, failures etc.}
dynamics affecting the system along the time causes a continuous loss in EDLs accuracy, i.e. the property of containing only subscribers that are interested in the event the EDL refers to. Event diffusion algorithms that don’t take into account such issues can incur a continuous and infinite degradation of accuracy that can have a severe negative impact on system performance in terms of wasted memory (potentially infinite).

In this section we introduce the problem of building event distribution lists in highly dynamic p2p networks. We consider an unstructured overlay where no dedicated broker exists and each peer (node) can act both as publisher and subscriber. After introducing a model for highly dynamic distributed system (Section 3.1.1), we define Event Distribution Lists, presenting the concepts of completeness and accuracy (Section 3.2). We show how EDLs built via the simple subscription flooding approach [8] incur in a continuous degradation of accuracy in a dynamic scenario, and, for this reason, we propose a simple variant of the same approach, where subscriptions are kept updated by means of a periodic retransmission while pending ones are removed through a simple lease mechanism (Section 3.2.2). Through a preliminary study based on simulations, we show how, in the same dynamic scenario, our variant of the algorithm is able to continuously pursue the optimum in terms of both accuracy and completeness (Section 3.3).

3.1 Highly Dynamic Distributed Systems

In this section we first introduce a model that describes the behavior of nodes in a dynamic scenario, and then show how the issues posed by this model can be partially solved by some unstructured peer-to-peer architectures.

3.1.1 Dynamic Distributed System

The system we consider consists of an infinite set of processes or nodes. Any process may fail by crashing. The system is asynchronous: there is no global clock and there is no timing assumption on process scheduling and message transfer delays. Processes can exchange messages through reliable channels that permit bi-directional communications. In order for a process $i$ to build a communication channel with another process $j$, it suffices that $i$ knows a pointer (e.g. the IP address) to $j$.

The admitted level of concurrency, i.e. the maximum number of processes that may be active simultaneously, is unbounded. This means that at any time $t$ the number of processes that are simultaneously active is finite but can grow without bound. The admitted level of participation is infinite, i.e., infinite processes can eventually become active. Infinite participation implies that there is no assumption on eventual stable components in the system since the set of concurrently active processes may change infinitely often.

3.1.2 Unstructured Peer-to-peer Systems

Since their introduction, peer-to-peer (p2p) technologies were thought as possible solutions to introduce new forms of communication in wide, heterogeneous and dynamic communities. In such systems participating nodes (peers) maintain some local data structures that are used to ensure their connectivity.
The intrinsic dynamic behaviour of the proposed model is a great obstacle to the building and maintenance of any logical structure, as nodes continuously entering and leaving the system disrupt this structure thus increasing the effort needed to maintain it. This disruptive effect can be so strong that, in extremely dynamic settings, even maintaining simple structures (like a connected graph) can become a non trivial problem. From this point of view unstructured p2p systems based on random graphs \cite{9,4}, are considered more suited to such environments than structured ones \cite{10,11,12}. On the other hand, such unstructured p2p systems only offer to the developer weak communication primitives, mainly based on broadcasting techniques. Such systems are built trying to approximate as much as possible a random graph, as this can offer good qualities for broadcast-based communications, like low diameter and high connectivity. Some strategy must then be exploited to maintain such randomness in presence of churn and faults.

In this work we refer to the algorithm introduced in Section \ref{sec:2.1.2} that tries to approximate a random graph through a technique known as view shuffling. Their main drawback of this protocol is inherent to the view shuffling technique it employs: the continuous change of views due to the shuffle mechanism can sometimes inhibit the construction of deterministic structures on top of the p2p layer.

### 3.2 Events Distribution Lists for Publish/subscribe Systems

The main problem in a publish/subscribe system (PS) is how to correctly identify the set of subscribers target for each specific event. From an abstract point of view, this problem is solved by considering, for each event, a list containing pointers (e.g. the IP address or other form of identification) to all the target subscribers. In the following we will refer to this list as the \textit{event distribution list} (EDL). In this section we first define the properties of EDLs and then analyze the problem of building EDLs in a dynamic distributed system.

#### 3.2.1 Completeness and Accuracy of EDLs

Each EDL is characterized by the two following properties:

**Completeness** : the EDL contains all the subscribers target for the event;

**Accuracy** : the EDL does not contain any subscriber that is not interested in the event.

Note that the notion of EDL is only logical, but it is actually present in different forms in any PS, and can be identified as the set of subscribers notified for a specific event. Event distribution through EDLs is based on a two-steps process: first data constituting EDLs is stored in the PS, then, when a new event is published, such data is retrieved and the EDL is built on-the-fly in order to proceed with notifications.

EDL data can be stored with a \textit{centralized} approach (each EDL can be built by exploiting data stored on a single process) or a \textit{decentralized} one (each EDL can be built by aggregating various pieces of data stored on different processes). EDL data can then be retrieved exploiting a \textit{cooperative} mechanism, where
nodes not directly interested in the event can collaborate to the retrieval, or a *non cooperative* one, where only the subscribers and the publisher collaborate to reach this goal.

For example, as for the centralized approach, a cooperative solution is the one presented in [13], where a single node is responsible for storing and matching subscriptions for a specific subset of events (rendezvous-based routing), while an example of non-cooperative solution is *subscription flooding*, where each node hosts a copy of all the subscriptions.

For what concerns the decentralized approach, an example of non-cooperative solution is the event flooding algorithm, where data needed to build EDLs is stored only at subscribers’ side. The classical filter-based approach of SIENA [7] is a cooperative version of this approach: nodes only store a summary of all the subscriptions and can help retrieving EDLs by routing the event in the system.

### 3.2.2 EDL Accuracy in a Dynamic Environment

Not all the above mentioned approaches are equally suited for the dynamic environment introduced in Section 3.1.1. Now we will analyze what problems a dynamic environment poses for each approach and if, or how, they can be addressed.

First of all, let us note that any cooperative approach for EDL retrieval need the imposition of some sort of structure on the overlay network able to provide the minimum information needed to realize collaboration among nodes. As we pointed out in Section 3.1.2, the considered dynamic model is a great obstacle to the building and maintenance of any logical structure on top of the overlay.

For this reason we prefer to consider non cooperative approaches. A correct implementation can be based on *event flooding* (EF) or on the dual *subscription flooding* (SF) approach. With EF each event is broadcast by the publisher in the network and matched on subscribers’ side; this solution leads to a small memory footprint (because every subscription is hosted only on the original subscriber), but a large overhead as every event is sent to every node in the network regardless its actual interest in it. From a theoretical point of view this approach should lead to the retrieval of correct and accurate EDLs.

On the other side, with SF every subscription change is broadcasted by the subscriber in the network; events are matched on the publishers’ side and sent directly only to interested subscribers; this solution dramatically reduces the overhead generated for event diffusion at the price of an higher memory footprint (as every node in the system must keep track of every subscription currently active) and some overhead due to SF. Note that this overhead is exactly equal to the one generated with the EF approach only in an unrealistic scenario where each event matches exactly and only one subscription, whereas in more realistic scenarios each subscription can be matched by thousands or millions of distinct events. Thus the overhead generated by the SF approach is usually various order of magnitude lower, and this is the reason why this approach is often preferable.

The SF approach assumes that each subscription update is sent to all nodes in the system. This behaviour is needed in order to keep subscriptions updated on the whole system, but it becomes a problem when the system itself is dynamic. In fact, in such a system, as shown in Section 3.1.2 nodes can leave
without any warning, abandoning on other nodes their active subscriptions as pending. From this point of view this method cannot provide completely accurate EDLs in a dynamic system. Moreover, as pointed out in section 3.1.1 our system model considers a potentially unbounded number of faulty nodes, and thus the potential quantity of wasted memory space is unbounded as well while the accuracy of the EDLs tends to zero as time passes by (i.e. the fraction of non valid subscriptions in each EDL grows with time and tends to one).

In order to address this problem, we here propose a simple variant of the original SF algorithm, with the following modifications:

• each active subscription is periodically broadcast in the network;

• when a node receives a new subscription from one of its neighbors, it stores along with it the timestamp of its reception;

• when a node receives a known subscription it just reset the associated timestamp;

• each node periodically purges those subscriptions whose age exceeds a predefined timeout constant.

The idea behind this approach (referred to in the following as periodic SF) is that subscriptions are no longer everlasting, but they have an expiration time and must be kept up-to-date by subscribers. Through this time-out mechanism pending subscriptions are easily removed on the whole network in a limited period of time. This garbage collection mechanism based on subscription leases is similar to the one presented in [14]. Note also that this mechanism actually makes unsubscriptions useless, and is thus perfectly suited for all those overlay networks, like [9, 4], that treat node leaves passively. Indeed, a failure detection mechanism is not necessary because subscriptions belonging to failed nodes are not refreshed after their expiration.

On the other hand, the periodic refresh of subscriptions could generate much more network load. However, we believe that, if the rate of periodic refresh is equal to (or lower than) the rate of subscriptions/unsubscription, our approach can still provide better results for accuracy without incurring in a significantly higher overhead. Optimizations are also possible, for example piggybacking several subscription on a single refresh message. A deep analysis of all such aspects is part of future work.

3.3 Evaluation

In this section we present some preliminary results we obtained from an implementation of both the SF and periodic SF algorithms on top of an unstructured P2P overlay network [9]. Aim of this partial evaluation is to show that a dynamic environment greatly affects accuracy and completeness of EDLs managed via the SF approach, through a continuous degradation in the quality of subscription data maintained on nodes.

The tests were conducted in a simulated network environment provided by J-Sim [15]. We simulated an unstructured overlay made up of 1000 nodes for a period of 1200 time units (τ) during which the system is perturbed by the arrival of 1000 new nodes and the departure of all the initial nodes. More specifically, after an initial stability period of 100 τ, the system undergoes a perturbation
period lasting for 1000 $\tau$ in which join/leave operations are periodically executed. After this perturbation period the system stabilizes for other 100 $\tau^{9}$.

The viewsize was set to 5 and the fan-out to 3 nodes.

Each node entering the system issues a single and unique subscription that lasts for its whole lifespan. Subscription maintenance is done through four different algorithms:

- SF without unsubscriptions (SF)
- SF with unsubscriptions (SFu)
- periodic SF without unsubscriptions (PSF)
- periodic SF with unsubscriptions (PSFu)

Protocols with unsubscriptions allow a node to broadcast an unsubscribe message immediately before leaving the system. Protocols without unsubscriptions treat a leave as a fault without allowing any message to be broadcasted before the leave $^{7}$.

Broadcast is implemented as an unreliable primitive: each peer forwards a broadcasted message only once to all the peers in its partial view and discards any further copy of a received message.

Average values of accuracy and completeness were collected periodically during the whole run duration.

Figure 7: Accuracy evaluation

Figure 7 shows the behaviour of the four algorithms with respect to EDL accuracy. The ideal behaviour is represented by a constant value 1. As you can devise from the figure the SF algorithm, without any way to remove pending subscriptions, quickly converges to low values for accuracy. Moreover SF doesn’t employ any mechanism to recover accuracy, thus this low value remains constant even if the perturbation period ends. The only way for SF to increment its accuracy could only be by the addition of new active subscriptions in the system. Adding unsubscriptions greatly changes the magnitude of this behaviour, as

$^{6}$Note that this simulated dynamic environment is not meant to be necessarily realistic, but it was adopted just to introduce some form of perturbation in the system in order to study how the proposed pub/sub protocols behave

$^{7}$Note that this is perfectly consistent with the behaviour described in 

19
reported by the SFu curve. Nevertheless it is very important to remark here that, besides the absolute values, the asymptotical behaviour of SFu is exactly the same of SF, as shown in the detail: correct unsubscriptions can only alleviate the problems caused by pending subscriptions but, given that the hypothetical lifespan of the system is unbounded, and that during its whole lifespan the system is going to incur dynamic behaviour of its nodes, both SF and SFu curves will eventually converge to 0.

A completely different behaviour is shown by PSF. The corresponding curve clearly shows that our algorithm is able to cope with the dynamic environment in which it works, continuously purging pending subscriptions. PSF maintains a quasi-constant (slightly increasing) value of accuracy during periods of perturbation, and quickly regains a perfect accuracy when the system stabilizes. A similar behavior is shown by PSFu, which in turns, exploiting unsubscriptions, can further increase accuracy during the perturbation period. Note that also PSFu, at the end of this period, rapidly returns to a perfect accuracy.

Figure 8: Completeness evaluation

Figure 8 shows the completeness acquainted by the various algorithms during the same tests. Surprisingly SF and SFu algorithms both show an average completeness value that is constantly under the optimum (i.e. 1). This unintuitive behaviour can be justified by the fact that the broadcast primitive used to diffuse subscriptions is not completely reliable as the overlay graph often contains nodes, with a transient empty in-degree, that cannot be reached by messages diffused by other nodes. Even though this peculiar state of few nodes is transient and doesn’t affect the connectivity of the overlay network, it has a non negligible impact on broadcast reliability. Moreover message broadcast can sometimes interfere in view shuffling operations that can actually affect its effectivity. The sum of this two aspects ends up with the fact that not all the subscriptions are received by every node. These problems are obviously present also in PSF and PSFu, that use exactly the same broadcast primitive, but they, thanks to the periodic retransmission of subscriptions, are able to obtain a more reliable dissemination of subscriptions. The neat result is that PSF and PSFu show a nearly optimum correctness.

To sum up the results shown here we can conclude that (i) plain subscription-flooding based algorithms clearly show some problems in dynamic scenarios;
these problems are not transient and can hamper indefinitely system’s performance; at the same time (ii) a simple approach based on subscription lease, can effectively address the problem of maintaining data structures needed to build accurate and complete EDLs.
References


