Abstract

Distributed algorithms in dynamic networks often employ communication patterns whose purpose is to disseminate information among the participants. Gossiping is one form of such communication pattern. In dynamic settings the set of participants can change substantially as new participants join, and as failures and voluntary departures remove those who have joined previously. A natural question for such settings is: how soon can newly joined nodes discover each other by means of gossiping? This paper abstracts and studies the Join Problem for dynamic systems that use all-to-all gossip. The problem is studied in terms of join-connectivity graphs where vertices represent the participants and where each edge represents one participant’s knowledge about another. Ideally, such a graph has diameter one, i.e., all participants know each other. The diameter can grow as new participants join, and as failures remove edges from the graph. Gossip helps participants discover one another, decreasing the diameter. The results describe the lower and upper bounds on the number of communication rounds such that the participants who have previously joined discover one another, under a variety of assumptions about the joining and failures. For example, in the case when new participants join at multiple participants and participants may crash, the number of rounds cannot be bounded. In the more benign cases when the failures can be controlled or when new participants join at only one participant, the bound on rounds is shown to be logarithmic in the diameter of the initial configuration.

1. Introduction

In order to be deployable over dynamic networks, distributed algorithms must include facilities for new participants to join the computation and to discover each other. There are two immediate distinct challenges that must be faced in joining a dynamic computation: (a) discovering a system access point allowing new participants to join, and (b) joining the computation and acquiring information about the computation. This work studies the latter problem of joining the computation, when the former problem, of finding where to join the computation, is solved.

In a realistic networked system, the knowledge describing the state of a distributed computation is inherently distributed. For example, in systems with dynamic participation, it may be impossible to know globally and precisely who the participants are. This is especially true if new participants can join the computation at arbitrary times. Thus the safety of sophisticated distributed algorithms must not depend on the global knowledge about the participants. At the same time, the system performance concerns may be well served by participants knowing about each other. This motivates our introduction and study of the Join Problem that we state as follows:

For any two participants that join a dynamic system, how long does it take for them to discover each other?

We study this question for distributed systems that use periodic all-to-all gossip whose purpose is to disseminate information among the participants. While all-to-all gossip may not be bandwidth-efficient, it is fault-tolerant and is the fastest form of gossip when bandwidth is of no concern. Furthermore, the understanding of the limitations of information propagation in all-to-all gossip patterns helps understanding gossip using more restricted communication patterns. We focus on an abstract model of the join algorithm, modelled after the join protocol of the reconfigurable...
atomic memory service for dynamic networks presented by Lynch and Shvartsman [11], and Gilbert et al. [4]. As is typical for dynamic systems, the performance of this memory service depends on how quickly the new participants become integrated.

The problem is studied in terms of join-connectivity graphs where vertices represent the participants and where each edge represents one participant’s knowledge about another. Ideally, such a graph has diameter one, i.e., all participants know each other. The diameter can grow as new participants join, and as failures remove edges from the graph. Gossip helps participants discover one another, decreasing the diameter. The results describe the lower and upper bounds on the number of communication rounds such that the participants who have previously joined discover one another, under a variety of assumptions about the joining and failures. We now review our contributions in detail.

Summary of contributions. In this paper we study the Join Problem with the help of an abstract join algorithm that we call the Join-Protocol (Section 2). We use the algorithm to study the upper and lower bounds of the Join Problem under a variety of scenarios.

In our model, nodes asynchronously join the system and leave the system (by failing), while the knowledge of the participants about one another is represented by a dynamically evolving communication graph, called the join-connectivity graph (Section 2.2). A new node joins the system by sending a request to join at one or more nodes that are already participating. The knowledge of the joined nodes about each other is propagated by means of perpetual gossip with known nodes.

We formally state the Join-Protocol (Section 2) using the Input/Output Automata [12] notation. The protocol is specified as a non-deterministic algorithm for asynchronous environments with arbitrary message delays and node crashes that may cause network partitions. In such environments it is not possible to quantify how information is propagated throughout the “known universe.” For the purpose of analysis, we restrict asynchrony, resolve some of the non-determinism of the algorithm, and impose constraints sufficient to guarantee that the universe is connected.

We restrict our attention to the timed executions of the protocol, in which the initial period of arbitrary instability is followed by a period of stability. We call such executions normal. Starting with admissible timed executions (that allow time to pass to infinity), we instrument the executions with states of interest, called the milestone states, by means of time-passage refinements. In order to make plausible a performance analysis, for the stable suffix of an execution, we introduce the notion of a timed communication round (Section 3).

Given a normal execution and a state \( s \) in the stable suffix of the execution, we define the join-delay \( jd(s) \) to be the maximum number of rounds required for any two nodes that have joined the computation by state \( s \) to learn about each other (Section 3.2). For each state \( s \) we define the join-connectivity graph \( s.G = (s.J, s.E) \), where \( s.J \) is the set of nodes that have joined, and the edges in \( s.E \) represent the nodes’ knowledge about each other. Since we are interested in results for networks that do not partition, it is sufficient to assume that these graphs are connected. We study join-delay for the cases with and without node failures, and the cases where new participants join at only one node or at multiple nodes. Thus we consider four distinct classes of executions (Section 4). Our result are as follows.

1. Joining at a single node in the absence of failures: For any state \( s \) in the stable segment of a normal execution, \( jd(s) = \Theta (\log (s.G.diam)) \), where \( s.G.diam \) is the diameter of the join-connectivity graph. (\( \Theta \) notation here specifies upper and lower bounds defined to be the minimum over all algorithms of the maximum join-delay in all relevant executions.)

2. Joining at multiple nodes in the absence of failures: For any state \( s \) in the stable segment of a normal execution, \( jd(s) = \Theta (\log (s.G.diam)) \). This is similar to the case (1), but the analysis is different.

3. Joining at a single node in the presence of failures: For any state \( s \) in the stable segment of a normal execution, we show an upper bound of \( jd(s) = O (\log (|s.J|)) \), and a lower bound of \( jd(s) = \Omega (\log (s.G.diam)) \).

4. Joining at multiple nodes in the presence of failures: We show that an adversary can force join-delay to be unbounded even in normal executions. Thus, in order to get an upper bound we constrain the adversary so that the rate at which it fails nodes in certain regions of the join-connectivity graphs is not more than exponential in the number of rounds. In this case, for any state \( s \) in the stable segment of a normal execution, we show an upper bound of \( jd(s) = O (\log (s.G.diam)) \).

Note that, surprisingly, joining at multiple nodes in the presence of failures leads to worse (unbounded) join-delay as compared to joining at a single node. The reason for this is that when joining at multiple nodes the system can tolerate larger patterns of node failures since we require that join-connectivity graphs remain connected. However this improved fault-tolerance can cause unbounded join-delays. Constraining the adversary allows to bound join-delays.

Relevant landscape. Gossip and broadcast are among the basic communication problems (e.g., [5, 14, 6, 3]). The goal of broadcasting is to spread a message known at one node to all other known nodes, while in the case of gossiping we are interested in all nodes exchange their initial messages. The most interesting metrics used in comparing communication algorithms are time and message complexity, that
is, the number of elementary transmissions required by the communication process. The dynamic version of gossiping, called perpetual gossiping, was introduced by Liestman and Richards [9]. Here the new information is generated continuously and the goal is to update the received information, hence the gossiping-like protocol must be repeated.

A related problem is maintaining consistency among the sites in the face of updates in the replicated database. Demers et al. [2] developed randomized algorithms for distributing updates and driving the replicas toward consistency. They use epidemic-like approach to model and analyze the performance of designed protocols.

Peer-to-peer systems provide decentralized access to the stored information. In a dynamic peer-to-peer environment, the time required to disseminate new information can vary greatly owing to rapid changes in the membership of individual peers that perpetually join and leave the system. This dynamic nature of the peer-to-peer network also creates the problem of residual peers that do not receive updates before they leave or fail [13]. Cuenca-Acuna et al. [1] introduce PlanetP, a content addressable publish/subscribe service for unstructured peer-to-peer communities. The simplicity of PlanetP may be illustrated by the fact that each peer must only perform a periodic, randomized, point-to-point message exchange with other peers. Liben-Nowell et al. [8] addressed the issue of assumption of an ideal connected overlay network across the Internet that goes with many peer-to-peer protocols that uses key lookups. They pointed out that most of these approaches ignore the very fact that peer-to-peer networks are dynamically evolving systems. The authors develop a theoretical analysis of peer-to-peer networks in the presence of concurrent joins and unexpected failures with a focus on Chord [15].

Kumar et al. [7] proposed and analyzed the statistical properties of Internet-like dynamic topology. They considered the structures different than previously used random graphs, requiring that the adding of new units/links to the network does not change the stochastic properties of the dynamic structure.

Lynch, Shvartsman, and Gilbert [11, 4] proposed a reconfigurable atomic memory service for dynamic networks. Our Join-Protocol is based on their join algorithm. The performance of the memory service depends largely on the gossip protocol that goes on perpetually as a background process at every participating node. The analysis of performance of the service makes assumptions about join-delays in the system. Our new work establishes specific bounds that can be used in lieu of those assumptions.

**Document structure.** In Section 2 we define the join protocol and the join-connectivity graphs. We give the models and definitions used in our analysis in Section 3. Section 4 deals with the analysis of the protocol. Finally, we summarize our contribution in Section 5.

## 2. The Join Protocol

We abstract the join protocol from the reconfigurable atomic memory service of Lynch and Shvartsman [11]. This service, called RAMBO (Reconfigurable Atomic Memory for Basic Objects), implements atomic read/write memory. The objects are replicated at multiple nodes in an asynchronous dynamic network, where new nodes may join the service, and previously joined nodes may leave the system or fail. The implementation is specified using Input/Output Automata [12] formalism. Correctness (atomicity) is guaranteed for any patterns of asynchrony and failures. The efficiency of reads and writes depends on the nodes that join the system learning about each other. A new node joins the system by contacting any previously joined node. The knowledge of the joined nodes about each other is propagated by means of all-to-all gossip. The goal here is for each joined node to find out about all other joined nodes.

### 2.1. Description of the Join Protocol

We specify the behavior of each node \( v (v \in P) \) participating in the join protocol as an I/O automaton, called \( \text{Join-Protocol}_v \). The complete I/O Automaton specification of \( \text{Join-Protocol}_v \) is given in Figure 1, and it includes the signature, the state variables, and the transitions. We now discuss the specification. To disambiguate among the state variables of different automata when these variables appear outside of the scope of their definition, we use the shorthand \( v.\langle \text{state-variable} \rangle \) to refer to the state variable \( \text{Join-Protocol}_v.\langle \text{state-variable} \rangle \), i.e., the state variable \( \langle \text{state-variable} \rangle \) of \( \text{Join-Protocol}_v \). Also, for a state \( s \), we use the notation \( s.v.\langle \text{state-variable} \rangle \) to denote the value of \( v.\langle \text{state-variable} \rangle \) in that state.

The join protocol is simple. Initially, a distinguished single node \( c \) (the “creator”) constitutes the entire system. When some other node \( v \) receives join(\( H \)) request from its environment, it sends out join requests to the set of processes in \( H \), representing a reasonable guess of what nodes may have already joined. \( \text{Join-Protocol}_v \) then waits for acknowledgements.

Initially, \( v.\text{status} = \text{idle} \) in \( \text{Join-Protocol}_v \), for each \( v \in P - \{c\} \), which results in all of its output actions being disabled, and \( v.\text{failed} = \text{false} \), meaning that the node has not crashed. However, when \( \text{Join-Protocol}_v \) receives a join(\( H \)) request from its environment, it changes its status to joining and initializes its set \( \text{hints} \) to the set \( H \). We impose the well-formedness condition on the environment that \( H \) is a finite subset of \( P \). \( \text{Join-Protocol}_v \) then can send a join request to each node in \( H \) using action send(join)\(_{v,u} \), \( u \in H \). This happens as long as \( v.\text{failed} = \text{false} \) and \( v.\text{status} = \text{joining} \).
Data-types:
P, the set of node identifiers
c ∈ P, a distinguished node (the “creator”)

M, the set of messages

Signature:
Input:
join(J), J a finite subset of P − \{v\}
recv(join)u,v, u ∈ P − \{v\}
recv(m)u,v, m ∈ M, u ∈ P − \{v\}
failv

Output:
send(join)u,v, u ∈ P − \{v\}.
send(m)u,v, m ∈ M, u ∈ P − \{v\}

States:
status ∈ \{idle, joining, active\}, initially idle if c ≠ v,
else active if c = v

hints ⊆ P, initially ∅

Transitions:
Input join(J)
Effect:
if ¬failed then
if status = idle then
status ← joining
hints ← J

Output send(join)v,u
Precondition:
¬failed
status = joining
u ∈ hints

Input recv(join)u,v
Effect:
if ¬failed then
if status = active then
world ← world ∪ \{u\}

Input recv(W)u,v
Effect:
if status = joining then
status = active
if ¬failed then
world ← world ∪ W

Figure 1. Specification of the Join-Protocol at v

At the receiving end, the join request is received at some u ∈ P via its input action recv(join)u,v. If u.status = active, i.e., node u is already participating in the Join-Protocol, the recipient adds the sender v to its set world.

When Join-Protocolv has its v.status = active and v.failed = false, it sends out messages, consisting of its v.world, to any processes u in v.world via the action send(m)u,v. If Join-Protocolu has u.status = joining and u.failed = false, then in the input action recv(W)v,u, it sets its status to active and sets its world to world ∪ W.

The input action failv models the crash of node v. When the environment triggers this action, it results in v.failed = true, which disables all output actions at v, and that prevent any input actions from changing the state of v. The crashes are terminal, and nodes cannot recover.

The nodes participating in the algorithm communicate via point-to-point channels (not formally specified here). Messages are not corrupted and are not spontaneously generated by the channels, but the messages may be lost, be delivered in an arbitrary order, and be duplicated. We denote by Channelu,v the channel from node u to node v.

The full system is defined as the composition of all Join-Protocolv for v ∈ P, and all Channelu,v for u, v ∈ P. We denote the resulting composition by JOIN-SYSTEM.

2.2. The Join-Connectivity Graphs

In this paper we study properties of the executions of JOIN-SYSTEM with the help of graphs induced by the executions. We call these graphs join-connectivity graphs.

Definition 2.1 Let s be a state of an execution of JOIN-SYSTEM. The join-connectivity graph at s is a derived state variable, a directed graph s.G = (s.J, s.E), where

- s.J = {v ∈ P : s.v.status = active ∧ s.v.failed = false}, is the set of vertices of the graph representing the set of nodes that successfully joined and that have not failed.
- s.E = {(u, v) : u, v ∈ s.J ∧ v ∈ s.u.world}, is the set of edges.

Note that in the definition of the join-connectivity graph, an edge (u, v) in s.E models the fact that u “knows” v in
time-passage actions, \(\nu(t)\), \(t \in \mathbb{R}^+\). The time-passage action \(\nu(t)\) represents the passage of time by amount \(t\). A \textit{timed execution} of a GTA \(A\), is defined to be either a finite sequence \(\alpha = s_0, \pi_1, s_1, \pi_2, \ldots, s_r\) or an infinite sequence \(\alpha = s_0, \pi_1, s_1, \pi_2, \ldots,\) where the \(\pi\)'s are actions (either input, output, internal, or time-passage) of \(A\), the \(s\)'s are states of \(A\), \((s_k, \pi_{k+1}, s_{k+1})\) is a transition of \(A\) for every \(k\), and \(s_0\) is a start state. It is possible that \(s_k = s_{k+1}\). The set of transitions of \(A\) is denoted by \(\text{trans}(A)\).

If \(\alpha_1\) and \(\alpha_2\) are two execution fragments where the last state of \(\alpha_1\) is equal to the first state of \(\alpha_2\) then \(\alpha_1 \circ \alpha_2\) is the concatenation of \(\alpha_1\) and \(\alpha_2\). The common state appears only once.

A timed execution is called \textit{admissible} if the sum of all the reals in the time-passage actions in the execution is \(\infty\).

We are interested in making observations about the executions of JOIN-SYSTEM at states corresponding to certain real times. We now give the technical background leading to the definition of these states. There are two axioms regarding the time-passage action \(\nu(t)\) for a GTA \(A\):

(A1) If \((s, \nu(t), s')\) and \((s', \nu(t'), s'')\) are in \(\text{trans}(A)\), then \((s, \nu(t + t'), s'')\) is in \(\text{trans}(A)\).

(A2) If \((s, \nu(t), s')\) \(\in\) \(\text{trans}(A)\) and \(0 < t' < t\), then there is a state \(s''\) such that \((s, \nu(t'), s'')\) and \((s'', \nu(t - t'), s')\) are in \(\text{trans}(A)\).

A timed execution fragment \(\alpha\) is a \textit{time-passage refinement} of another time execution fragment \(\alpha'\) if they are identical except that, in \(\alpha\), some of the time-passage actions of \(\alpha'\) are replaced with finite sequences of time-passage steps, with the same initial and final states and the same total amount of passage of time, in accordance to axioms A1 and A2. So, two admissible timed executions \(\alpha\) and \(\alpha'\) are \textit{time-passage equivalent} if they have a common time-passage refinement.

In order to analyse the properties of an execution \(\alpha\) of the JOIN-SYSTEM we need to examine the states in the execution occurring at regular intervals of time, \(d > 0\). Since \(\alpha\) might not have states corresponding to the time of interest, we construct a time-passage equivalent execution of \(\alpha\), that we call a \textit{milestone execution}, and that includes the unique states of interest, that we call \textit{milestone} states. We show that such executions can indeed be constructed. First we introduce some notation.

**Definition 3.1** Let \(\alpha = s_0, \pi_1, s_1, \pi_2, \ldots, s_r, \ldots\) be an admissible timed execution. We define:

(i) \(\text{prefix}(\alpha, s_r)\) is the execution fragment of \(\alpha\) from state \(s_0\) to state \(s_r\),

(ii) \(\text{suffix}(\alpha, s_r)\) is the execution fragment of \(\alpha\) that starts with state \(s_r\),

(iii) \(\text{l-prefix}(\alpha, s_r)\) is the set of time-passage actions in \(\text{prefix}(\alpha, s_r)\).
Lemma 3.1 shows that given an admissible timed execution $\alpha$ and any $t' > 0$, there is a time-passage equivalent execution $\beta$ where we can choose a state such that the sum of the reals of the time-passage actions occurring in $\beta$ up to that state is $t'$.

**Lemma 3.1** Let $\alpha$ be an admissible timed execution of a GTA $A$. Then for any $t' > 0$ there exists an execution $\beta$, a time-passage refinement of $\alpha$ that is time-passage equivalent to $\alpha$, and a state $s_r$ in $\beta$ so that $\sum_{t \in t \text{-prefix}(\beta, s_r)} t = t'$, and if a state $s$ is such that $\sum_{t \in t \text{-prefix}(\alpha, s)} t < t'$ then $\text{prefix}(\beta, s) = \text{prefix}(\alpha, s)$.

**Proof.** Since $\alpha$ is an admissible timed execution, the time passes to infinity. So, there exists a state $s'$ in $\alpha$ such that $\sum_{t \in t \text{-prefix}(\alpha, s')} t \geq t'$ and for any other state $s$ that occurs before $s'$ in $\alpha$ we have $\sum_{t \in t \text{-prefix}(\alpha, s)} t < t'$, and let us choose $s$ so that it occurs in $\alpha$ among such states. In other words, we are choosing the earliest state $s'$ in $\alpha$, such that $\sum_{t \in t \text{-prefix}(\alpha, s')} t \geq t'$ is satisfied and $s$ is the last state in $\alpha$ where the inequality $\sum_{t \in t \text{-prefix}(\alpha, s)} t \leq t'$ holds. Thus we have

$$\sum_{t \in t \text{-prefix}(\alpha, s)} t \leq t' \leq \sum_{t \in t \text{-prefix}(\alpha, s')} t.$$

If either inequality is in fact equality, then we found the required state called $s_r$ in the alternative.

$$\sum_{t \in t \text{-prefix}(\alpha, s)} t < t' < \sum_{t \in t \text{-prefix}(\alpha, s')} t.$$

Then there exists a step $(s, (\nu(\tau), s'))$ in $\beta$ where $\tau = \sum_{t \in t \text{-prefix}(\alpha, s')} t - \sum_{t \in t \text{-prefix}(\alpha, s)} t$. By axiom A2 there exists a state $s''$ such that $(s, (\nu(\tau), s''))$ and $(s'', \nu(\tau - t''), s')$ are in $\text{trans}(A)$ where $t'' = t - \sum_{t \in t \text{-prefix}(\alpha, s)} t$. Now, replacing the step $(s, (\nu(\tau), s'))$ in $\alpha$ by steps $(s, (\nu(t''), s''))$ and $(s'', \nu(\tau - t''), s')$ we get the time-passage refinement and time-passage equivalent execution $\beta$. Lastly, since $\sum_{t \in t \text{-prefix}(\alpha, s)} t < t'$, we know that $s$ is either $s$ or have occurred in $\alpha$ before $s$, hence $\text{prefix}(\beta, s) = \text{prefix}(\alpha, s)$ as claimed in the lemma.

Lemma 3.2 enables us to define specific states in an execution that we later use to define rounds in the execution.

**Lemma 3.2** Let $\alpha$ be an admissible timed execution, $\sigma_0$ a state in $\alpha$, and $\tau = \sum_{t \in t \text{-prefix}(\alpha, \sigma_0)} t$. Then for every $d > 0$ there exists a time-passage equivalent execution $\beta$ of $\alpha$ containing a subsequence of states $\{\sigma_i\}_{i \geq 0}$ such that $\sum_{t \in t \text{-prefix}(\beta, \sigma_0)} t = kd + \tau$, for every $k \in \mathbb{N}$ (that is the sum of the reals of the time-passage actions of $\beta$ up to $\sigma_k$ is $kd + \tau$).

**Proof.** Suppose $\alpha$ is $s_0, \pi_1, s_1, \pi_2, \ldots, s_r, \ldots$. First, we want to show that given $d > 0$ and any state $\sigma_0$ in the admissible timed execution $\alpha$ we can construct an execution $\beta$ and the sequence $\{\sigma_i\}_{i \in \mathbb{N}}$ satisfying the condition stated in the lemma. We construct a sequence of time-passage equivalent (to $\alpha$) executions $\alpha = \alpha^{(0)} , \alpha^{(1)} , \ldots$ such that $\alpha^{(k+1)}$ is a time-passage refinement of $\alpha^{(k)}$ that satisfies the following properties:

1. $\sum_{t \in t \text{-prefix}(\alpha^{(k)}, \sigma_i)} t = i \cdot d$, for $i = 0, 1, \cdots, k$.
2. $\text{prefix}(\alpha^{(k)}, \sigma_k) = \text{prefix}(\alpha^{(k+1)}, \sigma_k)$ for $k \in \mathbb{N}$.

Let $\alpha^{(0)} = \alpha = \text{prefix}(\alpha, \sigma_0) \circ \text{suffix}(\alpha, \sigma_0)$. Suppose we have constructed $\alpha^{(0)}, \ldots , \alpha^{(k)}$. We use Lemma 3.1. Choosing $\sigma_k$ for $\bar{s}$ and $(k + 1)d + \tau$ for $t'$, we can find a time-passage refinement $\alpha'$ of $\alpha$ such that there is a state $s'$ in $\alpha'$ and $\sum_{t \in t \text{-prefix}(\alpha', s')} t = (k + 1)d + \tau$ and $\text{prefix}(\alpha', \sigma_k) = \text{prefix}(\alpha, \sigma_k)$. By identifying $\sigma_{k+1}$ with $s_r$ and $\alpha^{(k+1)}$ with $\alpha'$ we see that the conditions (i) and (ii) are satisfied. Now, observing that the sequence $\langle \text{prefix}(\alpha^{(k)}, \sigma_k) \rangle_{k \in \mathbb{N}}$ is monotone (any execution fragment in the sequence is a prefix of the succeeding one), we choose $\beta$ to be the limit of the sequence as $k \to \infty$. $\square$

We now define the notions of milestone states and rounds used in the analysis.

**Definition 3.2.** Let $\alpha$ be an admissible execution, $\sigma_0$ any state in $\alpha$, and $d > 0$ a constant. Let $\beta$ be a time-passage equivalent of $\alpha$, constructed as in Lemma 3.2, containing a subsequence of states $\{\sigma_k\}_{k \geq 0}$ such that $\sum_{t \in t \text{-prefix}(\beta, \sigma_0)} t = kd + \tau$ for every $k \in \mathbb{N}$. Then:

1. We call $\beta$ a milestone execution and denote it by $\alpha \upharpoonright_{d, \sigma_0}^\alpha$.
2. We call the states $\{\sigma_k\}_{k \geq 0}$ milestone states.
3. We call the execution fragments of $\alpha \upharpoonright_{d, \sigma_0}^\alpha$ between any two consecutive milestone states $\sigma_k$ and $\sigma_{k+1}$ a round.

Given that we can always construct a milestone execution for any admissible execution, in the rest of the paper, to avoid notational clutter, we assume implicitly that given any admissible execution $\alpha$ with a state $\sigma_0$, and $d > 0$, $\alpha$ contains the milestone states $\langle \sigma_k \rangle_{k \geq 0}$.

### 3.2. Measuring Performance

Fix $d > 0$, the normal message delay. JOIN-SYSTEM allows sending of messages at arbitrary times. For the purpose of analysis, we restrict the sending pattern: We assume that each automaton has a local real-valued clock, and sends messages at the first possible time and at regular intervals of $d$ thereafter, as measured on the local clock. Our results also require restrictions on timing and failure behavior: We define a segment of an admissible timed execution to be normal provided that all local clocks progress at rate exactly 1, all messages that are sent are delivered within time $d$ to
non-faulty nodes, local processing time is 0, and information is gossiped at intervals of $d$. We now define this setting in more detail.

**Definition 3.3 (α'-normal execution)** Let $α'$ be a finite prefix of an admissible timed execution $α$ of JOIN-SYSTEM. Execution $α$ is called $α'$-normal if it satisfies the following properties after $α'$:

1. Regular timing behaviour of automata: The local clocks of all the Join-Protocol$_v$ automata progress at the same rate of real time.
2. Periodic messaging: The gossip and join-acknowledgements messages are sent at regular intervals of $d$.
3. Reliable message delivery: Any message reaches its destination correctly if the destination is non-faulty.
4. Bound of message delay: Any message reaches its destination with a delay bounded by $d$.

Given a normal execution $α$ of JOIN-SYSTEM with periodic gossip, we are interested in the properties of join-connectivity graphs at milestone states of the time-passage equivalent milestone execution. We are interested in time necessary for any two nodes that join the system to learn about each other.

**Definition 3.4** Let $α$ be a $α'$-normal execution of JOIN-SYSTEM. Let $σ_0$ be a state in $α$ after $α'$, and let $(σ_i)_{i≥0}$ be the sequence of consecutive milestone states of $α_{σ_0}$ beginning with $σ_0$ and $\{(σ_i,J,σ_i,E)\}_{i≥0}$ be the corresponding sequence of join-connectivity graphs.

(i) For vertices $u, v \in σ_0.J$ that do not fail, we define join-delay $jd(σ_0, u, v)$ between $u$ and $v$ with respect to state $σ_0$ as $\min_{i≥0} \{ i : (u, v) \in σ_i.E \land u, v \in σ_i.J \}$.

(ii) We define the join-delay $jd(σ_0)$ for $σ_0$ as $jd(σ_0) = \max\{jd(σ_0, u, v) : u, v \in σ_0.J \land \forall i ≥ 0 : ¬σ_i.J.\text{failed} \land ¬σ_i.J.\text{failed}\}$.

The definition of join-delay gives the maximum number of communication rounds of JOIN-SYSTEM sufficient for any two non-faulty nodes that joined the system to learn about each other. So, for a milestone state $σ_0$, the quantity $d \cdot jd(σ_0)$ corresponds to the maximum time sufficient for any two non-faulty nodes that joined by $σ_0$ to discover one another.

### 4. Analysis of JOIN-SYSTEM

In JOIN-SYSTEM the input action $\text{join}(H)_v$ the non-empty subset $H$ may be of any finite size. In this paper we distinguish two cases: $|H| = 1$, which we call single-request join, and $|H| > 1$, which we call multi-request join. In the rest of this section we analyze the behavior of JOIN-SYSTEM with the following assumptions:

1. single-request without any faulty processes,
2. multi-request without any faulty processes,
3. single-request with faulty processes, and
4. multi-request with faulty processes.

#### 4.1. Join-Delay in the Absence of Failures

For failure-free executions, single-request join behavior analysis is a special case of the multi-request case. We first evaluate the multi-request behavior, then specialize it for the single-request case. In both cases we show that if two nodes join the system by the time it becomes stable, then they learn about each other in time proportional to the logarithm of the diameter of the join-connectivity graph in which both nodes are originally represented.

#### 4.1.1. Multi-requests in the absence of failures

We start by giving two lemmas, then present the result.

**Lemma 4.1** Let $α$ be a $α'$-normal, $d$-delay (milestone) execution of JOIN-SYSTEM, where after $α'$ no nodes fail and new nodes may join at one or more nodes. Let $\{σ_j\}_{j≥0}$ be the milestone states after $α'$.

1. single-request without any faulty processes,
2. multi-request without any faulty processes,
3. single-request with faulty processes, and
4. multi-request with faulty processes.

For failure-free executions, single-request join behavior analysis is a special case of the multi-request case. We first evaluate the multi-request behavior, then specialize it for the single-request case. In both cases we show that if two nodes join the system by the time it becomes stable, then they learn about each other in time proportional to the logarithm of the diameter of the join-connectivity graph in which both nodes are originally represented.
Since $k' \leq k$ hence $u \in \sigma_{i+4}.B(v; \lceil \frac{k}{2} \rceil)$ which proves $\sigma_i.B(v; k) \subseteq \sigma_{i+4}.B(v; \lceil \frac{k}{2} \rceil)$.

**Lemma 4.2** Let $\alpha$ be a $\alpha'$-normal execution of JOIN-SYSTEM, where after $\alpha'$ no nodes fail and new nodes may join at one or more nodes. Let $\langle \sigma_j \rangle_{j \geq 0}$ be the milestone states in $\alpha$ after $\alpha'$. Then $jd(\sigma_0) = O(\log(\sigma_0.G.diam))$.

**Proof.** Recall that $\sigma_0.G.diam$ is the diameter of the join-connectivity graph $\sigma_0.G = (\sigma_0.J, \sigma_0.E)$. Let us consider any $u, v \in \sigma_0.J$. Since there are no faulty nodes we know that $u \in \sigma_0.B(v; \sigma_0.diam)$. Now, using Lemma 4.1 we get that $v \in \sigma_{i}.B(v; 1)$ for $i = O(\log(\sigma_0.G.diam))$ and hence $jd(\sigma_0) = O(\log(\sigma_0.G.diam))$. □

**Lemma 4.3** Let $\alpha$ be a $\alpha'$-normal execution of JOIN-SYSTEM, where after $\alpha'$ no nodes fail and new nodes may join at one or more nodes. Let $\langle \sigma_j \rangle_{j \geq 0}$ be the milestone states in $\alpha$ after $\alpha'$. Then $jd(\sigma_0) = \Omega(\log(\sigma_0.G.diam))$.

**Proof.** We assume that no new processes join after the system reaches system state $\sigma_0$ and consider some state $\sigma_i$, for $i \geq 0$. Suppose the diameter of the graph $(\sigma_j, J, \sigma_i.E)$ is $n_i$, i.e., $\sigma_i.diam = n_i \geq 1$. So, there exists $u, v \in \sigma_i.J$ such that $\sigma_i.dist(u, v) = n_i$ and hence there is a path $u = u_0, u_1, \ldots, u_{n_i-1}, u_{n_i} = v$ in $(\sigma_j, J, \sigma_i.E)$. Now, observe that for any $u_j, u_k \in \sigma_j.J$ for $0 \leq j, k \leq n_i$ we have $\sigma_i.dist(u_j, u_k) = |j - k|$, otherwise, $\sigma_i.dist(u_j, v) < n_i$. Let us denote by the set $U = \{u_0, u_1, \ldots, u_{n_i-1}, u_{n_i}\}$. Now, consider the undirected subgraph $(U, E_U(\sigma_i))$ of the graph $(\sigma_j, J, \sigma_i.E)$ by including those edges of $\sigma_i.E$ which are among the vertices of $U$. Now, we know that $E_U(\sigma_i) = \{(u_j, u_{j+1}) : 0 \leq j < n_i \land u_j, u_{j+1} \in U\}$.

Next, consider the join-connectivity graph $(\sigma_{i+1}.J, \sigma_{i+1}.E)$ when the system is in state $\sigma_{i+1}$. Since there are no node failures we have $U \subseteq \sigma_{i+1}.J$. Now, let us consider the subgraph $(U, E_U(\sigma_{i+1}))$ of the graph $(\sigma_{i+1}.J, \sigma_{i+1}.E)$. Now according to the assumption of a $\alpha'$-normal execution we have $\sigma_{i+1}.E_U \subseteq \sigma_i.E_U \cup \{(u_j, u_{j+2}) : 0 \leq j < n_i - 1 \land u_j, u_{j+2} \in U\}$. The existence of any edge other than the ones in $\sigma_{i+1}.E_U$ would signify that $\sigma_{i+1}.dist(u, v) < n_i$. Now, by inspecting the set of edges $\sigma_{i+1}.E_U$ we observe that $\sigma_{i+1}.dist(u, v) \geq \lceil \frac{\sigma_i.dist(u, v)}{2} \rceil$. Hence $\sigma_{i+1}.diam \geq \sigma_{i+1}.dist(u, v) \geq \lceil \frac{\sigma_i.dist(u, v)}{2} \rceil = \lceil \frac{\sigma_i.G.diam}{2} \rceil$. Clearly, we have for $i = \Omega(\log(\sigma_0.G.diam))$ we have $\sigma_{i+1}.diam \geq 2$. Therefore, we have $jd(\sigma_0) = \Omega(\log(\sigma_0.G.diam))$. □

**Theorem 4.4** Let $\alpha$ be a $\alpha'$-normal execution of JOIN-SYSTEM, where after $\alpha'$ no nodes fail and new nodes may join at one or more nodes. Let $\langle \sigma_j \rangle_{j \geq 0}$ be the milestone states in $\alpha$ after $\alpha'$. Then $jd(\sigma_0) = \Theta(\log(\sigma_0.G.diam))$.

**Proof.** Follows from Lemma 4.2 and Lemma 4.3. The lower bound is true since Lemma 4.3 holds even though no nodes join the system. □

4.1.2. Single-requests in the absence of failures

We now specialize the multi-request join result for the single-request case.

**Theorem 4.5** Let $\alpha$ be a $\alpha'$-normal execution of JOIN-SYSTEM, where after $\alpha'$ no nodes fail and new nodes may join at only one node. Let $\langle \sigma_j \rangle_{j \geq 0}$ be the milestone states in $\alpha$ after $\alpha'$. Then $jd(\sigma_0) = \Theta(\log(\sigma_0.G.diam))$.

**Proof.** This result is a special case of Theorem 4.4. □

4.2. Joining Delay in the Presence of Failures

In this section we consider executions with failures. We first evaluate the single-request case, then multi-request case. Recall that we assume that the network does not partition and that the join-connectivity graphs are always connected. For the single-request case we show that if two nodes join the system by the time it becomes stable, then they learn about each other in time proportional to the logarithm of the diameter of the join-connectivity graph in which both nodes are originally represented. However for the multi-request case, we show that it is possible to have unbounded join-delay.

4.2.1. Single-requests in the presence of failures

**Theorem 4.6** Let $\alpha$ be a $\alpha'$-normal execution of JOIN-SYSTEM, where after $\alpha'$ nodes may fail and new nodes may join at only one node. Let $\langle \sigma_j \rangle_{j \geq 0}$ be the milestone states in $\alpha$ after $\alpha'$. Then $jd(\sigma_0) = \Theta(\log(\sigma_0.J))$.

**Proof.** Let us define for any state $\sigma_i$ and vertex $w \in \sigma_i.J$, $\sigma_i.L(w) = \max\{k : \sigma_i.N(w; k) \neq \emptyset\}$. Now, consider the join-connectivity graph $(\sigma_j, J, \sigma_i.E)$ and any two vertices $u, v \in \sigma_i.J$ such that for some $i' \geq i, (u, v) \in \sigma_i.E$. Then there is a path from $v$ to $u$ in $(\sigma_j, J, \sigma_i.E)$. Also, we know $\sigma_i.L(v) \leq |\sigma_i.J|$. Now, if no node fails between the states $\sigma_i$ and $\sigma_{i+4}$, i.e., $\sigma_i.J \subseteq \sigma_{i+4}.J$, then we have $\sigma_{i+4}.L(v) \leq \lceil \frac{\sigma_i.L(v)}{2} \rceil + 1$. Next, we observe the above recurrence relation holds even for the case where some noes my fail between states $\sigma_i$ and $\sigma_{i+4}$. This is true because nodes can join through one other node hence no new node can be in any shortest path during the execution until the fourth milestone state after it received the join-acknowledgment. To show this observe that for any $w' \in \sigma_{i+4}.J(v)$ there is a shortest path between $u$ and $v'$ in $(\sigma_{i+4}.J, \sigma_{i+4}.E)$ that includes no node that joined between states $\sigma_i$ and $\sigma_{i+4}$. To see this suppose $w \in \sigma_{i+4}.J$ and $w \neq v, v'$ be such a node that received the join-acknowledgment between states $\sigma_i$ and $\sigma_{i+4}$ from some
there is a vertex \( v' \). But when \( v'' \) sent the join-acknowledgment to \( w \) at that instant \( N(v; 1) \) contained at least two other vertices that are in a shortest path from \( v \) to \( v'' \) in \( (\sigma_{i+4}.J, \sigma_{i+4}.E) \).

Let these vertices be \( v_1 \) and \( v_2 \) then because the states \( \sigma_i \) and \( \sigma_{i+4} \) corresponds to time instants that differ by at least 2\( d \) time units we have \((v_1, v_2) \in \sigma_{i+4}.E \) which obviates the inclusion of \( w \) in the shortest path between \( v \) and \( v' \).

This shows that \( \sigma_i. L(v) \leq \left[ \frac{\sigma_i. L(v)}{2} \right] + 1 \). So, for state \( \sigma_i \), such that, \( i = O(\log |\sigma_i.J|) \) we have \( \sigma_i. L(v) \leq 2 \).

Therefore, \( \sigma_i.J \subseteq \bigcup_{k=1}^{2} \sigma_i.N(v; k) \). Hence after one more step we have either \( v \in u.world \) or \( v \in v.world \). So, \( jd(\sigma_0) = O(\log (|\sigma_0.J|)) \).

\[ \square \]

**Theorem 4.7** Let \( \alpha \) be a \( \alpha' \)-normal execution of JOIN-SYSTEM, where after \( \alpha' \) nodes may fail and new nodes may join at only one node. Let \( \langle \sigma_j \rangle_{j \geq 0} \) be the milestone states in \( \alpha \) after \( \alpha' \). Then \( jd(\sigma_0) = \Omega(\log(\alpha_0.G.diam)) \).

**Proof.** This follows directly from Lemma 4.3. \[ \square \]

### 4.2.2. Multi-requests in the presence of failures

We now show that in the presence of failures, when new nodes join the system at multiple nodes, then it is possible to have unbounded join-delay (Theorem 4.8). We also show that by constraining failures somewhat, we are able to get a reasonable join-delay, e.g., logarithmic in the diameter of a join-connection graph at some instant (Theorem 4.9).

**Theorem 4.8** Let \( \alpha \) be a \( \alpha' \)-normal execution of JOIN-SYSTEM, where after \( \alpha' \) nodes may fail and new nodes may join at one or more nodes. Let \( \langle \sigma_j \rangle_{j \geq 0} \) be the milestone states in \( \alpha \) after \( \alpha' \). Then \( jd(\sigma_0) \) is unbounded.

**Proof.** Consider \( n \in \mathbb{N} \) such that \( n > 2 \) and the graph \( \langle \sigma_i.J, \sigma_i.E \rangle \), where \( \sigma_i.J = \{v_0, v_1, \ldots, v_{n-1}\} \) and \( \sigma_i.E = \{(v_i, v_j) : v_i \in \sigma_i.J \setminus \{v_0\}\} \) as shown in Figure 2. Clearly, the graph \( \langle \sigma_i.J, \sigma_i.E \rangle \) is connected and \( |\sigma_i.J| = n \). We call vertex \( v_0 \) the center vertex for the star graph \( \langle \sigma_i.J, \sigma_i.E \rangle \). Also, we assume that there is a vertex \( v' \) such that \( \sigma_i.v'.status = request \) and send(join) \( v', v_i \), \( v_i \in \sigma_i.J \setminus \{v_0\} \) and receive(m) \( v', v_i \), \( v_i \in \sigma_i.J \setminus \{v_0\} \) are executed before the system state was \( \sigma_i \).

Now, we consider the following adversary or join-failure pattern \( A \):

1. At the end of every round of communication, i.e., at \( \sigma_j, j \geq i \), the adversary \( A \) adds a node \( v_0' \) and so that \( v_0' \) executes send(join) \( v', v_i \), \( v_i \in \sigma_i.J \setminus \{v_0\} \) i.e. send join-request to the nodes \( \{v_1, v_2, \ldots v_{n-1}\} \) beginning at \( \sigma_j \).
2. Just before the beginning of every round of communication, i.e., just before the global clock tick the adversary \( A \) crashes the central vertex \( v_0 \).

![Figure 2. Illustration for the proof of Theorem 4.8](image)

It is easy to observe that a graph isomorphic to the graph \( \langle \sigma_i.J, \sigma_i.E \rangle \) is preserved at every \( i' > i \) with only the central vertex changing. \[ \square \]

As we have seen in the previous theorem, join-delay is unbounded in the case when with node failures and new nodes may join through multiple nodes. In order to guarantee bounded join-delay we consider a weaker adversarial behaviour. This adversary can fail almost an exponential number of nodes, with respect to the number of rounds.

In more precise terms, consider the join-connection graph \( \langle \sigma_0.J, \sigma_0.E \rangle \) and let us denote by \( F_{0,k} = \sigma_0.J \setminus \sigma_k.J \). So, \( F_{0,k} \) denotes the set of nodes that were alive at milestone state \( \sigma_0 \) but failed before milestone state \( \sigma_k \). We assume an adversary such that there exists some set of mutually disjoint open balls \( \{\sigma_0.B(v; 2^\frac{1}{k})\} \in \mathbb{N} \) such that \( F_{0,k} = \bigcup_{v \in T} \sigma_0.B(v; 2^\frac{1}{k}) \) for every \( k > 1 \).

**Theorem 4.9** Let \( \alpha \) be a \( \alpha' \)-normal execution of JOIN-SYSTEM, where after \( \alpha' \) nodes may fail and new nodes may join at one or more nodes. Let \( \langle \sigma_j \rangle_{j \geq 0} \) be the milestone states in \( \alpha \) after \( \alpha' \). Consider the adversarial behavior where there exists some set of mutually disjoint open balls \( \{\sigma_0.B(v; 2^\frac{1}{k})\} \in \mathbb{N} \) such that \( F_{0,k} = \bigcup_{v \in T} \sigma_0.B(v; 2^\frac{1}{k}) \) for every \( k > 1 \). Then \( jd(\sigma_0) = O(\log(\sigma_0.G.diam)) \).

**Proof.** Consider any \( v, u \in \sigma_0.J \) so that eventually the edge \((u, v)\) appears in some join-connection graph in a state occurring after \( \sigma_0 \). Since there is a path from \( v \) to \( u \) in the undirected version of \( \langle \sigma_0.J, \sigma_0.E \rangle \), hence there ex-
ists a sequence of, say \( n + 1 \), balls \( \sigma_0.B(u; 2) = \sigma_0.B(u_0; 2), \sigma_0.B(u_1; 2), \ldots, \sigma_0.B(u_{n}; 2) = \sigma_0.B(v; 2) \) such that \( \sigma_0.B(u_k; 2) \cap \sigma_0.B(u_{k+1}; 2) \neq \emptyset \) for \( k = 0, 1, \ldots, n - 1 \).

Observe that in state \( \sigma_4 \) for any three balls \( \sigma_0.B(u_k; 2), \sigma_0.B(u_{k+1}; 2) \) and \( \sigma_0.B(u_{k+2}; 2) \) for \( k = 0, 1, \ldots, n - 2 \) we have for some \( w \in \sigma_0.B(u_k; 2) \) and \( w' \in \sigma_0.B(u_{k+2}; 2) \) that \( (w, w') \in \sigma_4.E \). This is because of the restriction on the possible failures caused by the adversary.

Now in state \( \sigma_4 \) we have a sequence of, say, \( \left[ \frac{4}{1} \right] = n' \) balls \( \sigma_4.B(u; 2) = \sigma_4.B(u_0; 2), \sigma_4.B(u_1; 2), \ldots, \sigma_4.B(u_{n'}; 2) = \sigma_4.B(v; 2) \) such that \( \sigma_4.B(u_k; 2) \cap \sigma_4.B(u_{k+1}; 2) \neq \emptyset \) for \( k = 0, 1, \ldots, n' - 1 \). So, finally in some state \( \sigma_i \) for \( i = O \left( \log \left( \sigma_0.G.diam \right) \right) \) we have \( (u, v) \in \sigma_i.E \), and since \( n \leq \sigma_0.G.diam \) we have \( jd(\sigma_0) = O \left( \log \left( \sigma_0.G.diam \right) \right) \). \( \square \)

5. Conclusion

In this paper we introduced and studied the Join Problem for dynamic network algorithms. We specified an asynchronous protocol, called Join-Protocol, allowing nodes to join the system and to learn about each other by means of gossip. We studied join-delay, the time required for two nodes to learn about each other once they joined the system. We specified Join-Protocol using the Input/Output Automata notation and we analyzed the protocol by imposing some constraints on the executions of Join-Protocol.

In order to examine execution states at specific time instants we defined the concept of milestone states in the time-traversal refinement of arbitrary executions. We defined the performance measures for studying the problem in terms of milestone states and join-connectivity graphs. We performed the analysis of four distinct scenarios involving different assumptions about the failures and joining at single or multiple locations. We derived upper and lower bounds for join-delay in each of these scenarios. For example, in the case when new participants join at multiple participants and participants may crash, the number of rounds cannot be bounded. In the more benign cases when the failures can be controlled or when new participants join at only one participant, the bound on rounds is shown to be logarithmic in the diameter of the initial configuration. We made a surprising observation that while joining at multiple nodes leads to improved fault-tolerance, it also may lead to unbounded join-delays. Thus we considered a weaker adversary for which a logarithmic join-delay can still be obtained.

Future work includes assessing join-delay for protocols that incorporate constrained gossip. In this work we deal with the failures consisting of node crashes and message loss and reordering. It is also interesting to consider other types of failure models.

References


