Virtual Tree: a Robust Architecture for Interval Valid Queries in Dynamic Distributed Systems

Roberto Baldoni[a], Silvia Bonomi[a], Adriano Cerocchi[a,], Leonardo Querzoni[a,*]**

[a]Dipartimento di Ingegneria Informatica, Automatica e Gestionale Antonio Ruberti
Universitá degli Studi di Roma "La Sapienza", Rome, Italy

Abstract

This paper studies the problem of answering aggregation queries, satisfying the interval validity semantics, in a distributed system prone to continuous arrival and departure of participants. The interval validity semantics states that the query answer must be calculated considering contributions of at least all processes that remained in the distributed system for the whole query duration. Satisfying this semantics in systems experiencing unbounded churn is impossible due to the lack of connectivity and path stability between processes. This paper presents a novel architecture, namely Virtual Tree, for building and maintaining a structured overlay network with guaranteed connectivity and path stability in settings characterized by bounded churn rate. The architecture includes a simple query answering algorithm that provides interval valid answers. The overlay network generated by the Virtual Tree architecture is a tree-shaped topology with virtual nodes constituted by clusters of processes and virtual links constituted by multiple communication links connecting processes located in adjacent virtual nodes. We formally prove a bound on the churn rate for interval valid queries in a distributed system where communication latencies are bounded by a constant unknown by processes. Finally, we carry out an extensive experimental evaluation that shows the degree of robustness of the overlay network generated by the virtual tree architecture under different churn rates.

Keywords: dynamic distributed systems, distributed query answering, overlay networks, peer-to-peer systems, node clustering.

*Corresponding author
**Principal corresponding author
Email addresses: baldoni@dis.uniroma1.it (Roberto Baldoni), bonomi@dis.uniroma1.it (Silvia Bonomi), cerocchi@dis.uniroma1.it (Adriano Cerocchi), querzoni@dis.uniroma1.it (Leonardo Querzoni)

A short version of this paper appeared at the 13th International Conference on Distributed Computing and Networking [5].

1. Introduction

Today's large scale distributed systems are characterized by strong dynamics caused by the inherent unreliability of their constituting elements (e.g. asynchrony, process and link failures, processes joining or leaving the system). In particular, the continuous arrival and departure of processes from the network (phenomenon also know as *churn*) has a strong negative impact on algorithms designed to work on such systems. Regular registers [6], replication [14], in-network aggregation[10], are just few examples of application contexts where churn represents a huge challenge. Bawa et al. introduced in their seminal work [9] the problem of defining precise semantics for in-network aggregation query answering (e.g. sum, max, min etc...) in large-scale dynamic systems. As an example, consider a simple query whose purpose is to count the number of processes in the system. Due to the large scale of the system, the execution of a distributed in-network query answering algorithm would take some time to complete. During this time the system population would continuously change due to the effect of churn. What kind of semantics can be assigned to the result? What is the process set over which the result is evaluated?

One of the semantics introduced in [9], namely *Interval Validity* (IV), requires the answer to be evaluated considering *at least contributions from all the processes that remained in the system from the moment the query is issued, until the last answer is collected.* This kind of semantics plays a fundamental role in many applications as it prevents contributions from correct nodes that do not leave the system to be eclipsed by transient errors and failures.

In the same work, the authors also proved the impossibility of enforcing interval validity in a distributed system suffering unbounded levels of churn. This result stems from the fact that the overlay network topology used to disseminate the query and then collect its results must be characterized by two fundamental properties: (P1) it must be connected and (P2) a path connecting any two processes must remain stable (i.e. none of the processes constituting the path leaves the system and thus breaks the path while the algorithm is running).

The impossibility result is a consequence of the fact that guaranteeing the previous properties during the whole query execution is impossible if churn has no bound. However, real systems rarely experience completely unpredictable dynamism, but rather show limited and partially predictable fluctuation of churn rates (as shown by several analysis conducted on real large scale dynamic distributed systems like [27] and [16]). As a consequence, starting from the system model defined in [9], but realistically bounding the level of churn, the aim of this work is to show that it is possible to deterministically provide interval valid query answers by properly arranging nodes in a structured overlay and maintaining this structure along the time.

Existing solutions can only provide best effort validity semantics due to the absence of specific mechanisms for guaranteeing path stability (P2). The final results can thus vary depending on the churn rate, on how this churn impacts the network (i.e., hot spots vs uniformly distributed churn) and the specific overlay network

	tree forest	random graph
percentage of interval valid queries	0.20	0.95

Table 1: Percentage of interval valid queries in a scenario with 1000 processes and low churn rate for different overlay networks.

topology that is used. To give an intuitive measure of the low number of interval valid queries, let us consider the results shown in Table 1 where a counting algorithm is ran on top of two representative overlay networks, namely a structured forest of spanning trees (like the one generated by DHTs [28]) and a random graph, while both networks are subject to a low rate of churn (at every time unit 5 out of 1000 processes are replaced in the network)[2]. None of these solutions, with different degrees of quality, is able to provide guarantees on the interval validity of the query results.

This paper introduces the *Virtual Tree architecture* which is able to effectively support interval valid query answers in large-scale settings characterized by bounded churn. The architecture is composed of (i) an overlay network topology, named *Virtual Tree*, that exploits node clustering and a tree-like structure, (ii) an overlay management protocol able to maintain the virtual tree topology and, finally, (iii) a simple distributed in-network query answering algorithm. The rationale driving this design is that by substituting paths constituted by links and processes (that can fail) with virtual paths made of cliques of processes and their associated clustered links, it is possible to withstand several failures before properties (P1) and (P2) are violated, and to reactively repair the topology while queries are running. The paper proves that, as long as the churn experienced by the system is smaller than a given threshold, the Virtual Tree architecture deterministically provides interval valid query answers in a distributed system where communication latencies are bounded by a constant unknown from processes (i.e. relaxed asynchronous model). Moreover, an extensive set of experiments shows that the Virtual Tree architecture is extremely robust to disconnections, and is thus able to provide interval valid queries (with very high probability) when the churn rate surpasses the given threshold.

The rest of this paper is organized as follows: after discussing the related work (Section 2) we introduce the system model and describe the problem of defining a precise semantics for query answering in dynamic networks (Section 3). Section 4 describes the Virtual Tree architecture and proves that it provides interval valid query answers under the assumption of bounded churn. In Section 5 we provide an extensive experimental evaluation showing the performances of our solution in several scenarios experiencing different churn. Finally, Section 6 concludes the paper.

[2]Further details on the setup used to run these tests are available in section 5.

2. Related Works

Query Semantics in Dynamic Networks. Running an aggregate query on top of a dynamic network (e.g. a peer-to-peer network or a sensor network) is a non trivial task. Aggregate queries, in fact, require that all the values maintained by processes part of the network must be considered while computing the query result. Several algorithms have been proposed to provide query answers but most of them provide just best effort semantics [12], [17], [22] i.e., the algorithm does its best to gather all the values maintained by the processes part of the network despite their continuous arrival and departure. In [9], Bawa et al. propose three different semantics, namely *snapshot validity*, *interval validity* and *single-site validity*, defining when the result computed by an aggregate query can be considered *valid*. The authors also prove that in an unstructured dynamic network, with unbounded churn, the strongest semantics that can be deterministically satisfied is single-site validity. In [3] Baldoni et al. define one further semantics, namely *dynamic validity*, to take into account the effect of churn due to both join and leave operations. The same work also provides an aggregate query answering algorithm enforcing dynamic validity semantics.

More recently, Payton et al. [24] consider a slightly different validity properties. In addition, the authors provide an algorithm able to detect, for each query executed in the system, which is the strongest semantics that the answer can satisfy. A similar approach was previously proposed in [18], where the authors try to measure a metric able to state if the current network status is able to provide valid queries.

Virtual Nodes and Tree-based Topologies in Dynamic Systems. Several works [14, 20, 4, 15, 13] leveraged clusters of nodes in order to improve the robustness of an overlay network.

In [14], the authors introduce the notion of *Virtual Mobile Node*, while in [15], the authors use virtual nodes to tolerate byzantine behaviors.

The main difference between Overnesia [20] and the architecture proposed in this paper lies in the fact that the former does not provide any mechanism to prevent super peers from disappearing but they rather repair the overlay after this happened. As a consequence, the network can experience temporary disconnections showing the same problem arising with simple unstructured overlays. The idea of using Virtual Nodes has been adopted in Amazon's Dynamo [13] as well. However, Dynamo defines virtual nodes as virtual replicas in a logical key-space of "physical" nodes, and exploits them to balance load in its architecture.

In [4], the authors introduce a tree-based overlay network topology to support aggregate query executions satisfying interval validity in a distributed system prone to continuos churn. The main difference between the algorithm proposed in [4] and the Virtual Tree architecture introduced in this paper lies in the fact that the former adopts a purely probabilistic approach while the latter is able to provide deterministic guarantees when churn is bounded.

In [7] Baldoni and Tucci show the impossibility of achieving connectivity when an arbitrary large number of entities may arrive/depart/fails concurrently at any time and then present an algorithm maintaining a tree

overlay during *quiescent*[3] periods of the system. The algorithm implements an active leave (i.e. processes must execute a protocol before departing from the system) and the basic idea is to force the leaving process to find a leaf replacing it in the spanning tree before leaving. In [8], Bansal and Mittal propose an algorithm able to preserve a network spanning tree in a dynamic environment still ensuring (i) small node degree and network diameter and (ii) contained latency in the execution of a leave operation. Similarly to [7], the authors design an active leave operation. Differently, the solution we propose in this paper does not assume active leave but rather allows processes to depart at their wish, without taking any particular action, making then possible to mask failures.

The Virtual Tree graph topology introduced in this paper (see Section 4) can be considered part of the family of *cluster based* overlays. Some examples are represented by [21] and [1]. eQuus [21] makes use of small cliques (full graphs) of nodes to enhance performance and reliability in a DHT. Differently from our proposal, eQuus solves the problem of maintaining node clusters through *merge* and *split* operations; this solution was not applicable in our case due to fact that the merge operation introduces a change in existing virtual paths thus violating the required path stability property. PeerCube [1] builds and maintains an hypercube of virtual nodes constituted by cliques; also in PeerCube virtual nodes can merge and split. Recently the PeerCube structure was applied to the problem of isolating targeted attacks [2].

3. Background

System model. A dynamic distributed system is characterized by the continuous arrival and departure of processes (i.e. *churn* phenomenon). In order to model such dynamic behavior, we assume the *infinite arrival model* [23] where, in each run, infinitely many processes constituting the *system population* $\mathcal{V} = \{p_1, p_2, \cdots, p_i, \cdots\}$ may join/leave the system. However, at each time unit t, the distributed system is effectively composed only of a finite subset of the population, denoted as $\mathcal{V}_t$, including all the processes that have joined but have not yet left (i.e. $\mathcal{V}_t \subseteq \mathcal{V} = \{p_i \in \mathcal{V} |$ a time t, p_i has joined the system and it has not yet left$\}$). At every time t, each process $p_i$has a partial view on the set of processes currently part of the network i.e., it knows only a subset of the process identifiers in $\mathcal{V}_t$ and stores them in its *local view*.

Processes can communicate by exchanging messages on top of perfect point-to-point channels (i.e. messages are not created neither duplicated and if both sender and receiver do not leave the system, each message is eventually delivered). Every process p_i can exchange messages only with its neighbors, i.e. with processes in its local view.

Considering the set of processes part of the network and their local views, at each time t the system can be represented as a graph $\mathcal{G}(t) = (\mathcal{V}_t, \mathcal{E}_t)$, where $\mathcal{V}_t$ is the set of processes part of the network at time t and

[3] A time period is said to be quiescent if no more arrivals, departures and failures take place.

$\mathcal{E}_t$ is the set of edges $e_{i,k}$ connecting the processes in $\mathcal{V}_t$. We assume that edges are bi-directional: if process p_k is in the local view of process p_i, then p_i is in the local view of process p_k. In the following, we will use the terms node and process interchangeably wherever this does not create ambiguities. The graph $\mathcal{G}(t)$ represents the topology of the overlay network interconnecting processes in $\mathcal{V}_t$.

Note that, due to the effect of churn, the overlay network topology changes continuously as processes crash, join and voluntary leave the system. In particular, when a new process p_i wants to join the system, it executes a *join procedure* that, starting from a *bootstrap process*,provides p_i with a local view [19].

When a process p_i leaves the system, it does not perform any specific procedure: it just stops receiving and sending messages. A leave can thus be considered as a crash failure and in the rest of the paper we will refer to these two kinds of events as leaves. Without loss of generality, we assume that if a process leaves the system and later wishes to re-enter, it joins the system with a new identity.

Timing Model. As in [9], we assume the *relaxed asynchronous model*, i.e., there exist upper bounds on (i) process execution speeds, (ii) message transmission delays and (iii) clock drift rates. Note that, such bounds are not known by processes. In particular, we will assume that (i) computation time is negligible compared with message delays and (ii) there exists a known upper bound on the message delay δ i.e., given two correct processes p_i and p_k, connected by a communication link, then any message m sent from p_i, at a certain time t, will be delivered up to time $t + \delta$ to p_k, unless p_k leaves the system. For ease of presentation, we assume the existence of a global fictional clock.

Churn Model. At time t_0, the system is composed by N_0 processes. From time t_1, processes start to join and leave the system. We distinguish between *in-churn*, denoted as $\lambda(t)$, representing the percentage of processes that invoke the join operation at time t and *out-churn*, denoted as $\mu(t)$, representing the percentage of processes that leave the system at the same time t. Given the in-churn and the out-churn functions, the number of processes that join and leave at each time unit is represented respectively by the numbers $\lambda(t) \cdot N_0$ and $\mu(t) \cdot N_0$. We assume that churn is continuos, i.e. it does not exist a time instant t after which churn ends [25].

Aggregate Query Validity. In the following, we will consider the *interval validity* (IV) semantics, as defined in [9]. We assume that each process p_i maintains locally a value that can be used to calculate the result of an aggregation query.

Informally, IV semantics requires that the result of a query q is calculated by considering *at least* all the contributions coming from values maintained locally by processes that were already part of the system at the query issuing time and do not leave during the whole query execution period. Contributions from processes that leave/join during the query-processing are not necessarily required.

More formally, given a query q issued at a certain time t and the sequence $V = \{\mathcal{V}_t, \mathcal{V}_{t+1}, \cdots, \mathcal{V}_{t+j}\}$ of all the sets of processes part of the system during the query processing period $[t, t+j]$, the IV property can be defined as follows:

Definition 1. *Let $r = q(H)$ be the result of an aggregation query q executed in the period $[t, t+j]$ and evaluated on a set of values H. Let $V = \{\mathcal{V}_t, \mathcal{V}_{t+1}, \cdots, \mathcal{V}_{t+j}\}$ be the sequence of all the sets of processes part of the system during the query processing period $[t, t+j]$. We will say that q is interval valid if H is such that $\cap_{i\in[t,t+j]}\mathcal{V}_i \subseteq H \subseteq \cup_{i\in[t,t+j]}\mathcal{V}_i$.*

4. The Virtual Tree Architecture

In order to run an in-network query answering protocol, we must fist define the overlay network connecting processes that participate to the system. Several overlay network schemes are suitable, but tree-shaped topologies offer some clear advantages in the form of (i) low diameter (useful to quickly disseminate the query and collect its results), (ii) good scalability and (iii) the possibility to easily define protocols with deterministic termination conditions. However, three-shaped topologies are strongly susceptible to faults and dynamism, a problem that can severely affect the correct functioning of an in-network query answering protocol. In order to provide query answers complying with the IV semantics, in fact, two necessary conditions [9] must be met: (P1) the overlay network must always be connected and (P2) any process that does not leave the system during the query execution must have a stable path (a path that does not change) that connects it to the query source.

We solve the first problem proposing a new overlay network topology named *Virtual Tree* (VT) graph that exploits node clustering to improve its resilience to system churn. In order to address the second problem, we design an *overlay management protocol* (OMP) that migrates processes at run time from the lower levels of the VT graph to the upper ones in order to let churn impact only its leaves. Through this technique, the OMP can guarantee, as long as churn is bounded by a given constant, that the VT graph will meet conditions (P1) and (P2). On top of these two building blocks we deploy a simple in-network query processing algorithm that provides interval valid answers.

In the following we provide a more detailed and formal description of the three blocks of the *Virtual Tree Architecture*.

4.1. The Virtual Tree Graph

A VT graph is constituted by *virtual nodes* (VN) and *virtual links* (VL) arranged in a tree-shaped topology. A virtual node $VN_i = (V_i, E_i)$ is a subgraph of VT constituted by a set of nodes[4] V_i interconnected

[4]Note that, unless otherwise stated, in the following we refer to *processes* with the generic term *node* and to a *virtual node* with the acronym VN

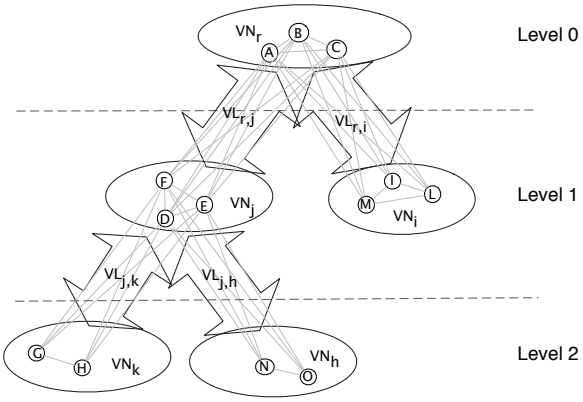

Figure 1: Example of a Virtual Tree graph.

by a set of edges E_i in a full graph (i.e. a *clique*). A virtual link $VL_{i,j}$ connecting two virtual nodes VN_i and VN_j is defined by the set of all the edges $e_{h,k}$ connecting any pair of nodes p_h and p_k such that $p_h \in VN_i$ and $p_k \in VN_j$. Let us note that nodes belonging to two adjacent VNs define a fully connected subgraph of the VT graph. As an example, in Figure 1, VN_i and VN_r represent two adjacent VNs constituted by nodes I, L, M and A, B, C respectively; $VL_{r,i}$ is the VL interconnecting them and it is constituted by links connecting each node in VN_i with every node in VN_r.

Now we can formally define the structure of a VT graph as follows:

Definition 2 (Virtual Tree graph). *Let $\mathcal{VN} = \{VN_1, VN_2, ..., VN_x\}$ be a set of virtual nodes and $\mathcal{VL}$ be a set of virtual links. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a* Virtual Tree *(VT) graph if:*

1. $\mathcal{V} = \bigcup_{i=1..x} V_i$*;*
2. $\mathcal{E} = (\bigcup_{i=1..x} E_i) \cup \mathcal{VL}$*;*
3. *each VN_i is associated to an integer defining its* level *in $\mathcal{G}$;*
4. *$\mathcal{G}$ contains a single virtual node at level* 0 *(*root *VN);*
5. *$\forall\ VN_i, VN_j$ with $i \neq j\ : V_i \cap V_j = \emptyset$;*
6. *every virtual node VN_j at level l is connected through a virtual link $VL_{j,k} \in \mathcal{VL}$ to one single virtual node VN_k at level $l-1$; in this case, we say that VN_j* is child *of VN_k and VN_k* is father *of VN_j;*
7. *no virtual link exists in $\mathcal{VL}$ connecting two virtual nodes at the same level;*

Given a VT graph, it is possible to define paths connecting any two virtual nodes:

Definition 3 (Virtual Path on VT graph). *Let $\mathcal{G} = (\mathcal{V},\mathcal{E})$ be a VT graph. Let $\mathcal{VN} = \{VN_1, VN_2, ..., VN_x\}$ be the set of virtual nodes of $\mathcal{G}$ and let $\mathcal{VL}$ be its set of virtual links. Given two virtual nodes VN_i and VN_j, a* virtual path *$\mathcal{P}_{i,j}$ on $\mathcal{G}$ between VN_i and VN_j is a sequence of virtual nodes such that for any two consecutive virtual nodes there exists a virtual link in $\mathcal{VL}$.*

From Definition 3, it follows that given a VT graph $\mathcal{G}$ and a virtual path between two virtual nodes VN_i and VN_j, there exists at least one path between any pair of nodes p_i and p_j, such that p_i belongs to VN_i and p_j belongs to VN_j. At the same time, given a path between two nodes p_i and p_j, such that p_i belongs to a virtual node VN_i and p_j belongs to a virtual node VN_j, there exists a virtual path between VN_i and VN_j.

4.2. Overlay Management Protocol

The OMP has two fundamental goals: (i) it must position joining processes in the overlay network maintaining a VT graph and (ii) it must guarantee that only leaf VNs can possibly disappear due to out-churn. This latter requirement stems from the observation that if a leaf VN is removed from the VT graph, the graph is still connected and none of the VNs still present in the graph will see a change in the paths that connect them to the root VN; therefore, both (P1) and (P2) will be preserved. The OMP relies on a view-maintenance algorithm [11] to keep local views on nodes up-to-date and consistent[5] with respect to each VN population. In the following, we will assume that such view maintenance algorithm will update the local views periodically.

Join Algorithm. New processes joining the system can, in principle, be accommodated in any VN (a join, in fact, cannot impact graph connectivity or path stability). However, considering that each VN is a fully connected graph, it makes sense to maintain its population small in order to reduce the overhead incurred for the maintenance of local views. Therefore, the OMP allows up to a maximum of N_{max} processes in a VN, where N_{max} is a configuration parameter. A VN that reaches this threshold will delegate the acceptance of joining processes to one of its children VNs (or will create a new one if needed).

In particular, when a new process p wants to join the system, it obtains a pointer to a process p_{ap} already part of the system from the bootstrap service [19]. Afterwards, p sends a join request message to p_{ap} that acts as follows:

- if the size of p_{ap}'s VN is smaller than N_{max}, then it sends back to p an acceptance message containing all the information concerning the population of the VN it belongs to, its father VN, all the children VNs (needed by p to join all the groups containing p_{ap}). In addition, p_{ap} also sends all the information related to concurrently running queries to ensure the termination of the query despite continuous process arrivals and departures (cfr. Section 4.3).
- if the size of p_{ap}'s virtual node is equal or larger than N_{max}, p can either forward the join request to a process in a child virtual node (if it already has K children) or create a new child virtual node

[5] In accordance with the relaxed asynchronous model assumption made in Section 3 here we assume that the view maintenance algorithm is able to guarantee deterministically consistent local views despite node joins/leaves.

containing only p. K is a configuration parameter defining the maximum number of children that a virtual node will create. Note that K affects the total number of edges in the VT graph and thus impact the cost incurred for maintaining local views.

Due to the dynamism induced by churn, the join procedure could possibly not terminate (e.g. because p_{ap} leaves the VT graph before correctly placing p in a virtual node). To avoid this problem joining nodes can use a timeout to reissue failed join request to a new bootstrap node.

Maintenance Algorithm. The maintenance algorithm has two goals: (i) maintaining the VT graph connected and (ii) ensuring virtual path stability. These can be both achieved by ensuring that at any time all VNs (with the exception of leaves) are composed by a *non-zero* population. This should be guaranteed when nodes leave the system too. To this aim, whenever the size of a certain virtual node VN_i decreases below a given threshold N_{min} the OMP starts migrating nodes from its children VNs and moves them in the father virtual node VN_i to reconstitute a "safe" size, thus avoiding its disappearance from the VT graph. When applied to the whole VT graph, this procedure can create a *cascade* effect such that nodes are progressively moved from leaf VNs to the upper levels. The N_{min} threshold is a function of the maximum allowed churn rate and, intuitively, must be calculated with the aim of giving "enough time" to the OMP to migrate nodes from the lower levels of the graph toward a non-leaf VN that is currently experiencing a local churn surge.

More in details, each process p periodically checks the size of the virtual node VN_i it belongs to and, as soon as it detects that the virtual node size $|V_i|$ is smaller than N_{min}, it takes the following steps:

1. each process p selects, according to a deterministic rule, $N_{min} - |V_i|$ processes, called *helpers*, among all those belonging to children VNs. Afterwards, p sends a HELP message to each helper; such a message contains all the information needed by the helpers to migrate in VN_i (i.e. all the membership information known at p).
2. delivering a HELP message, a process p_k, member of a virtual node VN_j, updates its local view according to the information received. From this point on, p_k starts to be a member of VN_i and stops to behave as member of VN_j.

The maintenance procedure continues to attract processes from children VNs until the virtual node size $|V_i|$ grows at least to N_{min}. Let us recall that the maintenance algorithm relies on a group membership service responsible of virtual nodes view maintenance and ensuring, in the relaxed asynchronous model, consistency of views.

4.3. In-network Query Processing Algorithm

The query processing algorithm is a simple adaptation of a broadcast/convergecast approach with partial result aggregation, modified to run on the VT graph topology. Without loss of generality, we assume that all queries are started by the root VN[6] that disseminates the query throughout the VT graph. Starting from the leaf VNs partial results are aggregated in intermediate VNs and forwarded to the upper levels until they reach the root of the VT graph. The absence of disconnections in the VT graph and the stability of virtual paths (both provided by the OMP), together with the structure of the query protocol guarantee that the returned result will include contributions from all processes that remained in the system for the whole query duration and will thus comply with the IV property.

More specifically, the query processing algorithm is started from the node p_i belonging to the root VN_r that issues the query. When a query q is issued by p_i the following steps occur:

1. p_i takes a snapshot of nodes belonging to VN_r (including itself)
2. p_i stores the lists of the current children VN identifiers at level 1.
3. p_i sends a message QUERY(q) to all processes part of VN_r and its children.
4. When a node p_j, belonging to a VN_x at some level i receives the QUERY(q) message, it:
   (a) repeats step 1, 2 and 3 at its own level;
   (b) sends QUERY_REPLY($q, value_j$) to all the nodes in VN_x;
   (c) waits until:
   (i) it collects QUERY_REPLY($q, value_k$) messages from each node p_k belonging in the snapshot of VN_x and
   (ii) it delivers a CHILD_QUERY_REPLY ($q, ag\_val_{VN_y}$) messages from at least one process in every child VN_y at level $i+1$;
   (d) evaluates the query on the set of values collected from its snapshot and on the partial aggregated results coming from the lower level;
   (e) sends a CHILD_QUERY_REPLY($q, ag\_val_{VN_x}$) message to all the nodes in the father virtual node at level $i-1$.

For any virtual node VN_i, its snapshot is updated every time that a view change occurs due to a leave and the corresponding process is removed. Nodes belonging to VN_i snapshot are required to participate in the query evaluation, while other nodes joining VN_i after the snapshot is taken are only required to participate in the aggregation process in order to guarantee that partial aggregated results flow toward the root VN. In this way, even if a virtual node population is completely "refreshed" between the query dissemination time

[6]Other nodes can delegate nodes in the root VN if needed.

and the partial aggregate evaluation delivery time, the query aggregate result can still proceeds toward the VT graph root.

During the query execution nodes can be attracted toward VNs at the upper levels. When a node moves to a new VN it keeps listening to messages exchanged in its old VN until all the pending query processing procedures are completed. In this way, a moving node will always answer to a running query either in its original VN or in the VN it is moving to avoiding the loss of its contribution.

4.4. Algorithm correctness

In the following, we will first prove that given an overlay network structured as a VT graph it remains connected as long as the churn rate is below a certain threshold (Lemma 1). Then we will show that in any connected virtual tree topology there always exists a stable virtual path between any pair of virtual nodes (Lemma 2) and in every interval of 2δ time there also exists a stable path between processes belonging to different virtual nodes (Lemma 3). Finally, we will show that these conditions are sufficient to ensure that the query algorithm introduced in the previous section always return interval valid results (Theorem 1).

Lemma 1. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the overlay network at time t_0. Let T_{move} be the upper bound on the time needed by any process to be part of a new view. If (i) $\mathcal{G}$ is a virtual tree graph and (ii) at any time t, $\sum_{i \in [t, t+T_{move}+1]} \mu(i) < N_{min}/N_0$, then $\mathcal{G}$ is always connected.*

Proof. $\mathcal{G}$ is connected as long as a path exists between any pair of nodes. Considering that at time t_0 $\mathcal{G}$ is a virtual tree (and thus it is connected), connectivity can be broken only due to the effect of churn. The join procedure has no impact on the connectivity of the overlay network as the algorithm adds new edges to $\mathcal{G}$ while old ones are unaffected. $\mathcal{G}$ can be partitioned if and only if all nodes belonging to a non leaf VN leave the system and no new node replaces them. Let us consider the worst case scenario where the out-churn affects a single VN while in-churn is 0. Without loss of generality, let VN_i be the first virtual node whose size reaches the threshold N_{min} and let t be the time when this is detected by processes belonging to VN_i. Let us denote as $V_i(t)$ the set of processes belonging to VN_i at time t and let $|V_i(t)| \geq N_{min} + 1 - \big(\mu(t) \cdot N_0\big)$[7].

According to the rules of the virtual tree maintenance algorithm, at time t, each process in VN_i selects, through a deterministic function, $N_{min} - |V_i(t)| + 1$ helper processes among those belonging to VN_i children, and asks them to move in VN_i. Considering that T_{move} represents the maximum amount of time needed for a node to move from a child VN to the father, the selected helper processes will complete their movement at time $t+T_{move}$ at the latest. In the meanwhile, for each time t' between t and $t+T_{move}$, $\mu(t') \cdot N_0$ processes leave and, in the worst case, they leave from VN_i (i.e. the size of VN_i shrinks to $N_{min} + 1 - N_0 \sum_{i \in [t, t+T_{move}]} \mu(i)$).

[7]Let us remark that, at time $t-1$, at least $N_{min}+1$ processes were part of VN_i and $\mu(t) \cdot N_0$ is the number of processes that leave at time t.

To avoid disconnections, we must guarantee that during T_{move}, at least one process remains in VN_i despite the out-churn. In the worst case, all processes leaving at time $t+1$ are all helper processes selected at time t. However, the maintenance algorithm will continue to select new helper processes, until there exists at least N_{min} processes in VN_i. Considering that, at each time t' the number of selected helper processes is $N_{min}+1-|V_i(t')|$ and that at time $t+1$ the number of selected processes is greater than $\mu(t)\cdot N_0$, we have that, $N_{min}+1-N_0\sum_{i\in[t,t+T_{move}+1]}\mu(i)>1$ from which the claim follows. $\square_{ProofLemma}$

Lemma 2. *Let $\mathcal{G}=(\mathcal{V},\mathcal{E})$ be the overlay network at time t_0. If (i) $\mathcal{G}$ is a virtual tree graph and (ii) at any time t, $\sum_{i\in[t,t+T_{move}+1]}\mu(i)<N_{min}/N_0$, then there always exists a stable virtual path connecting any two virtual nodes.*

Proof. The proof trivially follows by considering that (i) the graph is connected (Lemma 1) and (ii) at time t_0, $\mathcal{G}$ is a VT graph. As a consequence, no virtual node, except leaves, can disappear from the graph. Therefore, for any two virtual nodes the virtual path connecting them never changes and thus the claim follows. $\square_{ProofLemma}$

Lemma 3. *Let VN_i, VN_j and VN_k be three virtual nodes such that VN_i is father of VN_j and VN_j is father of VN_k. Let p_i and p_k be two nodes in VN_i and VN_k respectively. If, for any time t, $\sum_{i\in[t,t+T_{move}+1]}\mu(i)<N_{min}/N_0$, then at least one of the paths connecting p_i and p_k is stable for at least 2δ time units.*

Proof. The proof trivially follows from Lemma 1 and Lemma 2 considering that there exists at least one process in VN_j that does not leave its virtual node until an helper process completed the movement toward VN_j. $\square_{ProofLemma}$

Note that, Lemma 3 guarantees that messages can flow from VN_i to VN_k (and vice-versa) despite churn. Intuitively the migration of the helper nodes toward the root virtual node (induced by the maintenance algorithm) "moves" the effect of churn only in the leaf VNs. Observing the structure from a virtual-node point of view it is possible to see that only leaves disappear, while non-leaf virtual nodes remain stable preserving virtual paths. As a consequence, the query can safely be diffused in the virtual tree collecting all of the contributions from nodes that remain in the tree for the entire query duration.

Theorem 1. *Let p_i be the process issuing a query q at time t and let $\mathcal{G}=(\mathcal{V},\mathcal{E})$ be a VT graph at time t. If $\mathcal{G}$ is always connected then q terminates and satisfies the interval validity semantics.*

Proof. In the following we first show that every query eventually terminates and its results satisfies interval validity.

Termination. The query termination is guaranteed by the fact that eventually, each process will be

unblocked from the wait state. Due to the view maintenance mechanism, in fact, every process that left the system is eventually removed from the view of any other process and, due to the reliability of communication, every QUERY_REPLY$(q, r(q))$ message will be eventually delivered.

Interval Validity. Let us suppose, by contradiction, that $\mathcal{G}$ is connected, it is a VT graph, q terminates but it does not satisfy interval validity semantics. If q does not satisfy IV, it means that there exists at least one process p_j that has been part of the system for the whole period of the query evaluation while its contribution has not been considered in the query result. Let p_i be the process of the root virtual node that issued q (i.e. $p_i \in VN_r$). If the contribution of $p_j \in VN_j$ has not been considered, two cases could have happened: (i) p_j has never received the query or (ii) the contribution of p_j was not received by p_i.

Case 1: p_j never receives q. If p_j does not receive q, it means that the query(q) message has been lost while traveling from p_i to p_j. Due to the assumptions, there always exists at least one path connecting p_i and p_j. Considering the shortest path between two such nodes, two cases can happen: (i) the shortest path has length 1 or (ii) the shortest path has length greater than 1.

In the former case p_j and p_i are directly connected. Due to the reliability of the communication primitives, we have that any message sent from p_i to p_j will be eventually delivered. Thus, p_j never receives the query if and only if either p_i or p_j leave during the query execution. None of these two cases impact IV, thus leading to a contradiction.

In the latter case, there exists at least one virtual node VN_x on the path between VN_r and VN_j. Let us consider the generic case of three virtual nodes, namely VN_1, VN_2 and VN_3, belonging to the virtual path connecting VN_r and VN_j. Let us recall that any message m sent at time t will be eventually delivered; as a consequence, the query message broadcasted at time t from a node $u \in V_1$ will be eventually delivered to any node belonging to V_2, the message will be processed and forwarded to any node in VN_3 that will eventually receive the query. As a consequence, due to the fact that nodes can only move from children VNs to the father VN, i.e. p_j can only reduce its distanceto p_i, at any time the query moves along the virtual path always closer to its destination p_j, and thus this is true on all disjoint paths connecting two arbitrary nodes belonging to VN_1 and VN_3, respectively. Considering that the virtual path between VN_r and VN_j can always be decomposed in sub-paths of length 2, it is possible to iterate the reasoning above and find again a contradiction.

Case 2: p_i never receives the contribution sent by p_j. This case trivially follows from the previous reasoning by substituting p_i and p_j. $\square_{ProofTheorem}$

5. Experimental Evaluation

In this section we provide a detailed evaluation of the Virtual Tree architecture through simulations. The usage of a simulator was dictated by the need of testing the proposed approach in both large scale and

dynamic settings.

The evaluation is focussed on three distinct aspects:

- *overlay network connectivity*: we tested the robustness of the virtual tree overlay network topology under different levels of churn. In this set of experiments we have highlighted the main effects of the algorithm parameters on the connectivity of the overlay network.
- *message overhead*: this set of experiments assesses the cost of the virtual tree overlay management protocol in terms of messages needed to maintain the topology.
- *comparison with alternative approaches*: this set of experiments compares the Virtual Tree architecture with other solutions with respect to their ability to provide interval valid results in dynamic settings.

Note that we do not present the results obtained for the valid queries because for each query completed without disconnection in the VT the result was interval valid. The graphs that we can propose for the interval validity result to be a copy of the ones obtained for the connectivity, thus we decided to avoid to show them because they do not provide any additional information.

5.1. General Settings

The Virtual Tree architecture has been implemented in a round-based simulator used to simulate concurrent process activities and message network transmission delays. Processes are equipped with a view-maintenance algorithm. All the experiments start from an initial configuration with $N = 13500$ processes arranged in a complete VT graph (i.e. each virtual node, except the leaves, has the same number $k = 4$ of children[8]). Each virtual node is initially populated with N_{max} processes with values changing from test to test. When the simulation runs the VT graph is stressed with churn that lasts for the entire test duration (1000 rounds). All reported values are the result of 10 independent runs. Standard deviations are not shown as their values were always smaller than 5%.

Churn is modeled as a continuous periodic triangle-shaped process: during the first half period, processes enter the system (*growing phase*), while processes are progressively removed during the second half period (*shrinking phase*) . This churn model tries to reproduce the characteristic periodic oscillations observed in real large-scale dynamic distributed systems [29].

Metrics and Parameters. In the experiments we collected the following metrics:

(i) *% correct tests* representing the percentage of runs completed with a single connected component in the VT graph (i.e. no partitioning),

[8]Other tests have been performed for different values of k without sensible differences in the results.

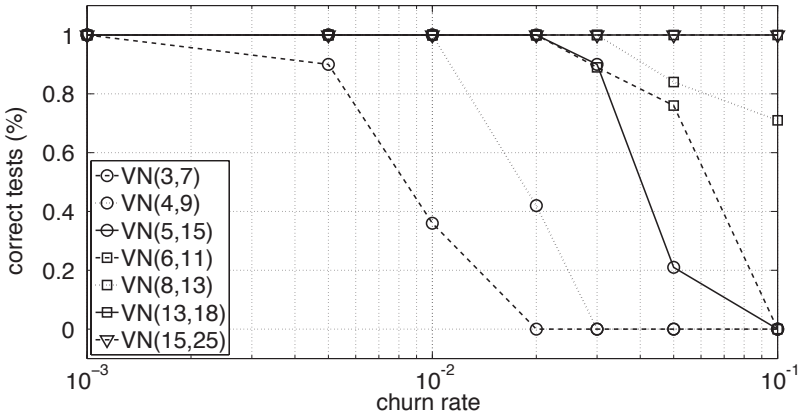

Figure 2: Graph connectivity vs. churn level for different configurations of N_{min} and N_{max}.

(ii) *virtual tree height* measuring the maximum path length between the root and leaf nodes in the VT graph,

(iii) *virtual node size* representing the average size of virtual nodes in the VT graph,

(iv) *message overhead* measuring the number of messages produced for the correct maintenance of the VT graph.

All the previous metrics have been evaluated by varying the following parameters:

(i) *virtual node size*: virtual node size is defined according to the lower threshold N_{min} that triggers the migration of processes from the lower level of the VT graph and the upper threshold N_{max} that imposes the forwarding of a join request to children VNs. Several configuration have been tested and each configuration is identified by a pair $VN(N_{min}, N_{max})$.

(ii) *churn level* $c \in [0, 1]$: it represents the percentage of processes that join/leave the system at each round.

During the experiments, we also collected measures about the percentage of interval valid queries. However, the results showed that only queries ran in graph experiencing disconnections lead to results violating interval validity. For this reason, we will only show pictures related to the connectivity evaluation.

5.2. Connectivity Evaluation

The first test checked graph connectivity at various churn rates for different settings of N_{min} and N_{max}. Figure 2 shows how the overlay network connectivity is affected from churn under different virtual node size configurations.

The virtual tree topology shows a typical bimodal behaviour: connectivity remains stable at 100% until a certain threshold for c (that depends on $VN(N_{min}, N_{max})$) is met; from that point on, the VT graph connectivity quickly drops to 0.

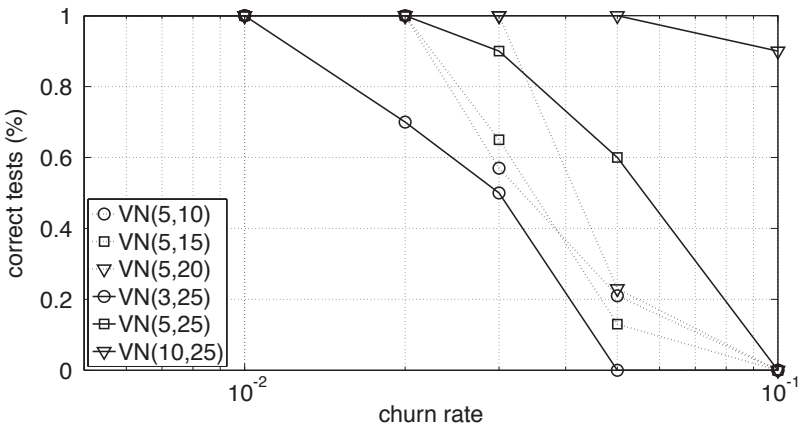

Figure 3: Impact of N_{min} and N_{max} on graph connectivity.

Note that the threshold is way larger than the limit for deterministic connectivity as defined in Section 4.4; if we consider, for example, the curve $VN(4, 9)$, the amount of churn needed to disconnect the graph is $c = 10^{-2}$; conversely, with the same settings the deterministic threshold inferred by the formula proposed in Lemma 1 is about two orders of magnitude smaller.

In Figure 3 we have tried to separately evaluate the effect on the connectivity of the variation of the two parameters N_{min} and N_{max}, for different churn rates. N_{min} regulates the robustness of VNs with respect to out-churn while N_{max} has no direct connection to the graph robustness as it only regulates the acceptance of new processes in "large" VNs. However, N_{max} could have an indirect impact on the VN resilience to out-churn surges, as it defines how large a VN can grow. Therefore, we tested this parameter as well.

The three solid curves show the algorithm behaviour by varying N_{min} only ($N_{max} = 25$): as expected they closely resemble the behaviour already reported in Figure 2. Conversely, the three dotted curves show the behaviour by varying N_{max} only ($N_{min} = 5$): all three cases report almost identical performance. We can thus conclude that the connectivity performance can be controlled only by varying the N_{min} parameter; this result doesn't come as a surprise as Section 4 already showed how connectivity is dependent only on the churn level, the time needed by the protocol to attract processes from low level virtual nodes, and the N_{min} parameter. We can thus conclude that the indirect impact of N_{max}, if present, is negligible.

Figure 4 shows the ability of the Virtual Tree OMP to move the effect of churn toward the leaf virtual nodes. The plot reports the average virtual node size at different levels of the VT graph for several N_{min} values with $c = 10^{-2}$. The plot shows how virtual nodes always maintain a stable size that is slightly larger than N_{min}; the only exception is represented by virtual nodes positioned at the lower levels of the tree that with high probability are leaves: these nodes cannot attract processes from children nodes, and are thus condemned to sizes that are way below the N_{min} value.

The concept is also remarked by Figure 5 in which we plot the average size of a non leaf virtual node

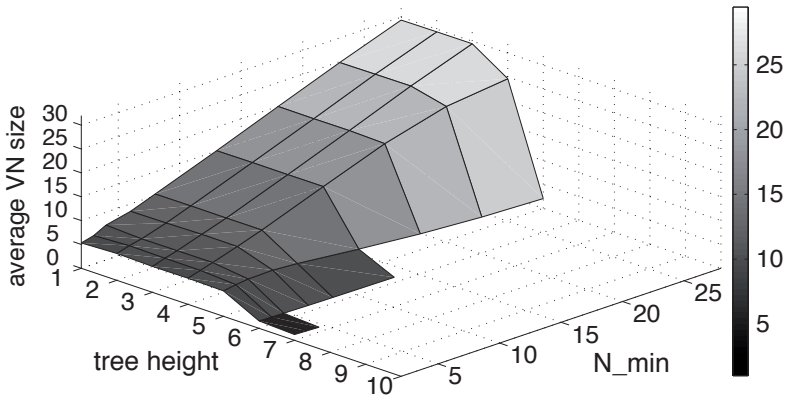

Figure 4: Virtual node sizes at different levels of the VT graph.

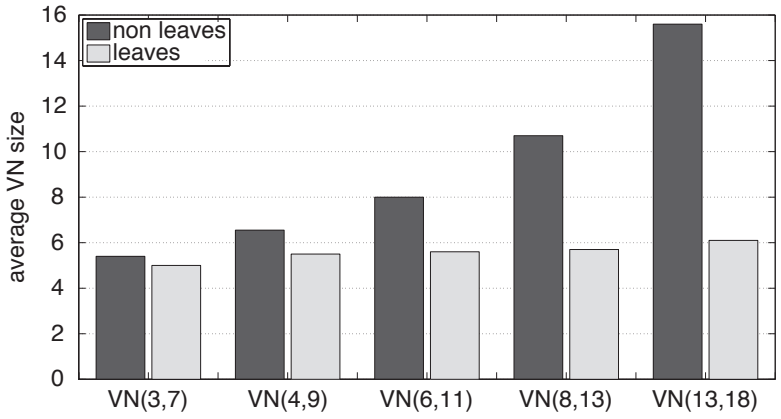

Figure 5: Non leaf virtual node size vs virtual leaf size.

versus the average size of a leaf virtual node. As the previous graph, the comparison shows that the managing procedures are able to constantly maintain the size of all the non-leaf virtual larger than N_{min}.

Figure 6 shows how the maximum three height varies with churn. From the different curves it is possible to catch a general behaviour: as the values of N_{min} and N_{max} are increased, the maximum height tends to decrease; this is an obvious consequence of the larger amount of processes that fit in virtual nodes at the highest level of the tree. The curves for small N_{min} values are truncated as experiments with larger churn levels reported disconnections in the graph. Curves for large N_{min} values (i.e. $N_{min} \in \{8, 13, 15\}$) show a rather unexpected behaviour as the max height first increases as churn grows, then reaches a maximum and starts to decrease for high churn levels. The initial growing phase is a consequence of growing churn that tends to increase the average number of children VNs that are created in the VT graph. After this initial growth the churn start to be so intense that many nodes are not able to finish their join procedure before being removed from the graph; as a consequence, the average number of nodes in the VT graph starts to

shrink and its height shrinks as well.

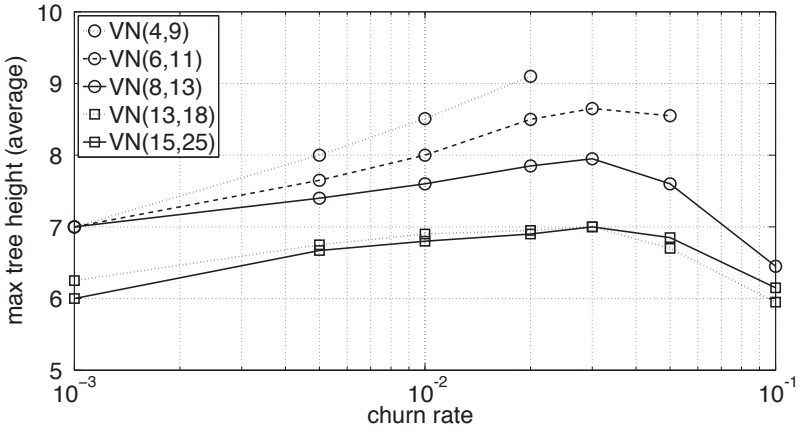

Figure 6: Maximum Virtual Tree height vs churn level.

5.3. Overhead Evaluation

The Virtual Tree architecture relies on the presence virtual nodes whose membership is maintained by a view-maintenance algorithm. The test reported in Figure 3 showed us that N_{min} controls the resilience of VNs to out-churn. Conversely, N_{max} is responsible of limiting the cost incurred in the VN maintenance: the larger N_{max} is and the larger is the overhead imposed by the view-maintenance algorithm. However, at the same time, the larger the difference between N_{min} and N_{max} is and the higher the probability of fixing the erosion introduced by the out-churn on the VN only waiting for new joins will be. This is an important aspect, as every time a virtual node size falls below the N_{min} threshold the OMP spends a large amount of messages for moving nodes from the children VNs to the father. In some sense the gap between N_{min} and N_{max} represents a *buffer* which prevents premature *attraction procedures* to happen.

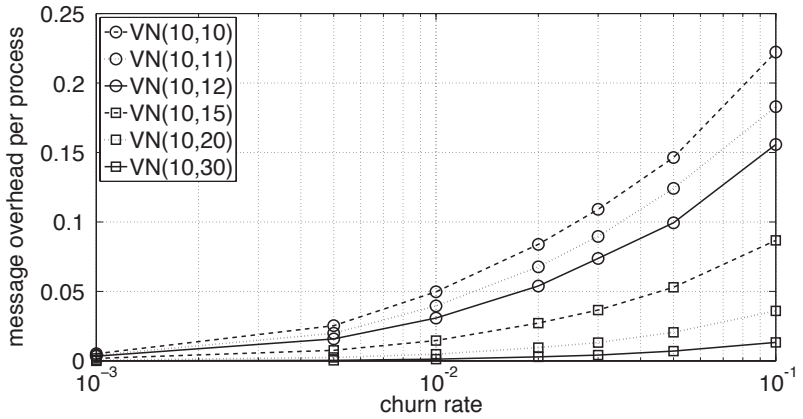

Figure 7: Impact of N_{max} on overhead generated by the Virtual Tree OMP.

Figure 7 reports the message overhead per process versus churn for different N_{max} values and $N_{min} = 10$.

The message overhead, expressed as average number of HELP messages produced per nodes and per round, increases with churn as a consequence of the large process mobility among virtual nodes needed to keep non leaf virtual node populations above the N_{min} threshold. The growth is larger for configurations with lower $N_{max} - N_{min}$ gaps. This behaviour is justified by the fact that the smaller is the delta, the larger is the probability to have a VN with size smaller that N_{min}. The extreme case is represented by the configuration $vn(10, 10)$ that forwards all process joins toward the leaf virtual nodes and, for every leave, attracts a node from a children.

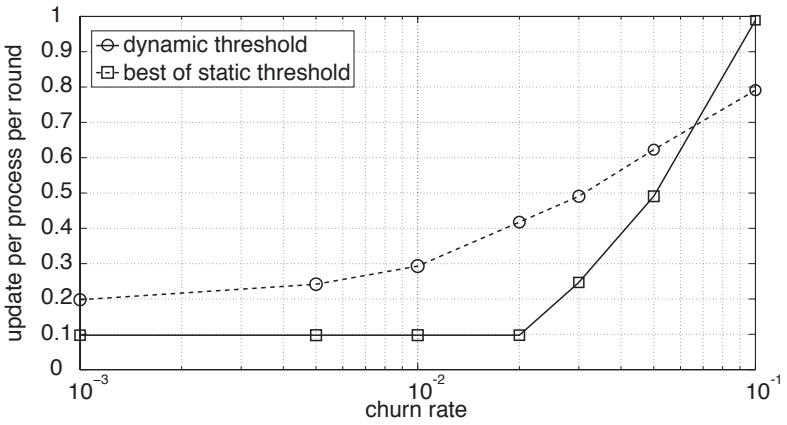

Figure 8: View Update frequency: best of static vs dynamic.

In addition, the view maintenance procedure is usually configured to run periodically and check the consistency of information stored in each local view. The update frequency is usually fixed and must be defined as a function of the maximum churn that the system expects to experience. This means that a careful evaluation of the expected churn rate must be done before starting the system and that the latter must be configured as to tolerate large churn surges even if churn rates will be mild on average for most of the system lifetime. In order to improve the efficiency of our solution we thus introduced a dynamic update frequency whose purpose is to dynamically adapt by locally monitoring at run time churn rates experienced in each single VN. The dynamic frequency adaptation works as follows: let f be the current update period. After f rounds, if the local view was not updated, then the new update period will be $f = f + 1$, otherwise it will be $f = f/2$. In order to avoid too large update periods, that can easily lead to the loss a VN in case of a local churn surge, the value of f is limited by an upper threshold defined as a configuration parameter.

Figure 8 shows the effect of the dynamic update frequency for the local view-maintenance algorithm. For each level of churn included in the plot, we first evaluated the lower static update frequency needed to avoid

disconnections[9]. Then we run the same test with the dynamic frequency mechanism.

The solution based on the dynamic frequency mechanism exhibits worse performance if compared with the best static setting with the only exception of churn rate $c = 10^{-1}$. This is due to the fact that the dynamic frequency grows linearly and decreases exponentially and thus assumes values that are often bit larger the perfect one (represented by the best static frequency). The trend is inverted in the case of churn rate $c = 10^{-1}$ because, for this leave/join rate churn is less uniform on different VN as a consequence of shorter update periods. In this case the dynamic update frequency mechanism is able, for example, to impose a larger update period $f = 2$ in VNs experiencing mild churn, while using higher frequencies in VNs where churn is stronger.

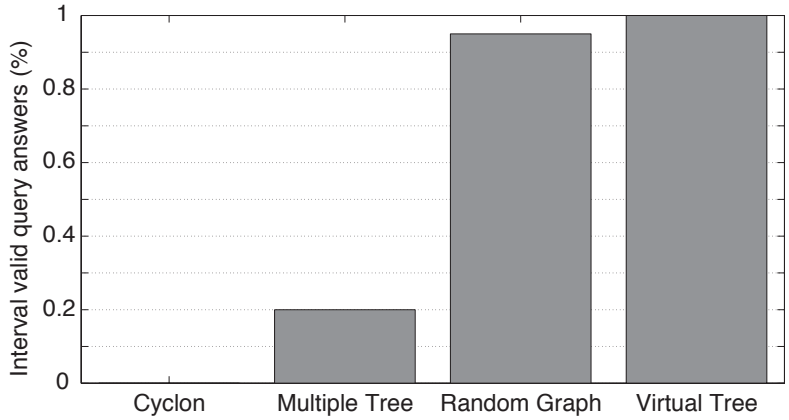

Figure 9: Percentage of interval valid queries in a scenario with 1000 processes and low churn rate for different overlay networks.

5.4. Interval validity evaluation

Figure 9 reports an evaluation of the ability of different overlay networks to support aggregate query answering algorithms. This comparison provides the reader with a qualitative analysis that substantiate the motivation presented in the Introduction. We compared an overlay network generated by Cyclon [30], an overlay network designed as a forest of spanning trees, a random graph and a virtual tree overlay network.

Cyclon is a gossip-based protocols that maintains local views that closely represents uniform random samples of the system population. Graphs built by Cyclon show strong connectivity and low diameters even when perturbed with strong churn. These characteristics are provided by means of a periodic shuffle of local views that tends to evict dangling links caused by out churn. As a query answering protocol we implemented a simple flooding-based techniques that naively propagate the query and the corresponding answer through all the reachable processes. The second implementation is based on a forest of spanning trees[10] (multiple tree) where the query is propagated in a parallel way (all the trees are exploited at the same time for query

[9]Note that this optimal frequency can be calculated only *a posteriori*.
[10]note that each DHT [28],[26] can be represented by a forest of spanning trees

propagation and answers collection). This setup resembles the Virtual Tree query answering protocol with the only difference being represented by the lack of a OMP protocol able to repair damages occurring to trees due to churn. The third implementation is based on a random graph, a topology known for its strong connectivity and low diameter. Strong connectivity lets random graphs sustain massive node failures with extremely low probabilities of disconnection. However, without specific algorithms, they are not able to repair damages caused by churn. In this case we employed the same query answering algorithm considered in the Cyclon case.

The topologies considered in Figure 9 are composed by 1000 processes. In each run we have injected churn and we tried to execute 100 queries. All the topologies have been rebuilt from scratch before any single query execution. Concerning Cyclon, we let the protocol execute 50 view shuffles before executing the query to let it reach a steady state. For the multiple trees case we built a forest formed by 10 spanning trees. In all the three cases we configured the graphs such that, on average, nodes always had the same degree.

Figure 9 reports the percentage of queries that ended with an interval valid answer for the three solutions in a scenario with $c = 0.005$.

The overlay network generated by Cyclon completes a very few Interval Valid query answering. Such bad performance is caused by the shuffling techniques employed by Cyclon to refresh local views. This shuffle mechanism continuously changes the paths between two processes on the overlay networks making the paths extremely unstable. Therefore, the probability that a querying node gets all the answers from nodes that remain in the system for the whole query duration is extremely low and close to 0.

The solution based on multiple trees overlay network is able to provide interval valid query answers with low probability. This performance is caused by the absence of any mechanism able to counteract the disruption of trees due to churn. Therefore this overlay network disconnects quickly. On the other hand if a tree, by chance, is not disconnected by churn during the time of the query, then the returned answer is interval valid.

The random graph-based solution exhibits good performance: it achieves Interval valid querying answers with high probability. This is due to the robustness of the random graph structure with respect to churn and to the huge number of redundant paths between any two processes. This provide a good degree of stability to paths during the time of the query. Only tests performed with larger populations (up to 10.000) show that also the performance of the Random Graph get worse. This result stem from the fact that the larger is the population and the larger is the query execution time: with larger query execution time the churn has more time to disconnect the network corrupting the query result.

Figure 10 compares the cost of executing interval valid queries. To the best of our knowledge WildFire [9] is the only other protocol able to ensure valid results. More precisely WildFire was introduced by Bawa et al. as an algorithm able to provide *single-site* valid results. Note that single site validity semantics is weaker

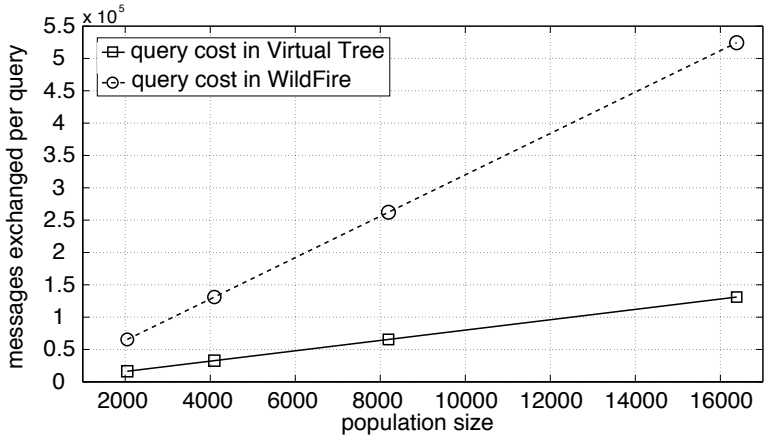

Figure 10: Query cost comparison.

than interval validity as it only guarantees that the result of the query contains at least contributions from all the processes that were connected through a stable path toward the querying node for the query duration. WildFire consists of two phases: the broadcast phase and the convergecast phase. In the broadcast phase the query is forwarded inside the network using a broadcast primitive; the broadcast propagation creates multiple spanning trees that are used during the convergecast phase. Each process, receiving the broadcast, starts its convergecast phase sending back its contribution toward the root of the spanning trees. In our tests we reproduced within a VT graph the same settings proposed in [9]: variable number of processes $N = [2^{11}, 2^{12}, 2^{13}, 2^{14}]$, and continuous churn ($c = 10^{-2}$).

The result shows that a query performed in Virtual Tree uses, on average, 1/4 of the messages of a WildFire query. This difference stems directly from the approaches adopted by the two different solutions: WildFire builds the structure needed to disseminate the query and collects its results *on-demand*, i.e. when a query is started, while our solution exploits the existing structured overlay (a VT graph) reducing the query cost but paying the overlay maintenance one. Thus, we can conclude that the strategy adopted by WildFire clearly pays back in all those situations where queries are executed not so frequently while the Virtual Tree architecture could be preferable in query-intensive computations as it supports a stronger query validity semantics with a lower *per-query* cost.

6. Concluding Remarks

Providing interval valid queries in large scale dynamic networks with unbound churn is provably impossible due to the lack of connectivity and path stability between processes. This paper presented Virtual Tree , an architecture comprising an overlay management protocol able to build and maintain a Virtual Tree graph that remains connected when churn is below a given threshold. The Virtual Tree architecture also includes a distributed query protocol that guarantees deterministically interval valid queries when run on the top of a

Virtual Tree graph. The paper proves that this latter property holds, in a distributed system with bounded communication latencies, as long as churn remains below a threshold that is function of a configuration parameter defining the size of virtual nodes. The experimental evaluation confirms and extends the theoretical results showing that connectivity, and thus interval validity, can be practically obtained even for churn rates larger that the theoretical one.

Acknowledgements

We want to thank the anonymous reviewers for their feedbacks and suggestions that greatly helped us in improving the quality of the paper.

[1] E. Anceaume, R. Ludinard, A. Ravoaja, and F. Brasileiro. PeerCube: A Hypercube-Based P2P Overlay Robust against Collusion and Churn. In *Proceedings of the 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, 2008.

[2] E. Anceaume, B. Sericola, R. Ludinard, and F. Tronel. Modeling and Evaluating Targeted Attacks in Large Scale Dynamic Systems. In *Proceedings of the 41th annual IEEE/IFIP Internation Conference on Dependable Systems and Networks (DSN)*, 2011.

[3] R. Baldoni, M. Bertier, M. Raynal, and S. Tucci Piergiovanni. Looking for a Definition of Dynamic Distributed Systems. In *Proceedings of the 9th International Conference on Parallel Computing Technologies (PaCT)*, 2007.

[4] R. Baldoni, S. Bonomi, A. Cerocchi, and L. Querzoni. Improving Validity of Query Answering in Dynamic Systems. In *Proceedings of the 3rd ACM SIGOPS/SIGACT Workshop on Reliability, Availability, and Security (WRAS)*, 2010.

[5] R. Baldoni, S. Bonomi, A. Cerocchi, and L. Querzoni. Virtual tree: A robust overlay network for ensuring interval valid queries in dynamic distributed systems. In *Proceedings of the 13th International Conference on Distributed Computing and Networking (ICDCN)*, 2012.

[6] R. Baldoni, S. Bonomi, A.-M. Kermarrec, and M. Raynal. Implementing a Register in a Dynamic Distributed System. In *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2009.

[7] R. Baldoni and S. Tucci Piergiovanni. Connectivity in Eventually Quiescent Dynamic Distributed Systems. In *Third Latin-American Symposium on Dependable Computing (LADC)*, pages 38–56, 3 2007.

[8] T. Bansal and N. Mittal. A scalable algorithm for maintaining perpetual system connectivity in dynamic distributed systems. In *Proceedings of the 24th IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–12, 2010.

[9] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani. The price of validity in dynamic networks. *Journal of Computer and System Sciences*, 73(3):245–264, 2007.

[10] P. Bonnet, J. Gehrke, and P. Seshadri. Towards Sensor Database Systems. In *Proceedings of the 2nd International Conference on Mobile Data Management (MDM)*, 2001.

[11] G. V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Computing Surveys*, 33:427–469, December 2001.

[12] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate Aggregation Techniques for Sensor Databases. In *Proceedings of the 20th International Conference on Data Engineering (ICDE)*, 2004.

[13] G. De Candia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of 21st ACM SIGOPS symposium on Operating systems principles (SOSP)*, 2007.

[14] S. Dolev, S. Gilbert, N. A. Lynch, E Schiller, A. A. Shvartsman, and J. L. Welch. Virtual mobile nodes for mobile ad hoc networks. In *Proceedings of the 18th Annual Conference on Distributed Computing (DISC)*, pages 230–244, 2004.

[15] I. Eyal, I. Keidar, and R. Rom. Distributed Clustering for Robust Aggregation in Large Networks. In *Proceedings of the Workshop on Hot Topics in Dependable Systems (HotDep)*, 2009.

[16] S. Guha, N. Daswani, and R. Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[17] I. Gupta, R. van Renesse, and K. P. Birman. Scalable Fault-Tolerant Aggregation in Large Process Groups. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2001.

[18] N. Jain, P. Mahajan, D. Kit, P. Yalagandula, M. Dahlin, and Y. Zhang. Network Imprecision: A New Consistency Metric for Scalable Monitoring. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 87–102, 2008.

[19] M. Jelasity, A. Montresor, and Ö. Babaoglu. The Bootstrapping Service. In *Proceedings of the International Workshop on Dynamic Distributed System (IWDDS)*, 2006.

[20] J. Leitão and L. Rodrigues. Overnesia: a Resilient Overlay Network for Virtual Super-Peers. Technical Report 56, INESC-ID, December 2008.

[21] T. Locher, S. Schmid, and R. Wattenhofer. eQuus: A provably robust and locality-aware peer-to-peer system. In *Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing (P2P)*, 2006.

[22] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Operating System Review*, 36:131–146, 2002.

[23] M. Merritt and G. Taubenfeld. Computing with infinitely many processes. In *Proceedings of the 14th International Conference on Distributed Computing (DISC)*, 2000.

[24] J. Payton, C. Julien, G. C. Roman, and V. Rajamani. Semantic self-assessment of query results in dynamic environments. *ACM Transactions on Software Engineering and Methodology*, 19:12:1–12:33, 2010.

[25] S. Rhea, D. Geels, and J. Roscoe, T.and Kubiatowicz. Handling churn in a dht. In *Proceedings of the USENIX Annual Technical Conference*, 2004.

[26] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg (Middleware)*, pages 329–350, 2001.

[27] S. Saroiu, K. P. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking (MMCN)*, 2002.

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

[29] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, 2006.

[30] S. Voulgaris, D. Gavidia, and M. Van Steen. CYCLON: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2), 2005.