

UNIVERSITÀ LA SAPIENZA

VIA ARIOSTO 25, I-00185 ROMA, ITALY

**Virtual Tree: a Robust Overlay  
Network for Ensuring Interval  
Valid Queries in Dynamic  
Distributed Systems**

Roberto Baldoni, Silvia Bonomi, Adriano Cerocchi and Leonardo Querzoni  
[baldoni, bonomi, cerocchi, querzoni]@dis.uniroma1.it

MIDLAB TECHNICAL REPORT 4/11 2011

## Abstract

This paper studies the complexity of answering aggregation queries with the interval validity semantics in a distributed system prone to churn. The interval validity semantics states that the query answer must contain contributions from at least all the processes that were part of the system for the whole query execution time. Solving this problem in unstructured distributed systems with unbounded churn is impossible due to the lack of connectivity and path stability between processes. This paper presents a novel overlay management protocol, namely Virtual Tree OMP, that, under the assumption of bounded churn, maintains a structured overlay network with guaranteed connectivity and path stability. The resulting Virtual Tree graph is a tree-shaped topology with virtual nodes constituted by cliques of processes and virtual links constituted by multiple communication links connecting processes located in adjacent virtual nodes. This paper also introduces a distributed query algorithm that, working on top of the Virtual Tree graph, guarantees answers complying with the interval validity semantics. Finally, the paper explores through an extensive experimental analysis the performance characteristics of the overlay maintenance protocol under different churn levels.

## 1 Introduction

Today's large scale distributed systems are characterized by strong dynamics caused the inherent unreliability of their constituting elements (e.g. process and link failures, processes joining or leaving the system). This continuous dynamism has a strong negative impact on distributed algorithms designed to work on such systems.

In-network aggregate query answering algorithms [1] are run on top of such systems (e.g. in large scale sensor networks) to collect application-level aggregate statistics in a scalable manner. Strong network dynamism can severely affect the correct functioning of such protocols. As an example, consider the representation of a sensor network reported in Figure 1; assume that at time  $t_1$  sensor  $a$  issues a query in the network (left box) to collect some aggregate data (e.g. the sum of all sensed values); following a simple broadcast/convergecast scheme the query will be routed throughout the network (center box) and the results aggregated on their way back to the source  $a$  (right box) where it will complete at time  $t_2$ . What happens to the query result if a sensor fails during the time interval  $\{t_1, t_2\}$  required to calculate the query result? If node  $b$  fails this does not constitute a problem as only its contribution to the result will be missed. Conversely, if node  $c$  fails an entire branch of the query broadcast tree will be pruned and the query result will thus be severely affected by missing contributions from nodes that are still part of the network. In the end,  $a$  will receive some results, but it will be unable to assign them a precise semantics.

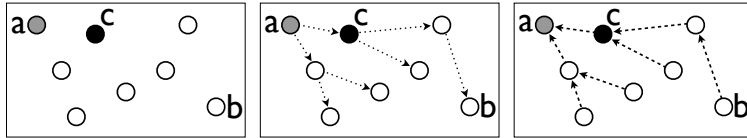


Figure 1: aggregate query in a sensors network

The problem of defining precise semantics for in-network query answering in large-scale dynamic networks was introduced by Bawa et al. in their seminal work [2]. One of the semantics introduced in that work, namely *Interval Validity* (IV), requires the answer to *contain 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. The same work also proved the impossibility of enforcing interval validity as long as churn (i.e. the rate at which processes can crash/leave or join the system) is unbounded. However, practical experience shows that many systems experience a continuous (in time) but limited (in its strength) level of churn, and that this level can be reasonably predicted by analytical assessment (e.g. by knowing the duty cycle period and mean time between failure figure of sensors) or by direct measurements [3]. By exploiting this aspect, it would be thus possible to circumvent the previously cited impossibility result and provide interval valid answers to in-network aggregate queries.

On the basis of these motivations, this paper presents the first solution for in-network aggregate query processing that is able to provide query answers complying with the *interval validity* semantics in large-scale dynamic systems with bounded churn. More specifically, our solution includes an overlay management protocol able to build an application-level network constituted by virtual nodes and virtual links; the protocol guarantees that (i) the network remains connected and (ii) the path connecting any two virtual nodes remain stable (i.e. it does not change) despite churn. Our solution also include a simple algorithm for in-network aggregate query processing that, when run on top of our overlay, provides results that provably comply with the IV semantics. The correctness of our approach is supported by both formal proofs<sup>1</sup> and an extensive set of simulation-based experiments.

The rest of this paper is organized as follows: Section 1 introduces the system model and precisely defines the Interval Validity semantics, Section 1.1 introduces the Virtual Tree architecture, Section 1.2 evaluates the Virtual Tree graph connectivity under different churn levels, Section 1.3

<sup>1</sup>Note that, due to space constraints, the proofs are available in a separate technical report[4].

discusses the related works and, finally, Section 1.4 concludes the paper providing some future directions for the improvement of the proposed solutions.

## 2 Interval Validity in Dynamic Distributed Systems

**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* (as defined in [5]) where, in each run, infinitely many processes  $\Pi = \{\dots, p_i, p_j, p_k \dots\}$  may join/leave the system (the set  $\Pi$  is also called *distributed system population*).

However, at each time unit  $t$ , the distributed system is effectively composed only by a subset of the population, denoted as  $\Pi(t)$ , including all the processes that have joined but have not yet left (i.e.  $\Pi(t) \subseteq \Pi = \{p_i \in \Pi \mid \text{a time } t, p_i \text{ has joined the system and it has not yet left}\}$ ).

Processes of the distributed system can communicate only by exchanging messages on top of perfect point-to-point channels (i.e. messages are not created or lost from the channel and if both sender and receiver are not left, each message is eventually delivered).

Each process  $p_i$  has only a *partial knowledge* of the distributed system, i.e. it has a *local view* of processes to communicate with.

Thus, at each time  $t$ , processes of the distributed system, together with their partial views, can be seen as a graph  $\mathcal{G}(t) = (\mathcal{V}, \mathcal{E})$ , where the set  $\mathcal{V}$  of nodes representing processes of the distributed system (i.e. at each time  $t$ ,  $\mathcal{V} = \Pi(t)$ ) and there is an edge  $e_{i,j} \in \mathcal{E}$  connecting two nodes  $p_i \in \mathcal{V}$  and  $p_j \in \mathcal{V}$  if process  $p_j$  is in the local view of process  $p_i$  and vice-versa. In the following, we will use the terms node and process interchangeably.

Note that, due to the effect of churn, the overlay network graph changes continuously due to process crashes, joins and voluntary leaves. 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, containing the identifiers of a subset of processes part of the system [6]. As a consequence, a new node corresponding to  $p_i$  is added to the graph together with all the edges.

When a process  $p_i$  leaves the system, it does not perform any specific procedure, it just stops receiving and sending messages and the corresponding node of the overlay is removed together with all its incident edges. Note that, a voluntary leave can be considered as a crash failure and in the rest of the paper we will refer to these two kinds of event 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 [2], we assume the *relaxed asynchronous model*, i.e., there exists known upper bounds on (i) process execution speeds, (ii) message transmission delays and (iii) clock drift rates. In particular, we will assume the existence of a known universal maximum delay  $\delta$  on the communication channels, i.e., any message  $m$  sent at a certain time  $t$  by a correct process  $p_i \in \Pi(t)$  is delivered up to time  $t + \delta$  to its receiver  $p_j$ . Note that, in this setting, processes can reliably monitor its neighborhood (e.g. though heartbeats sent periodically).

**Churn Model.** At time  $t_0$ ,  $|\Pi(t_0)| = N_0$ . At time  $t_1$ , processes start joining and leaving the system. We distinguish between (i) *in-churn*, denoted as  $\lambda(t)$ , representing the percentage of processes that invokes the join operation at time  $t$  and (ii) *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 in each time unit is represented respectively by the numbers  $\lambda(t) \cdot N_0$  and  $\mu(t) \cdot N_0$ . We assume that the churn is continuous, i.e. there not exists any time  $t$  after which the churn ends.

**Validity of query answering** A generic configuration, identified as  $\mathcal{V}_i$ , is defined by the set of processes belonging to the system at a certain point in time. During its lifetime the system is characterized by a totally ordered sequence of configurations. Two successive configurations in this sequence,  $\mathcal{V}_i$  and  $\mathcal{V}_{i+1}$ , differ only for one process that either left or joined the system. The sequence of configurations is an abstraction representing a serialization of the evolution of  $\Pi$  in time due to the effects of churn. Given a query and the sequence  $V = \{\mathcal{V}_i, \mathcal{V}_{i+1}, \dots, \mathcal{V}_{i+j}\}$  of all the configurations experienced during the query execution, the interval validity property [2] can be defined as follows:

**Definition 1** *A query is said to be interval valid if its result is calculated on a set of processes  $H$  such that  $\bigcap_{x \in [i, i+j]} \mathcal{V}_x \subseteq H \subseteq \bigcup_{x \in [i, i+j]} \mathcal{V}_x$ .*

Intuitively, interval validity is satisfied when the result is calculated considering at least all contributions coming from the set of processes that remained in the system during the whole query execution; contributions from processes that leave/join the network while the query is running are not necessarily required.

### 3 The Virtual Tree Architecture

In order to run an in-network query answering protocol we must first define the overlay network connecting the nodes that will 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 broadcast the query and collect its results), (ii) good scalability and (iii) the possibility to easily define protocols with clear stopping conditions. However, tree-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 [2] must be met: (1) the overlay network must always be connected and (2) any node 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 an overlay network topology named the *Virtual Tree (VT)* graph that exploits node clustering to improve its resistance to system dynamics. In order to address the second problem we design an overlay management protocol (OMP) that at run time migrates nodes from the lower layer of the *VT* graph to the upper ones in order to let churn impact its leaves. Through this technique the OMP can guarantee, as long as churn is bounded by a given constant, that the *VT* graph will remain connected and its paths will be stable. On top of these two building blocks we deploy a simple in-network query processing algorithm that provides interval valid answers.

Here we give a high level description of these three blocks and leave the finer details to the subsequent, more formal, sections.

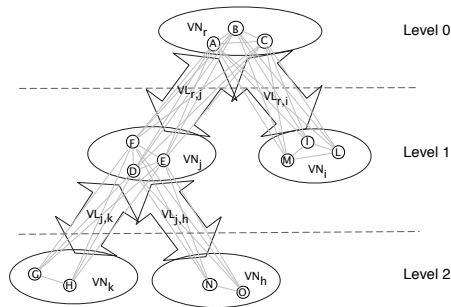


Figure 2: Example of a Virtual Tree graph.

**Virtual Tree graph:** a *VT* graph is constituted by *virtual nodes (VN)* and *virtual links (VL)* arranged in a tree-shaped topology. A *VN* is constituted

by a set of nodes<sup>2</sup> interconnected by a full graph (i.e. a *clique*). A *VL* connecting two *VNs* is constituted by the set of links connecting any pairs of nodes pertaining to the two different *VNs*. Nodes pertaining to two adjacent *VNs* form a completely connected subgraph of the *VT* graph. For example, in Figure 1.1,  $VN_i$  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 the links connecting every node in  $VN_i$  with every node in  $VN_r$ .

**Overlay management protocol:** the OMP has two fundamental goals: (i) it must position joining nodes in the *VT* graph and (ii) it must ensure that only *VNs* representing leaves of the tree will possibly disappear as a consequence of a node leave. This last requirement stems from the observation that if a leaf *VN* disappears, none of the *VNs* still present in the graph will see a change in the paths that connect them to the root *VN*, thus the stability of paths connecting *VNs* will be guaranteed. These two goals are reached by arranging nodes in the *VNs* such that at any point in time any *VN* (with the exception of leaf *VNs*) is constituted by a number of nodes that is possibly above a minimum given threshold  $N_{min}$ . When the size of a *VN* falls below  $N_{min}$  the OMP starts to attract nodes from its children *VNs* and moves them in the father node to reconstitute its “safe” size. This procedure can create a *cascade* effect such that nodes are progressively moved from the leaf *VNs* to the upper levels. As we will show in the following, the  $N_{min}$  threshold is a function of the maximum allowed churn rate and, intuitively, is 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. New nodes joining the system can, in principle, be accommodated in any *VN* (a join, in fact, cannot negatively impact IV). However, given that *VNs* are maintained as full graphs, in order to reduce the overhead incurred in their maintenance, the OMP allows up to a maximum of  $N_{max}$  nodes in a *VN*. A *VN* that reaches this bound will delegate the acceptance of newly joining nodes to one of its children *VN* (or will create a new one if needed) as these have a higher probability of containing less nodes.

**In-network query processing algorithm:** the query processing algorithm we propose is a simple adaptation of a broadcast/convergecast approach with partial result aggregation, modified to run on the *VT* graph topology. We assume that all queries are started by the root *VN* (either spontaneously or via delegation from other nodes) that broadcasts 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*

---

<sup>2</sup>In the following, unless otherwise stated, we will refer to a *physical node* with the generic term *node* and to a *virtual node* with the acronym *VN*

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 physical nodes that remained in the system for the whole query duration and will thus comply with the IV semantics.

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

### 3.1 The Virtual Tree Graph

Let us first formally define the structure of a *VT* graph:

**Definition 2 (Virtual Tree graph)** *Let  $\mathcal{VN}$  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 graph if:*

1.  $\mathcal{V} = \bigcup_{VN_i \in \mathcal{VN}} V_i$
2.  $\mathcal{E} = (\bigcup_{VN_i \in \mathcal{VN}} E_i) \cup \mathcal{VL}$ .
3. *Each virtual node  $VN_i$  is associated to an integer defining its level in the VT graph*
4. *Only one virtual node is at level 0 and we call such virtual node root*
5. *Given two virtual nodes  $VN_i$  and  $VN_j$  then  $V_i \cap V_j = \emptyset$*
6. *Every virtual node  $VN_j$  at level  $i$  is connected through a virtual link  $VL_{j,k} \in \mathcal{VL}$  to one single virtual node  $VN_k$  at level  $i - 1$ . 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 identify on it paths connecting any two virtual nodes:

**Definition 3 (Virtual Path on a Virtual Tree graph)** *Let  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  be a VT graph. Let  $\mathcal{VN}$  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  $VN_j$  belonging to  $\mathcal{VN}$ , 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 processes  $p_i$  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 processes  $p_i$  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  $VN_j$ . In the following, we will show the relationship existing between a virtual path  $\mathcal{P}_{i,j}$  and the number of disjoint paths between any pair of processes  $p_i$  and  $p_j$  belonging respectively to  $VN_i$  and  $VN_j$ .

**Lemma 1** *Let  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  be a *VT* graph and  $\mathcal{P}_{i,j}$  the virtual path between  $VN_i$  and  $VN_j$ ; let  $VN_{min}$  be the virtual node in  $\mathcal{P}_{i,j}$  with the minimum number of processes. For any pair of processes  $p_i, p_j$  belonging respectively to  $VN_i$  and  $VN_j$  there exists at least  $|VN_{min}|$  disjoint paths connecting  $p_i$  and  $p_j$ .*

Due to the lack of space, all the proofs are reported in Appendix A.

### 3.2 Overlay Maintenance Protocol

In this section we discuss the two main algorithms which compose our maintenance protocol: *join* and *maintenance*. The algorithms are based on the assumption that each process is able to maintain its local view updated and consistent with all the other processes belonging to its VN and to its father and children VNs. In the following, we will provide the description for both algorithms.

**The Join Algorithm.** When a new process  $p$  wants to join the overlay, it obtains an access point  $p_{ap}$  already part of the system from the bootstrap service. Afterwards,  $p$  sends a join request message to  $p_{ap}$  and then waits. When  $p_{ap}$  delivers the join request, depending on its size, it can decide to admit  $p$  in the virtual node or forward the request to a child virtual node. In particular:

- if the size of  $p_{ap}$  virtual node is smaller than  $N_{max}$ , then  $p$  is accepted and  $p_{ap}$  sends back an acceptance message containing all the information concerning the virtual node it belongs to, its father virtual node, all the children virtual nodes (needed by  $p$  to join all the groups containing  $p_{ap}$ ) and also all the information related to concurrently running queries;
- if the size of  $p_{ap}$  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 containing only  $p$ .  $K$  is a configuration parameter defining the maximum number of children that a

virtual node will create. Note that, the number of children impacts the total number of edges in the VT graph; thus, the value of  $K$  has to be defined according to the cost of the view-maintenance algorithm used to manage virtual nodes.

Note that due to the dynamics induced by churn, the join procedure could possibly not terminate (e.g. because the access point leave the system before placing  $p$  in a virtual node). To this aim, starting the join operation, it sets a timeout and if it does not receive an answer, it will issue again the procedure on a different access point.

The join procedure can be enhanced by applying an heuristic based on a locality principle (e.g. latency) to make sure that nodes participating to a same  $VN$  experience reciprocal communication latencies that on average are below a given threshold. In this case a joining node would be given multiple possible access points and will join the  $VN$  whose nodes are “closer” (locality-wise) to it. The rationale behind this choice is that locality helps the correct and efficient functioning of the view-maintenance algorithm used within the  $VN$ .

**The Maintenance Algorithm.** The maintenance algorithm is continuously running and when processes belonging to a  $VN$  detects that its size is below the threshold  $N_{min}$ , they take the following steps:

1. each process  $p_i$  selects, according to a deterministic rule,  $N_{min} - |V|$  processes, called *helpers*, among all those belonging to children  $VNs$ . Afterwards,  $p_i$  sends a HELP message to each helper process; such a message contains all the information about  $VN$ , its father and children, to let the *helper* process move in  $VN$ .
2. delivering a HELP message, a process  $p_j$  updates its local view according to the information received.

Note that, the maintenance procedure is always running and helper processes continue to be selected until the virtual node size returns to be larger than  $N_{min}$ .

**Virtual Tree Overlay Network Connectivity.** In the following, we will show that given the upper bound  $T_{move}$  on the time needed to let a process move from a  $VN$  to its father, and assuming the out-churn below of a certain threshold, the VT graph is connected.

**Lemma 2** *Let  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  be a VT graph at time  $t_0$ . If, for any time  $t$ ,  $\sum_{i \in [t, t+T_{move}+1]} \mu(i) < N_{min}/N_0$ , then the overlay network is connected.*

**Virtual Path Stability.** In the following, we will show that assuming a churn level (and in particular the out-churn level) below a certain threshold, the Virtual Tree OMP is able to maintain stable virtual paths. Let us recall that a virtual path between two VNs is stable if the sequence of VNs defining the path does not change.

**Lemma 3** *Let  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  be a VT graph at time  $t_0$ . If, for any time  $t$ ,  $\sum_{i \in [t, t+T_{move}+1]} \mu(i) < N_{min}/N_0$ , then there always exists at least one stable virtual path among any pairs of virtual nodes.*

*Property 3:* Let  $p_i$  be a process belonging to a virtual node  $VN_i$  at level  $x$  and  $p_r$  be a node belonging to the root virtual node  $VN_r$ . If  $p_i$  moves from  $VN_i$  to  $VN_j$  then the level of  $VN_j$  is  $x - 1$ . As a consequence, the length of the shortest path between  $p_i$  and  $p_r$  decreases to  $x - 1$ .

**Lemma 4** *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 belonging respectively to  $VN_i$  and  $VN_k$ . 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  $p_k$  does not change for at least  $2\delta$  time units.*

### 3.3 Virtual Tree Query Protocol

The basic idea of the query protocol is first to propagate the query from the root virtual node to the leaves (*query broadcast phase*) and then to gather results level by level (*query convergecast phase*). More in details, let us describe the two phases:

**Phase 1: Query broadcast phase.** This phase is started from the process  $p_i$  that issues the query. For easy of presentation, let us assume that  $p_i$  belongs to the root virtual node<sup>3</sup>. When a query  $q$  is issued by  $p_i$  the following steps occur:

1.  $p_i$  takes a snapshot of processes belonging to its virtual node and takes also a snapshot of the virtual nodes at level 1.
2.  $p_i$  sends a QUERY( $Q$ ) message to every process belonging to its virtual node (including itself) and to all the processes it knows belonging to virtual nodes at level 1.

---

<sup>3</sup>Otherwise, a simple forwarding procedure can be employed to delegate one process in the root virtual node.

3.  $p_i$  waits until (i) it receives values from any process in its virtual node and (ii) at least one aggregate value for each child virtual node contained in the snapshot.
4. When a process  $p_j$ , belonging to a VN at some level  $i$  receives the  $\text{QUERY}(Q)$  message, it iterates the steps 1, 2 and 3 on the next level of the VT graph. In addition, receiving the  $\text{QUERY}(Q)$  message,  $p_j$  also sends its value to all the processes belonging to its own virtual node.

Note that the snapshot are always updated accordingly to the local view: if a node or a virtual node disappears from the system it also disappears from the snapshots.

**Phase 2: Query convergecast phase.** This phase starts when a process  $p_j$ , belonging to a leaf virtual node (at some level  $i$ ), exits from the wait condition. Unblocking from the waiting condition,  $p_j$  stores the contribution for the aggregate query collected in its virtual node. Then, it executes the query on this set and sends a  $\text{QUERY\_REPLY}(q, res)$  message, containing such partial result, to all the processes belonging to the father virtual node at level  $i - 1$ . The procedure is iterated at each level of the VT graph and the query terminates when processes at the root virtual node unblock from the wait statement.

Note that during the query execution attractors can move physical nodes. It can happen that during a migration a node can miss the broadcast phase, for this reason when a physical node moves it keeps listening to the messages exchanged in its old VN until all the pending procedure are not completed.

This simple query algorithm allows to define a termination condition and, together with the connectivity ensured by the Virtual Tree, is also able to achieve interval validity. The claim is stated by the following theorem:

**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 interval validity semantics.*

## 4 Connectivity Evaluation

In this section we evaluate the Virtual Tree through simulations. The usage of a simulator was dictated by the need to test the proposed structure in large scale scenarios. Simulations were focused on evaluating the Virtual Tree behaviour in dynamic settings.

### 4.1 General Settings

The Virtual Tree has been implemented in a round-based simulator. The simulator simply models concurrent activities among participating processes

and message transmission delays. A round represents the minimal indivisible unit of time in the simulation. All the tests start from an initial configuration with  $N = 13500$  processes arranged in a complete Virtual Tree graph (i.e. each virtual node, except the leaves, has the same number  $k = 4$  of children<sup>4</sup>). Each virtual node in the graph is composed by  $N_{max}$  processes.

When the simulation is run the Virtual Tree graph is stressed with churn that last for the entire test duration (1000 rounds). Churn is modeled as a continuous periodic triangle-shaped process: during the first half period processes enter the system (*growing phase*), while processes are abruptly 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 [7]. The churn level  $c \in [0, 1]$  represents the percentage of processes that join/leave the system in each round: during the growing (shrinking resp.) phase  $cN$  processes invoke a join (leave) operation at each round. With reference to section 1 in the growing phase  $\lambda(t) = c$  and in the shrinking phase  $\mu(t) = c$ .

During the tests several metrics were taken into account: (i) *correct tests* represents the percentage of runs completed with a single connected component (i.e. no partitioning in the Virtual Tree graph), (ii) *Virtual Tree height* measures path lengths between the root and leaf nodes (maximum value), (iii) *virtual node size* represents the average size among all the virtual nodes, (iv) *message overhead* reports the average number of messages exchanged to move processes among virtual nodes.

Tests have been performed by varying the values for  $N_{min}$  and  $N_{max}$  (reported in the graphs with the form  $vn(N_{min}, N_{max})$ ). All values are the result of 10 independent runs. Standard deviations are not shown as their values were always smaller than 5%.

## 4.2 Connectivity Evaluation

In this subsection we evaluate the Virtual Tree connectivity. Note that thanks to Theorem 1 it is possible to assume that having a connected Virtual Tree is a sufficient condition to obtain interval valid queries. However, some preliminary tests with the same settings proposed in the following were run in order to verify the interval validity of the queries. The results were quite trivial: the query execution time was variable but all of the queries eventually terminated satisfying the property, in fact, the Virtual Tree OMP is able to *deterministically*<sup>5</sup> maintain a connected graph under a given threshold and *probabilistically* maintain the connectivity for a fairly larger amount of churn without disconnection.

Figure 1.2 reports the percentage of correct test vs. the churn rate. Different curves show results for different configurations. The graph shows

---

<sup>4</sup>Tests were also run for different settings of  $k$  without sensible differences in the results.

<sup>5</sup>see 2 in Appendix A

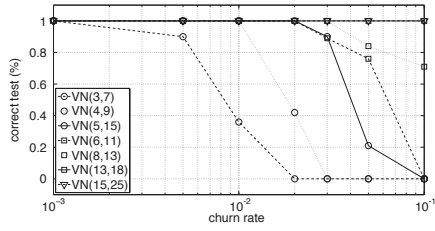


Figure 3: Graph connectivity vs. churn level.

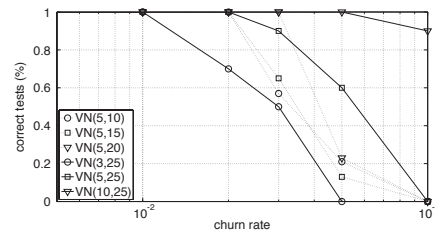


Figure 4: Impact of  $N_{min}$  on connectivity.

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 the Virtual Tree graph connectivity probability quickly drops to 0.

Note that the threshold is way larger than the limit for deterministic connectivity, in fact, if we consider the curve for  $vn(2, 7)$ , the *probabilistic* churn limit is  $c = 10^{-3}$ , conversely with the same settings the *deterministic* limit inferred by the formula proposed in Lemma 2 is about two orders of magnitude smaller.

Figure 1.3 reports test that try to explain the role played by  $N_{min}$  and  $N_{max}$  in the Virtual Tree OMP performance. The graph reports the percentage of correct test vs. the churn rate. The three solid curves show the algorithm behaviour by varying  $N_{min}$  only: it closely resembles the behaviour already reported in Figure 1.2. Conversely, the three dotted curves show the behaviour by varying  $N_{max}$  only: 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 1.1 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.

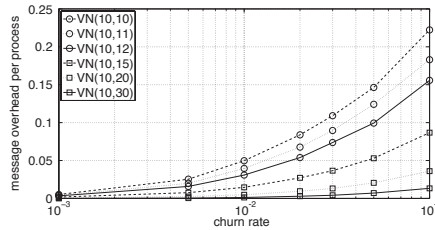


Figure 5: Impact of  $N_{max}$  on overhead generated by the Virtual Tree OMP.

The impact of  $N_{max}$  on performance is related to the amount of process exchanges among virtual nodes that is performed at runtime when new nodes are added. Figure 1.4 reports the message overhead per process versus the churn level for different  $N_{max}$  values. The message overhead 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  $N_{max} - N_{min}$  deltas: this behaviour is justified by the fact that the smaller is the delta and the larger is the probability to have a size smaller than  $N_{min}$ . The extreme case is represented by the configuration  $vn(10,10)$  that forward all process joins toward the leaf virtual nodes and, for every leave, attracts a node from the children.

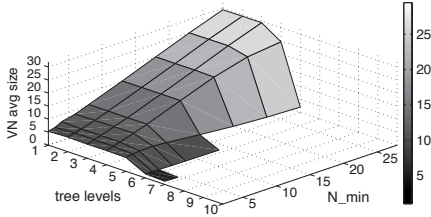


Figure 6: Virtual node sizes at different levels of the Virtual Tree graph.

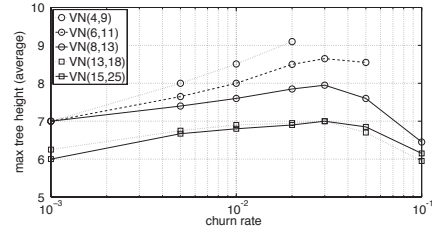


Figure 7: Maximum Virtual Tree height vs churn level.

The ability of the Virtual Tree OMP to move the effect of churn toward the leaf virtual nodes is shown in Figure 1.5. The graph reports the average virtual node sizes at different levels of the Virtual Tree for several  $N_{min}$  values with  $c = 10^{-2}$ . The surface shows how virtual nodes always maintain a stable size that is slightly larger than  $N_{min}$ ; the only exception is represented by nodes positioned at the lower levels of the tree that with high probability are leaves: these nodes cannot attract processes from child nodes, and are thus condemned to sizes that are way below the  $N_{min}$  value.

Figure 1.6 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 reach a maximum and starts to decrease for high churn levels. This can be explained by the fact that more leaves are added to the Virtual Tree graph as churn increases; however, when churn is particularly large this effect is counterbalanced by node removals that tend to severely affect leaf virtual nodes.

## 5 Related Work

**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. Several algorithms have been proposed to provide query answers but most of them provide only best effort semantics [8], [9], [10] that is, the algorithm does its best to gather all the values maintained in the network. In [2], Bawa et Al. propose three different semantics, namely *snapshot validity*, *interval validity* and *single-site validity*, that define when an aggregate query returns a *valid* answer. 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 this paper, we have shown that defining a bound on the churn rate and arranging processes into a defined structure, it is possible to deterministically answer interval validity queries.

In [11] Baldoni et al. define one further semantics, namely *dynamic validity*, to take into account churn (due to both join and leave operations); the same work provide an aggregate query answering algorithm supporting such semantics.

More recently, Payton et al. in [12] analyzed the query answering problem in dynamic networks introducing slightly different validity properties proposing an algorithm able to recognize the strongest one satisfied by each query answer. A similar work was previously proposed in [13] where the author try to measure a metric able to state if the current network status is able to provide valid queries.

**Virtual Nodes in dynamic systems.** Different works [14], [15], [16], [17] leveraged virtual nodes instead in order to improve the robustness of the network.

In [14], the authors introduce the notion of *Virtual Mobile Node*, while in [17], the authors use a virtual node to tolerate byzantine behaviors.

In Overnesia [15] the authors use virtual nodes, called *super peers*, defined as small cliques of processes. Given two super peers, they are connected through multiple probabilistic links. Differently from our proposal Overnesia was not designed to provide deterministic connectivity and path stability; its employment for query answer would thus show problems similar to those that can be encountered with other robust overlay network topologies. In [16] we introduced a tree-based overlay network structure to support aggregate queries execution satisfying interval validity with high probability, in a distributed system prone to continuous churn. The difference between the algorithm proposed in [16] and Virtual Tree lies in the fact that the first is purely probabilistic in its nature while the second is able to provide deterministic guarantees when churn is bounded.

Virtual Tree can be considered part of the family of the cluster based overlay. Some example are represented by [18] and [19]. The first protocol

namely eQuus is very close to our approach, in fact it makes use of small clique of nodes fully connected with the others, in eQuus the authors use this idea in order to enhance the DHT performance. eQuus differs from Virtual Tree by (i) the *virtual* architecture and the absence of attractors procedure, in fact it uses simple operation like merge and split. These operations are not suitable for our architecture due to fact that the merge operation introduces a change in the existing virtual paths violating of path stability. PeerCube [19] builds and maintains an hypercube of nodes substituted by cliques, even in this case the cliques are merged and split. Moreover, in order to enhance the overhead load, the management messages are handled by a well-known subset of nodes of the group instead of the whole clique. Recently in [20] the authors make use of a PeerCube-like structure in order to isolate targeted attacks.

## 6 Conclusions

Deterministically providing interval valid queries in unstructured distributed systems with unbound churn is impossible due to the lack of connectivity and path stability between processes. This paper presented Virtual Tree, an architecture comprising a 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 issued on the top of a Virtual Tree graph.

As future work we are adapting the Virtual Tree structure to large scale complex infrastructures with the aim of searching and controlling software and hardware resources (cpu, memory, virtual machines etc) with specific features within interconnected data-centers. In this case nodes of the overlay network represents software or hardware resources, and churn is due to the volatility of such resources whose availability depends both on the current status of hardware device hosting them and on the load incurred by the system.

## References

- [1] P. Bonnet, J. Gehrke, and P. Seshadri, “Towards sensor database systems,” in *Proceedings of the Second International Conference on Mobile Data Management*, ser. MDM '01. London, UK: Springer-Verlag, 2001, pp. 3–14. [Online]. Available: <http://portal.acm.org/citation.cfm?id=648058.746944>

- [2] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani, “The price of validity in dynamic networks,” *Journal of Computer and System Sciences*, vol. 73, no. 3, pp. 245–264, 2007.
- [3] S. Saroiu, K. P. Gummadi, and S. D. Gribble, “A measurement study of peer-to-peer file sharing systems,” in *Multimedia Computing and Networking (MMCN)*, January 2002.
- [4] R. Baldoni, S. Bonomi, A. Cerocchi, and L. Querzoni, “Virtual tree: a robust overlay network for ensuring interval valid queries in dynamic distributed systems,” MIDLAB 4/2011 - Università degli Studi di Roma “La Sapienza”, Tech. Rep., 2011.
- [5] M. Merritt and G. Taubenfeld, “Computing with infinitely many processes,” in *Proceedings of the 14th International Conference on Distributed Computing*, ser. DISC ’00. London, UK, UK: Springer-Verlag, 2000, pp. 164–178.
- [6] M. Jelasity, A. Montresor, and O. Babaoglu, “The bootstrapping service,” in *Proceedings of the International Workshop on Dynamic Distributed System (IWDDS)- ICDCS Workshops*, 2006, p. 11.
- [7] D. Stutzbach and R. Rejaie, “Understanding churn in peer-to-peer networks,” in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, 2006, pp. 189–202.
- [8] 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*, ser. ICDE ’04. IEEE Computer Society, 2004.
- [9] I. Gupta, R. van Renesse, and K. P. Birman, “Scalable fault-tolerant aggregation in large process groups,” in *Proceedings of the 2001 International Conference on Dependable Systems and Networks (formerly: FTCS)*, ser. DSN ’01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 433–442.
- [10] 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*, vol. 36, pp. 131–146, December 2002.
- [11] R. Baldoni, M. Bertier, M. Raynal, and S. Tucci Piergiovanni, “Looking for a Definition of Dynamic Distributed Systems,” in *9th International Conference on Parallel Computing Technologies (PaCT’07)*, LNCS, 1-14, S. V. LNCS, Ed., 9 2007.

- [12] J. Payton, C. Julien, G. C. Roman, and V. Rajamani, “Semantic self-assessment of query results in dynamic environments,” *ACM Trans. Softw. Eng. Methodol.*, vol. 19, pp. 12:1–12:33, April 2010.
- [13] N. Jain, P. Mahajan, D. Kit, P. Yalag, M. Dahlin, and Y. Zhang, “Network imprecision: A new consistency metric for scalable monitoring,” in *OSDI*, 2008.
- [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 *DISC04*, 2004, pp. 230–244.
- [15] J. Leitão and L. Rodrigues, “Overnesia: a resilient overlay network for virtual super-peers,” INESC-ID, Tech. Rep. 56, December 2008.
- [16] R. Baldoni, S. Bonomi, A. Cerocchi, and L. Querzoni, “Improving Validity of Query Answering in Dynamic Systems,” in *WRAS '10 Proceedings of the Third International Workshop on Reliability, Availability, and Security*. Zurich, Switzerland: ACM, 7 2010, pp. 4:1–4:6.
- [17] I. Eyal, I. Keidar, and R. Rom, “Distributed clustering for robust aggregation in large networks,” in *HotDep09*. IEEE, 2009.
- [18] T. Locher, S. Schmid, and R. Wattenhofer, “equus: A provably robust and locality-aware peer-to-peer system,” in *Peer-to-Peer Computing, 2006. P2P 2006. Sixth IEEE International Conference on*, sept. 2006, pp. 3–11.
- [19] A. E., R. Ludinard, A. Ravoaja, and F. Brasileiro, “Peercube: A hypercube-based p2p overlay robust against collusion and churn,” in *Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 15–24. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1475691.1475938>
- [20] E. Anceaume, B. Sericola, R. Ludinard, and F. Tronel, “Modeling and evaluating targeted attacks in large scale dynamic systems,” in *DSN 2011 - the 41th annual ieee/ifip International Conference on Dependable Systems and Networks*, June. 2011.

## Appendix A - Proof omitted in the text

**Lemma 1** *Let  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  be a VT graph and  $\mathcal{P}_{i,j}$  the virtual path between  $VN_i$  and  $VN_j$ ; let  $VN_{min}$  be the virtual node in  $\mathcal{P}_{i,j}$  with the minimum number of processes. For any pair of processes  $p_i, p_j$  belonging respectively to  $VN_i$  and  $VN_j$  there exists at least  $|VN_{min}|$  disjoint paths connecting  $p_i$  and  $p_j$ .*

**Proof** (Sketch) If  $VN_i$  and  $VN_j$  are directly connected through a virtual link  $VL_{i,j}$  (i.e.  $\mathcal{P}_{i,j}$  has length 1), then there exists a single path of length 1 (represented by the edge  $(p_i, p_j)$ ) and  $|V_i| + |V_j| - 2$  disjoint paths of length 2 (i.e. a path  $p_i, v, p_j$  for any  $v \neq p_i \in V_i$  and a path  $p_i, u, p_j$  for any  $u \neq p_j \in V_j$ ).

If  $\mathcal{P}_{i,j}$  has length 2, it means that there exists a virtual node  $VN_k$  between  $VN_i$  and  $VN_j$  on  $\mathcal{P}_{i,j}$ . Thus, there exist  $|V_k|$  disjoint paths of length 2 (i.e. a path  $p_i, v, p_j$  for any  $v \in V_k$ ) and  $((|V_i| - 1) \cdot |V_k|) + ((|V_j| - 1) \cdot |V_k|)$  disjoint paths of length 3. The claim follows simply iterating the above reasoning.

□*Lemma 1*

**Lemma 2** *Let  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  be a VT graph at time  $t_0$ . If, for any time  $t$ ,  $\sum_{i \in [t, t+T_{move}+1]} \mu(i) < N_{min}/N_0$ , then the overlay network is connected.*

**Proof** (Sketch) To guarantee connectivity, we need to maintain a path between any pair of processes.  $\mathcal{G}$  can be partitioned if and only if all the processes belonging to a non leaf virtual node leave the system and no new process replaces them (either a joining process or a process moving from a children virtual node). However, the maintenance algorithm is able to move processes from one virtual node to another (i.e., from a child virtual node to its father) before the virtual node size reaches 0. Let assume the worst case scenario where the out-churn affects a single virtual node while the in-churn is 0. Without loss of generality, let  $VN_i$  be the first virtual node whose size reaches the value  $N_{min}$  and let  $t$  be the time when this is detected by processes belonging to  $VN_i$ . Note that, at time  $t - 1$ ,  $|V_i|$  is at least  $N_{min} + 1$ . Let us denote as  $V_i(t)$  the set of processes belonging to  $VN_i$  at time  $t$  and let  $|V_i(t)| \geq N_{min} + 1 - (\mu(t) \cdot N_0)$ .

At time  $t$ , each process belonging to  $VN_i$  selects, through a deterministic function,  $N_{min} - |V_i(t)| + 1$  helper processes among those belonging to  $VN_i$  children, and ask them to move in  $VN_i$ . Given  $T_{move}$ , the selected helper processes will complete their movement, at time  $t + T_{move}$  in the worst case. 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  $VN_i$  decreases at most to  $N_{min} + 1 - N_0 \sum_{i \in [t, t+T_{move}]} \mu(i)$ ). To avoid

disconnections, we must guarantee that during  $T_{move}$ , at least one process remains in  $VN_i$  despite the out-churn. In the worse case, all processes leaving at time  $t + 1$  are all the helper processes selected at time  $t$ . However, the maintenance algorithm will continue to select new helper processes, until  $|V_i| > N_{min}$ . 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.  $\square_{Lemma 2}$

**Lemma 3** *Let  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  be a VT graph at time  $t_0$ . If, for any time  $t$ ,  $\sum_{i \in [t, t+T_{move}+1]} \mu(i) < N_{min}/N_0$ , then there always exists at least one stable virtual path among any pairs of virtual nodes.*

**Proof** The proof trivially follows by considering that (i) from Lemma 2 the graph is connected and (ii) at time  $t_0$ ,  $\mathcal{G}$  is a VT graph. As a consequence, no virtual node, except the leaf ones, can disappear from the VT graph. Therefore, for any two virtual nodes the virtual path connecting them never changes and thus claim follows.

$\square_{Lemma 3}$

**Lemma 4** *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 belonging respectively to  $VN_i$  and  $VN_k$ . 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  $p_k$  does not change for at least  $2\delta$  time units.*

**Proof** The proof trivially follows by considering that by Lemma 2 and Lemma 3, there exists at least one process in  $VN_j$  that does not leave until an helper process completed the movement toward  $VN_j$ .  $\square_{Lemma 4}$

**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 interval validity semantics.*

**Proof** (Sketch) **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( $q, res$ )` 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 the interval validity semantics, 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 and its contribution is not considered in the query final result. Let  $p_i$  be the process that issued  $q$  and, without loss of generality, let us assume that  $p_i$  belongs to the root virtual node (i.e.  $p_i \in V_r$ ). If the contribution of  $p_j$  has not been considered, two cases could be happened: (i)  $p_j$  has never received the query or (ii) the contribution of  $p_j$  has never been received by  $p_i$ .

**Case 1:  $p_j$  never receives  $q$ .** If  $p_j$  does not receives  $q$ , it means that the  $\text{query}(q)$  message has been lost while travelling from  $p_i$  to  $p_j$ . Due to the connectivity, there always exists at least one path connecting  $p_i$  and  $p_j$ . Considering the shortest path between such two 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  $p_i$  leaves or  $p_j$  leaves. In the first case, the query initiator left and thus the query does not terminate and we have a contradiction. In the second case, it means that  $p_j$  leaves before the query ends but in this case its contribution is not mandatory and the query is interval valid bringing again to a contradiction

- In the latter case, there exists at least one virtual node  $VN_j$  on the path between  $VN_r$  and  $VN_i$ . 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  $VN_i$ . 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 receives the query. As a consequence, due to the fact that the nodes can only move from the children to the father, i.e.  $p_j$  can only reduce the distance<sup>6</sup> to  $p_i$ , at any time the query moves along the virtual path always closer to its destination, and thus this is true on all disjoint paths connecting two arbitrary nodes belonging respectively to  $VN_1$  and  $VN_3$ .

Considering that the virtual path between  $VN_r$  and  $VN_i$  can always be decomposed in sub-paths of length 2, it is possible to iterate the reasoning above and we have then a contradiction.

**Case 2:  $p_i$  never receives the contribution sent by  $p_j$ .** This case

---

<sup>6</sup>for a more formal proof please refer to Lemma 4 in Appendix Appendix B

trivially follows from the previous reasoning by substituting  $p_i$  and  $p_j$ .  
□*Theorem 1*