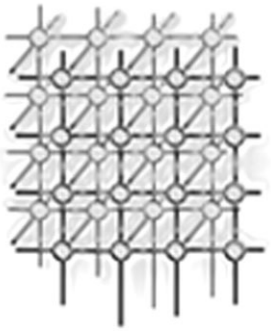
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2005; **17**:1471–1495
Published online 22 April 2005 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.879

On the modelling of publish/subscribe communication systems



R. Baldoni*,†, R. Beraldi, S. Tucci Piergiovanni and A. Virgillito

Dipartimento di Informatica e Sistemistica, Università di Roma 'La Sapienza', Via Salaria 113, 00198 Roma, Italy

SUMMARY

This paper presents a formal framework of a distributed computation based on a publish/subscribe system. The framework abstracts the system through two delays, namely the subscription/unsubscription delay and the diffusion delay. This abstraction allows one to model concurrent execution of publication and subscription operations without waiting for the stability of the system state and to define a Liveness property which gives the conditions for the presence of a notification event in the global history of the system. This formal framework allows us to analytically define a measure of the effectiveness of a publish/subscribe system, which reflects the percentage of notifications guaranteed by the system to subscribers. A simulation study confirms the validity of the analytical measurements. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: publish/subscribe; event-based middleware

1. INTRODUCTION

Communication systems following the publish/subscribe (pub/sub) paradigm have experienced a relevant gain in popularity during the last few years. Each participant in a pub/sub system can take on the role of a *publisher* or a *subscriber* of information. Publishers produce information (in the form of *notifications*), which is consumed by subscribers. The basic characterization of pub/sub derives from the way notifications flow from senders to receivers: receivers are not directly targeted by the publisher, but rather they are indirectly addressed through the content of notifications. That is, subscribers express their interest by issuing *subscriptions* for specific notifications, independently from the publishers that produce them, and then they are *asynchronously* notified for all notifications,

*Correspondence to: R. Baldoni, Dipartimento di Informatica e Sistemistica, Università di Roma 'La Sapienza', Via Salaria 113, 00198 Roma, Italy.
†E-mail: baldoni@dis.uniroma1.it

Contract/grant sponsor: Project EU-PUBLI.COM; contract/grant number: #IST-2001-35217

Received 31 March 2003
Accepted 20 September 2003

submitted by any publisher, that match their subscriptions. Both subscriptions and notifications are managed by a logically centralized entity, namely the *Notification Service*, which creates a complete decoupling among the participants.

Since pub/sub has been largely recognized as an effective approach for information diffusion, lots of pub/sub-based systems, both research contributions [1–6] and commercial products [7], have been presented and are actually used in several application contexts. From the research side, a lot of work has been done in this field by the software engineering community, focusing on scalability, efficient information delivery or efficient and expressive information matching. However, only a few contributions exist [8] that define, for example, the computational model underlying a pub/sub system and, most importantly, the safety and Liveness properties that a pub/sub system must ensure in the underlying computation with respect to publish and notification events. This step is necessary for carrying out, for example, an analytical study of the performance of a pub/sub system which is the base of a rigorous QoS policy. The lack of this rigorous approach is currently one of the main pitfalls of the pub/sub paradigm, which limits its applicability, for example, to mission critical systems.

In this paper we propose a computational model based on a Notification Service that is abstracted as a box connecting all participants to the computation and the operations done by this box (i.e. subscription/unsubscription storage and publication diffusion) are modelled by two delays: the subscription delay and the diffusion delay, which characterize, respectively, (i) the non-atomicity of the subscription/unsubscription storage and (ii) the non-instantaneous diffusion of a notification. These delays depend, of course, on the implementation of the Notification Service itself (e.g. centralized, network of brokers, etc.).

This model produces a global history of the computation for which we give two simple, safety basic properties, namely legality (i.e. a history contains only notify events included in a matching subscription interval) and validity (i.e. a notify event implies the presence in the system of a prior corresponding event of publishing). These properties are independent of the delays. Then we propose a Liveness property which states when a notify event belongs to the history: this is affected by the interval when a subscription is 'active' and by the two delays which act as a filter for the generation of the notify events after the execution of a publishment[‡]. In other words, our Liveness property gives a timing condition implying, given a publish event, the presence of a corresponding notify event in the global history. Note that the opposite is not true, i.e. there could be a notify event in the history even though the condition is not verified.

This Liveness condition gives us the opportunity to define a measure of the effectiveness of a Notification Service in which publications are notified to the set of interested subscribers. In particular, we provide a probabilistic model for measuring the effectiveness of a Notification Service, in which we evaluate the probability d that a publication x issued at time t will be notified to each subscriber matching x, provided that the subscription was active t. Therefore, the system behaves ideally if this probability is equal to 1. We study this probability as a function of the subscription delay and of the diffusion delay. A simulation study, carried out on a real pub/sub implementation, validates the analytical model by confirming the results of the evaluations.

[‡]We would like to remark that usually in distributed computing liveness means 'something good eventually happens' in this dynamic context where (i) participants can subscribe and unsubscribe dynamically and (ii) there are operations which take time to take effect, this sentence should be reworded as follows 'under some timing conditions something good happens'.

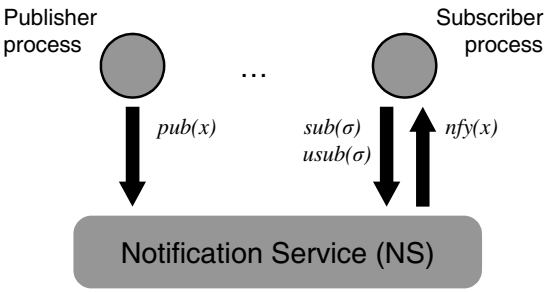


Figure 1. A pub/sub system interaction.

Even though the granularity of this model is quite coarse, we believe that it could be very useful for the designer of a Notification Service, in that it can predict the probability of delivering notifications, only by estimating the two delays. Furthermore, more refined analysis (such as simulation studies) must take into account so many parameters (from network to application), which makes it very hard to have an overall and precise performance assessment in a practical setting because of the dynamic nature of the pub/sub paradigm and of the underlying network.

The paper is structured as follows: Section 2 introduces the formal framework, Section 3 presents the analytical probabilistic model for performance evaluation, Section 4 presents the experimental results and a comparison with the analytical ones, and finally, Section 5 surveys the related work.

2. A FRAMEWORK FOR PUBLISH/SUBSCRIBE

We consider a distributed system composed of a set of processes $\Pi = \{p_1, \ldots, p_n\}$ that communicate by exchanging information in a pub/sub communication system. Processes are decoupled in the sense that they never communicate directly with each other but only through a common *Notification Service* (NS). Processes can act both as producers and consumers of information, taking on the role of *publishers* and *subscribers*, respectively (Figure 1).

On the subscribers' side, interest in specific information is expressed through *subscriptions*. A subscription is a pair $\sigma = (\phi, p)$, where $p \in \Pi$ is the subscriber which is interested in all publications declared through the *filter* ϕ. A filter ϕ is a query expression composed of a set of constraints. The constraints, depending on the attribute type, can comprise equality, comparison, substring, etc. and can be joined inside filters through AND/OR expressions.

We say a notification x *matches* the filter ϕ if each attribute in x satisfies all the constraints in ϕ. The task of verifying whenever information x matches a filter ϕ is called *matching* ($x \sqsubset \phi$). We say that x matches a subscription σ if it matches $\sigma.\phi$ ($x \sqsubset \sigma.\phi$).

2.1. Process–NS interaction

The execution of a pub/sub system comprises both process-side operations, started by subscribers and publishers, and NS-side operations, started by the NS. More specifically, any process p_i is able to register (and cancel) a subscription or to publish a notification in the system, but it is actually the NS that has the role of notifying a matching occurrence to interested subscribers.

We denote as $op = \{sub(\sigma), usub(\sigma), pub(x), ntfy(x)\}$ the operations of registration of a subscription σ, cancellation of a subscription σ, publication of a notification x and issue of the notification of x, respectively. Then, the operations $sub(\sigma), usub(\sigma), pub(x)$ are issued by a process and executed by the NS, while $ntfy(x)$ is issued by the NS on a process p_i and then executed by p_i. The $ntfy(x)$ issue occurs after (i) the $pub(x)$ execution and (ii) a matching operation executed within the NS. Note that the NS issues $ntfy(x)$ on the set of processes computed after the matching operation.

2.2. Computational model

To simplify the presentation, we assume the existence of a discrete global clock whose range $\mathcal{T}$ is the set of natural numbers. We stress the fact that this is only a fictional, abstract device to which the processes *do not have access*. We will use it only for convenience of specification.

The first modelling step is the representation of the execution of each process. Through an abstract representation of the processes' computation, we describe which global computations are allowed in a NS by specifying properties that characterize them.

We assume either the *issue* of an operation $op = \{pub(x), sub(\sigma), usub(\sigma)\}$ at time t at a process p_i or the *execution* of $op = ntfy(x)$ at p_i at time t produces an *event* $e_i(op, t)$ at process p_i[§]. We denote then the *local history* of a process p_i as the set of events occurring at p_i and ordered by their occurrence time $h_i = \{e_i(op, t_1), e_i(op, t_2), \ldots e_i(op, t_m)\}$ (with $t_1 < t_2 < \cdots < t_m$). The global computation is then the *global history* $H = \langle h_1, h_2, \ldots, h_n \rangle$, i.e. a collection of local histories, one for each process.

Any two successive events, $e_i(sub(\sigma), s)$ and $e_i(usub(\sigma), u)$ $(s < u)$, define a *subscription interval* of p_i for the subscription σ, denoted by $I(\sigma)$. Such a subscription interval includes all events $e_i(op, t)$ s.t. $s \leq t \leq u$. Therefore, to univocally identify each subscription issued in the system by the same process, a generic subscription σ becomes a triple (ϕ, p, s) where $\sigma.s$ indicates the time in which the subscription is issued. The time between s and u actually represents the time in which the subscription σ is active from the subscriber view-point. We denote such a time interval as $T_{\mathrm{ON}}(\sigma)$. A subscription interval is also defined by those *sub* events that have no corresponding *usub*. In this case the interval will include all events that occur after the *sub* and T_{ON} will consequently be infinite. Figure 2 shows an example of the global history of three processes, with two subscription intervals $I(\sigma)$, $I(\sigma')$ and their corresponding T_{ON}.

[§]In this section we use the term 'event' only in referring to events belonging to the internal computation of processes. Pieces of information produced by publishers are *always* referred to as 'notifications'.

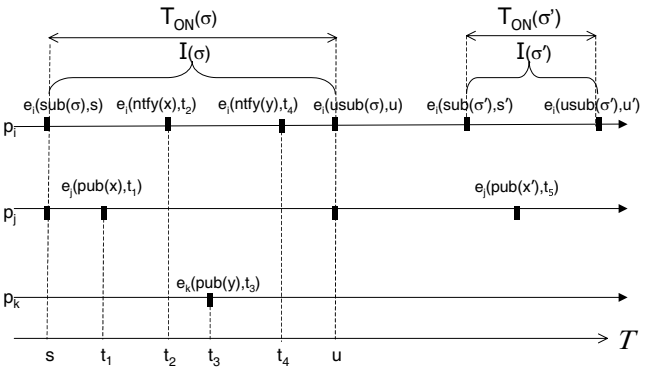


Figure 2. Global history respecting safety.

2.2.1. *Safety properties*

Safety properties pose constraints on which global histories are not allowable in a NS. The first property has to state the basic semantics of the system: a subscriber cannot be notified of information it is not interested in. Formally,

$$\forall e_i(ntfy(x), t) \in H \Rightarrow e_i(ntfy(x), t) \in I(\sigma) \text{ s.t. } x \sqsubset \sigma.\phi \wedge \sigma.p = i$$

P1: Legality

In Figure 2 a generic computation satisfying Legality is shown: supposing that x and y match σ, then both notify events of x and y in p_i fall in the subscription interval $I(\sigma)$ of p_i. While Legality states that a notify event belongs to H only if it is included in a subscription interval matching that event, we need a property ensuring that the notify events are not invented by a process. This is taken into account by the Validity property which states

$$\forall e_i(ntfy(x), t) \in H \Rightarrow \exists e_j(pub(x), t') \in H \text{ s.t. } t' < t$$

P2: Validity

The computation in Figure 2 also respects Validity: both notify events, $e_i(ntfy(x), t_2)$ and $e_i(ntfy(y), t_4)$ follow the corresponding publications, $e_i(pub(x), t_1)$ and $e_i(pub(x), t_3)$, as $t_1 \leq t_2$ and $t_3 \leq t_4$.

Once safety properties have been defined, it is interesting to ascertain under which condition a notify event should be generated, i.e. to define a Liveness property. As already mentioned, the global history in Figure 2 satisfies safety. However supposing x' matches σ', should we expect that the NS system generates a computation with the notify event for x' in $I(\sigma')$? To answer this question it is first essential to make some considerations about how a NS is physically built. This is actually the aim of the following section.

2.3. NS implementation parameters

Basically, we can say that the NS has two main tasks:

- store and manage subscriptions from processes caused by the issue of subscribe/unsubscribe operations;
- diffuse a notification x to the interested subscribers after a publish operation was issued by a process.

Obviously, behind this abstract and informal description of a NS there exists an actual NS physical implementation (e.g. centralized, distributed, network of brokers, etc.) that performs the desired functionality. In order to capture the behaviour of *any* NS implementation we define two parameters that respectively take into account (i) non-instantaneous effects of subscribe/unsubscribe operations and (ii) the non-instantaneous diffusion of a notification x to interested subscribers after a publish operation issued by a process. These parameters model the time required for the internal processing at the NS and the network delay elapsed to route subscriptions and notifications in a distributed implementation. Let us finally assume that any message sent by a process of a NS implementation uses reliable channels.

2.3.1. Subscription/unsubscription delays

When a process issues a subscribe/unsubscribe operation, the NS is not immediately aware of the occurred event. In other words, at an abstract level, the registration (respectively cancellation) of a subscription takes a certain amount of time to be stored in the NS. This time encompasses, for example, the update of the internal data structures of the NS and the network delay due to the routing of the subscription among all the entities constituting the NS. To consider such non-instantaneous operations, we define a maximum acceptable threshold of time (implementation dependent) after which a subscribe/unsubscribe operation is *surely* stored in the NS. As an example, in a distributed implementation of a NS, this means *each entity* implementing the NS after this threshold of time is aware of the registration/cancellation operation.

We denote such a delay as T_{sub} for subscribe operations and T_{usub} for unsubscribe operations. Therefore, if a subscribe operation is issued at time s, then it takes effect at a time t, such that $s < t \leq s + T_{\text{sub}}$[¶]. The same holds for unsubscribe operations, i.e. an unsubscribe operation, issued at time u, takes effect at a time t' such that $u < t' \leq u + T_{\text{usub}}$.

To model this effect on the NS, we consider the NS characterized by a *subscription configuration* sc composed of a set of subscriptions. In particular, we define $sc(t) = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ to be the set of all subscriptions stored in the NS at time t. We assume the initial configuration $sc(t_0) = \emptyset$. Therefore, if a subscribe (respectively unsubscribe) operation for a subscription σ takes effect at time t (respectively t') then $\sigma \in sc(t)$ (respectively $\sigma \notin sc(t')$). As a consequence, even though t and t' are *a priori* unknown, we can state with certainty that $\sigma \in sc(s + T_{\text{sub}})$ and $\sigma \notin sc(u + T_{\text{usub}})$. For example, in Figure 3, $\sigma \in sc(t_1)$ and $\sigma \notin sc(t_2)$, but in both $[s, t_1]$, $[u, t_2]$ time intervals there is uncertainty whenever $\sigma \in sc$ or not.

[¶]In our framework we reasonably assume for each subscription σ we have $T_{\text{ON}}(\sigma) > T_{\text{sub}}$.

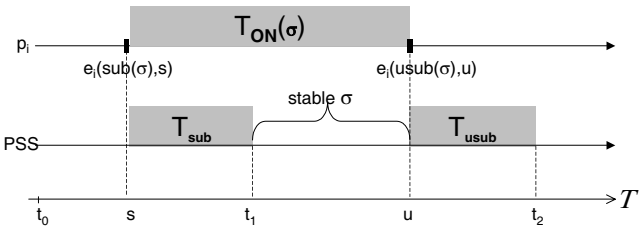


Figure 3. Subscription/unsubcription delays.

At an abstract level each subscription of the NS state at time $t \in \mathcal{T}$ can therefore be stable (i.e. it surely belongs to NS state) or non-stable. A subscription σ is stable with certainty at time t, iff $s + T_{\text{sub}} \leq t \leq s + T_{\text{ON}(\sigma)}$.

2.3.2. *Diffusion delay*

As soon as a publication is issued, the NS performs a *diffusion* of the information: it performs a matching to compute the set of interested subscribers and sends the notification to them. Note that, depending on the NS implementation, the diffusion can be performed in several ways (for example, through an application-level network of brokers [1–3] or by leveraging an overlay network infrastructure [9–12]). Without entering implementation details, we can say that this operation takes a certain amount of time *during* which the NS computes and issues notify operations to interested subscribers, i.e. diffusion takes a non-zero period of time. Let us suppose that a publication of a notification x is made at a given time t, and there is a matched subscription σ that is stable at time t, i.e. $\sigma \in sc(t)$. Then the NS starts the diffusion to notify x to $\sigma.p = p_i$. We denote as Δ_i the time elapsed in order to complete the diffusion of x to p_i. An event $e_i(ntfy(x), t')$ can be generated only at time $t' \leq t + \Delta_i$. After the completion of the diffusion, the notification x disappears from the NS, i.e. a further notify event can no longer be generated.

Note that in the worst case scenario, the set of subscribers to be notified and the whole set of processes coincides. In this case the diffusion takes the maximum time among $\{\Delta_1, \Delta_2, \ldots, \Delta_n\}$. We define such maximum delay as *diffusion delay*, denoted as T_{diff}.

To clarify the meaning of the diffusion delay, see Figure 4. For the sake of simplicity and without loss of generality, we assume that the communication delay between a process and the NS is zero. This implies that (i) a notification published by a process immediately gets the NS, and (ii) if the NS issues a notify operation on a process p_i, the corresponding local event at p_i is immediately generated. Immediately after the publication of the notification x at the time t_1, the NS, during T_{diff}, notifies the interested subscribers. Supposing that $\{p_j, p_k, p_h\}$ is the set of interested subscribers, then $T_{\text{diff}} = \max\{\Delta_j, \Delta_k, \Delta_h\} = \Delta_h$. Let us remark that each generic interested subscriber p_i is notified in a specific instant of time $(t + \Delta_i)$, but Δ_i is not *a priori* known.

It is important to point out that the set of interested subscribers is clearly computed on the basis of the configuration of NS. However, *how and when* the state is considered is implementation dependent.

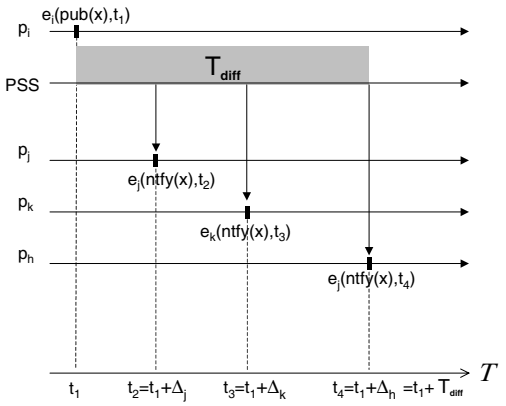


Figure 4. Example of diffusion.

Moreover, *the configuration can change during the diffusion*. In the following these aspects will be clarified.

2.4. Liveness property

The concept of 'interested subscriber' has been considered quite intuitively until this point. The desirable NS behaviour is the following: once a notification is published (i.e. $e_j(pub(x), t)$ is generated in H), x is notified to each interested subscriber; but what is an interested subscriber? Ideally it is a process p_i that expresses its interest for x through a subscription σ s.t. $x \sqsubset \sigma.\phi$ and $\sigma.s \leq t \leq \sigma.s + T_{\mathrm{ON}}(\sigma)$. However, the NS system is surely aware of the subscription σ by p_i only when the subscription becomes stable, i.e. at time $\sigma.s + T_{\mathrm{sub}}$. Then, at first check, an interested subscriber seems to be a process whereby subscription is stable (i.e. belonging to the NS state), at the moment in which the matching information is published, i.e. $\sigma.s + T_{\mathrm{sub}} \leq t \leq \sigma.s + T_{\mathrm{ON}}(\sigma)$.

However, as (i) the interest of a subscriber is a dynamic dimension and (ii) a notify can be issued to a subscriber at any time during the diffusion interval of the corresponding publication, it is still difficult to characterize the exact behaviour of the system. Let us demonstrate this with an example. Let p_i be a process producing a subscription σ and p_j be a process producing an event $e_j(pub(x), t)$, such that $x \sqsubset \sigma.\phi$ and $\sigma.s + T_{\mathrm{sub}} \leq t \leq \sigma.s + T_{\mathrm{ON}}(\sigma)$. However, if NS *is able to notify* x at p_i *only* at a time $t' = t + \Delta_i$ such that $t' > \sigma.s + T_{\mathrm{ON}}(\sigma)$, then p_i will discard x as it is no longer interested in x.

Then, the definition of a Liveness property, which states exactly to which subscribers a publication is notified to, must be necessarily defined considering both the subscription/unsubscription delays and the diffusion delay. Given a subscription interval $I(\sigma) \doteq [\sigma.s + T_{\mathrm{sub}}, \sigma.s + T_{\mathrm{ON}}(\sigma)]$, this property can be given as follows:

$$\forall (e_j(pub(x), t) \wedge (I(\sigma) \in H \text{ s.t. } I(\sigma) \supset [t, t + T_{\mathrm{diff}}])) \Rightarrow \exists e_{\sigma.p}(ntfy(x), t'') \in H$$

P3: Liveness

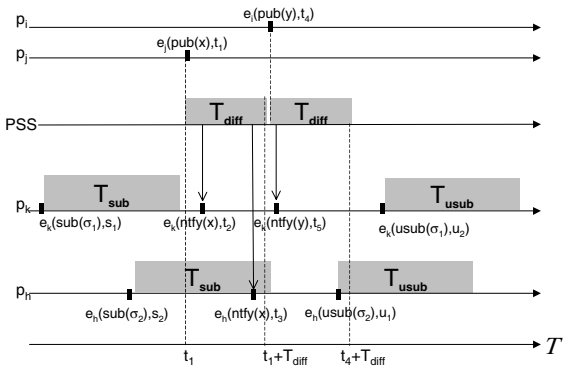


Figure 5. Global history with unexpected notify events.

This property states that the delivery of a notification can be guaranteed only for those subscribers that maintain stable subscriptions for the entire time taken by notification diffusion (diffusion delay). In other words, the Liveness property defines the NS system condition under which a notify event belongs to the global history. However, a notify event can also belong to the history even though this system condition is not verified. This is due to the uncertainty of the system state and on the diffusion time of information through the NS, as shown in the example depicted in Figure 5.

From application of the Liveness property, the only notify events *guaranteed* to be in the global history are $e_k(ntfy(x), t_2)$ and $e_k(ntfy(y), t_5)$, as, p_k has a subscription (matched by both x and y) stable during the whole diffusion of x and y. However, the global history also contains $e_h(ntfy(x), t_3)$. This depends on the fact that: (i) the subscribe operation for subscription σ_2 has taken effect *before* the $\sigma_2.s + T_{\text{sub}}$; (ii) the NS has made the diffusion relying on a state containing σ_2; and (iii) the diffusion to p_h has completed before $t_1 + T_{\text{diff}}$. Note that such 'lucky' conditions *may* occur but the probability of their occurrence is not equal to one.

2.5. Persistent notifications

The Liveness property introduced above can be extended by considering the possibility of the NS persistently storing notifications for a finite, non-zero amount of time. Persistence is usually exploited in distributed NS implementations to provide reliable delivery of notifications to processes: periodically processes request from each other a list of notifications they have sent and if a process cannot find a particular notification it requests its retransmission.

However, persistence can strongly influence the semantics of delivery if we assume that while the notification is maintained in the NS it can be delivered to processes that subscribe *after* it has been published. In particular, let us introduce a persistence interval $T_{\text{ON}}(x)$, representing the time interval a notification x is stored in the NS after it has been published by a process. If x is published at time t_x, all subscribers whose subscription becomes stable after t_x have to be notified. This behaviour is expressed

by the following extended definition of the Liveness property:

$$\forall(e_j(pub(x),t) \wedge (I(\sigma) \in H \text{ s.t. } (I(\sigma) \supset [t, t+T_{\text{diff}}] \vee I(\sigma) \cap [t+T_{\text{diff}}, t+T_{\text{ON}}(x)] \neq \emptyset))) \\ \Rightarrow \exists e_{\sigma.p}(ntfy(x), t'') \in H$$

P3a: Liveness (with persistent notifications)

In contrast to the previous definition of Liveness, in this case for the issue of a notification x to be guaranteed to a process p with a matching subscription σ, the subscription has to be maintained for an interval that (i) entirely contains the diffusion interval of x (as in P3) or (ii) intersects the interval during which x is persistently stored in the system.

As pointed out above, depending on the value of $T_{\text{ON}}(x)$, the semantics of the NS can be radically altered. In particular, we can consider three representative cases.

0-persistence: each published notification x expires as soon as it becomes available. Only a processes whose subscription matches x at the very moment of its publication are guaranteed for notification. This implementation scheme is very undemanding since it does not require storing notifications. However, it is subject to runs between concurrent *pub* and *sub* events, i.e. a subscriber may miss a notification x if its subscription is even slightly delayed with respect to the publication.

Δ-persistence: each published notification x expires after $\Delta > 0$ from the instant it becomes available. Garbage collection is performed by the pub/sub system on all expired notifications. This is more resilient to runs between the publisher and subscribers: a subscriber whose subscription 'is seen' in the system after Δ since x is available can still be satisfied. On the other hand, high values for Δ may provoke undesirable out-of-date notifications.

∞-persistence: each published notification x remains available in the pub/sub system for an indefinitely long time. When a subscriber installs a new subscription σ it will receive all the previously published and already available notifications that match σ. This implementation style obviously requires an ideal infinite memory to store all the notifications, since no notification is ever garbage collected.

This classification can be related to the non-deterministic behaviour deriving from the decoupling time between participants, which is one peculiar feature of the pub/sub paradigm. In general, the more an information item remains available in the system, the less non-determinism is experienced (for example, the effect of runs between publications and subscriptions is limited). Reduction of non-determinism increases the probability that an intended receiver gets the information. If the information is stored in a persistent way, non-determinism is completely removed and this probability goes to one [12]. Of course, this impacts on the size of the memory necessary within the system.

The three classes feature decreasing levels of non-determinism requiring increasing memory. However, we point out that this is not the case where a single class can be identified to be absolutely better than the others, as each class is suited to meet different application requirements. Hence we give examples of applications that may belong to each class.

An example of a 0-persistent application is a stock exchange system. Notifications represent instant values of stock quotes, expiring very quickly. Since the publication rate is high, missing notifications due to runs pose no problem, since subscribers can get new information after a short time.

An example of a Δ-persistent application is a daily news diffusion system. Each information item represents the daily issue of the news, having a lifetime of 1 day. Suppose issues are published every morning at 3 a.m., an issue will also be notified to processes submitting their subscription during the day.

An example of an application based on ∞-available information is a digital library where catalog updates are published as information items, kept available for future subscribers. A new subscriber will receive all the previous notifications in order to build its local copy of the catalog.

2.6. On the Liveness specification in dynamic systems

As we pointed out in [13], understanding and comparing different pub/sub systems is quite a difficult task due to informal and different semantics. From this stems the requirement of precisely defining formal semantics in terms of Safety and Liveness properties as in any distributed system.

To our knowledge, the first step in this direction was done in [8], where the author defines Safety properties which are actually similar to the ones defined in Section 2.2.1. However defining 'no bad thing can happen' is the easy part of the job in dynamic distributed systems (such as pub/sub applications). The tricky part is defining a property of progress of the whole system (i.e. the Liveness property) when processes behave independently and dynamically. In the classical (static) distributed system, Liveness constrains a system to *eventually* make progress on the global computation towards a certain target. Liveness is defined in [7] along this line: 'If a notification matching a set of active subscribers is published, then each subscriber will eventually be notified unless it cancels its subscription'. In other words, if a subscriber *never disconnects* with a subscription matched by a published notification, it eventually will be notified for that notification. Then nothing is guaranteed if the subscriber remains connected only for a certain time (even though this is a very long time!). Of course the assumption that a subscriber never disconnects is unrealistic in a pub/sub system.

Another example of the inadequacy of the Liveness property as defined in classical distributed systems comes from the crash-prone model. In this setting, the verification of the Liveness property ensures progress toward the termination of a computation. This usually requires assuming a minimum number of correct processes in the system (i.e. processes that never fail). If we make a parallel with a pub/sub system, this means making an assumption on the minimum number of subscribers that never disconnect. It is clear that in pub/sub such an assumption does not make any sense[‖]. A disconnected process *is not a bad process* as 'disconnection' is a matter of life in a pub/sub and not an undesirable event to cope with. A Liveness specification for this dynamic context should capture this normal behaviour.

Roughly speaking, our Liveness definition actually considers 'each notify event' as the target of our computation and defines timing assumptions under which a notify event is in the global history of the computation (i.e. this event has to be notified by the NS). This presence depends on three delays (T_{diff}, T_{sub}, T_{usub}), which abstract the dynamic behaviour of the computation and the NS implementation. Thanks to the latter point, our Liveness condition can also be used to compare different NS implementations. To explain this point, consider the following example. Suppose there are two

[‖] Defining a Liveness that gives guarantees if and only if a process never disconnects is the equivalent of not giving any guarantee.

different NSs managing the same set of clients and the same type of subscriptions and notifications. Moreover, suppose that:

1. a process publishes in both systems a notification matching an active subscription made by the same subscriber (a subscriber connected to both systems);
2. the subscription will remain active for two days after the publication but will be notified only by the first system.

In this scenario both systems satisfy Liveness as defined in [8], but are they equally good? It seems that the latter is a 'lazy' system, while the former is more reactive and effective. In the next section we show how this reactiveness can be measured to give an idea of the effectiveness of the implementation.

Let us finally remark that if T_{diff} was infinite, our definition of Liveness would not guarantee anything as the classical Liveness stated in [8]. However, in contrast to [8], when the three delays are finite (typical practical case) some subscription (satisfying conditions stated in the Liveness property) must be notified.

3. ANALYTICAL MODEL

The semantics identified through the abstract computational model allows us to reason on the practical consequences of T_{sub} and T_{diff}. Starting from the consideration that high values of the delay can prevent some subscribers from receiving notifications that were issued during the publication time, the idea is to take the probability that this could happen as a general performance parameter (namely the *notification loss*) for the NS.

In this section we provide a simple and general analytical model for the computation of the notification loss, given the implementation parameters T_{sub}, T_{diff}. We also specialize the model discussing how the practical deployment of a pub/sub system can influence such parameters and, subsequently, the notification loss. In particular, we identify three *operational parameters* that characterize the size of the system and the speed of the propagation of subscription and notifications. These parameters can be measured experimentally, subsequently obtaining T_{sub} and T_{diff}, hence the notification loss.

3.1. Measuring notification loss

Let x be a notification issued at time t and p be a generic process that has a subscription at time t matching x. We denote as d the probability that x is notified to p (*notification probability*). Therefore, the *notification loss* is the probability that x is not notified to p (i.e. $1 - d$), though p has a matching subscription.

Our analysis rests on the following assumptions:

1. the process p issues the subscription σ for a period $T_{\text{ON}} \geq T_{\text{sub}} + T_{\text{diff}}$;
2. any other subscription can only be issued by p after T_{usub} from the last *usub* operation;
3. the time a publication matching σ is issued is a uniformly distributed random variable defined over the subscription interval T_{ON};
4. all publishers have the same probability to generate a notification matching σ;
5. each notification is persistently stored in the NS for a time $T_{\text{ON}}(x)$.

The delay T_{usub} does not affect d because we assumed that a subscriber can issue a new subscription only after T_{usub} and, by definition, σ has been cancelled from the NS's configuration after this time interval.

In the following we give two expressions for d respectively considering $T_{\text{ON}}(x) = 0$ (volatile notifications) and $T_{\text{ON}}(x) > 0$ (persistent notifications).

3.1.1. *Volatile notifications*

Let t_{sub} be the time the process p issues the *sub* operation, t_{usub} the time when p issues the corresponding *usub* operation and t_{pub} the time the notification x is published. Then, the NS guarantees the delivery of any publication occurring at a time t_{pub} such that $t_{\text{sub}} + T_{\text{sub}} \leq t_{\text{pub}} \leq t_{\text{usub}} - T_{\text{diff}}$. This means, in fact, that the publication was issued when the subscription was stable and there was enough time for information diffusion to be completed. Moreover, for those publications such that $t_{\text{sub}} < t_{\text{pub}} < t_{\text{sub}} + T_{\text{sub}}$ as well as for those with $t_{\text{usub}} - T_{\text{diff}} < t_{\text{pub}} < t_{\text{usub}}$, there is also some probability of being notified.

For example, let us consider a distributed implementation of the NS as a network of brokers. Then, roughly speaking, it is possible that x was published by some process 'close' to p so that, after a delay $t < T_{\text{sub}}$, the portion of the NS involved in the diffusion of x towards p has already received the updates for correctly notifying x, as suggested by our simulations.

To model this aspect, we denote with $f(t)$ the probability density function that the NS notifies x to p, given that x was issued at time $t_{\text{pub}} = t_{\text{sub}} + \tau$, where $0 \leq \tau \leq T_{\text{sub}}$. Clearly, $f(t)$ must be a monotonically increasing function with $f(0) = 0^{**}$ and $f(T_{\text{sub}}) = 1$. In our experiments, $f(t)$ corresponds to the function plotted in Figure 11(a).

Also, $g(t)$, where $0 \leq \tau \leq T_{\text{diff}}$, is the probability density function that a notification published at a time $t_{\text{pub}} = t_{\text{usub}} - T_{\text{diff}} + \tau$ is notified to p. In a real implementation, this function captures the probability that the notification x reaches p before it unsubscribes for σ. The function $g(t)$ must be a monotonically decreasing function with $g(0) = 1$ and $g(T_{\text{diff}}) = 0$. In our experiments, $g(t)$ corresponds to the function plotted in Figure 11(b).

Figure 6 sketches the overall probability density function $P(t)$ that a notification x matching σ, issued at a time t inside T_{ON}, is notified by the NS.

Due to assumption (iv), $(1/T_{\text{ON}})\,\mathrm{d}t$ is the conditional probability that $t_{\text{pub}} \in [t, t + \mathrm{d}t]$ ($t_{\text{sub}} \leq t \leq t_{\text{usub}}$), given that information was published during the subscription interval. Moreover, $(P(t)/T_{\text{ON}})\,\mathrm{d}t$ is the probability that an event is published in the interval $[t, t + \mathrm{d}t]$ and it is notified.

Applying the total probability theorem, we can thus evaluate d as follows:

$$d = \frac{1}{T_{\text{ON}}}\left(\int_0^{T_{\text{sub}}} f(t)\,\mathrm{d}t + \int_{T_{\text{sub}}}^{T_{\text{ON}}-T_{\text{diff}}} 1\,\mathrm{d}t + \int_0^{T_{\text{diff}}} g(t)\,\mathrm{d}t\right)$$

which can also be rewritten as

$$d = \frac{T_{\text{ON}} - T_{\text{diff}} - T_{\text{sub}}}{T_{\text{ON}}} + \frac{1}{T_{\text{ON}}}\left(\int_0^{T_{\text{sub}}} f(t)\,\mathrm{d}t + \int_0^{T_{\text{diff}}} g(t)\,\mathrm{d}t\right) \tag{1}$$

**Actually, the value is slightly higher than 0, because there is the probability that x is issued by p itself. We discuss this later.

Figure 6. A sketch of $P(t)$, volatile notifications.

Note that d is directly proportional to the area of the curve depicted in Figure 6. Clearly, if $T_{\text{diff}} = T_{\text{sub}} = 0$, then the NS behaves as an ideal system with $d = 1$ (all publications are immediately notified).

3.1.2. Persistent notifications

The evaluation of d with persistent notification is basically the same as the method performed above for volatile notifications, just considering the fact that when notifications are stored inside the NS for a time $T_{\text{ON}}(x) > 0$[††], a subscriber also receives notifications issued *before* its own matching subscription was stable. More precisely, the NS guarantees the delivery of any notification occurring at a time t_{pub} such that (i) $t_{\text{sub}} + T_{\text{sub}} \leq t_{\text{pub}} \leq t_{\text{usub}} - T_{\text{diff}}$ (as in the previous case) or (ii) $t_{\text{sub}} + T_{\text{sub}} - T^x_{\text{ON}} \leq t_{\text{pub}} \leq t_{\text{sub}}$. In particular, notifications satisfying the latter condition will be delivered because as the subscription is surely stable at $t_{\text{sub}} + T_{\text{sub}}$, it will get notifications issued up to a time T^x_{ON} in the past.

For notifications issued between $t_{\text{sub}} - T^x_{\text{ON}}$ and $t_{\text{sub}} + T_{\text{sub}} - T^x_{\text{ON}}$, the probability of their delivery is conditioned to the probability that the subscription σ is stable before a time T^x_{ON} in the future. Hence, the trend of $P(t)$ will simply follow that of the function $f(t)$. Finally, persistent notifications do not have any influence on notifications published in proximity to the unsubscribe of σ.

[††]We will use a simplified notation, writing T^x_{ON} instead of $T_{\text{ON}}(x)$.

The analytical evaluation of d obviously resembles that obtained for volatile notifications, but with an added term accounting for the probability of 'past' notifications:

$$
\begin{aligned}
d &= \frac{1}{T_{\mathrm{ON}} + T_{\mathrm{ON}}^{x}} \left(\int_0^{T_{\mathrm{sub}}} f(t)\,\mathrm{d}t + \int_{T_{\mathrm{sub}}}^{T_{\mathrm{sub}}+T_{\mathrm{ON}}^{x}} 1\,\mathrm{d}t + \int_0^{T_{\mathrm{sub}}} f(t)\,\mathrm{d}t + \int_{T_{\mathrm{sub}}}^{T_{\mathrm{ON}}-T_{\mathrm{diff}}} 1\,\mathrm{d}t + \int_0^{T_{\mathrm{diff}}} g(t)\,\mathrm{d}t \right) \\
&= \frac{1}{T_{\mathrm{ON}} + T_{\mathrm{ON}}^{x}} \left(T_{\mathrm{ON}} - T_{\mathrm{diff}} - 2T_{\mathrm{sub}} + T_{\mathrm{ON}}^{x} + 2\int_0^{T_{\mathrm{sub}}} f(t)\,\mathrm{d}t + \int_0^{T_{\mathrm{diff}}} g(t)\,\mathrm{d}t \right) \\
&= 1 - \frac{2T_{\mathrm{sub}} + T_{\mathrm{diff}}}{T_{\mathrm{ON}} + T_{\mathrm{ON}}^{x}} + \frac{1}{T_{\mathrm{ON}} + T_{\mathrm{ON}}^{x}} \left(2\int_0^{T_{\mathrm{sub}}} f(t)\,\mathrm{d}t + \int_0^{T_{\mathrm{diff}}} g(t)\,\mathrm{d}t \right) \qquad (2)
\end{aligned}
$$

This expression is valid for $T_{\mathrm{ON}}^{x} > T_{\mathrm{sub}}$.

3.2. Analytical results

In order to provide an analytical expression for d, the two functions f and g have to be specified. The shape of the curves will obviously be determined by the mechanisms used internally by the NS for routing subscriptions and notifications. These mechanisms are dependent on three operative parameters: N representing the size of the system in terms of the number of entities it is composed of; s and r, respectively the subscription and the notification update rates, representing the efficiency of the routing processes. In the following, we first consider a general distributed implementation for the NS and give an analytical evaluation for the two functions in this case. Then, we investigate how the operative parameters of the NS are related to the implementation parameters T_{sub} and T_{diff}.

3.2.1. NS reference model

We consider a general case where a *subscription routing* algorithm is used to make all participants in the system aware of each active subscription. Thus, a subscription is stable after its routing process is terminated. That is, the subscription delay is the *maximum* convergence time of the subscription routing algorithm. The f function should represent the variation over time in the number of NS participants that received a subscription update. The maximum convergence time is obtained when all subscriptions are flooded within the entire system, reaching all its participants.

Once the subscription is stable, the corresponding subscriber can be contacted when a matching notification is published. A *notification routing* algorithm is then used to propagate the notification among brokers in order to reach all the matching subscribers for such a notification. During the routing process, a broker receiving a notification from another broker has the opportunity to issue the notify operation to its local matching subscribers. Moreover, it can forward the notification to the other brokers that can have some active subscription. Following the Liveness property we defined in Section 2.2, the set of subscribers to be reached by a notification x must comprise all those subscribers whose subscription matches x and is stable at publication time. Then, the diffusion delay is the *maximum* time interval required to diffuse a notification from a broker serving the source publisher to all the interested subscribers in the set. The g function should represent the variation over time of the number of interested NS participants that received the notification. Again, the maximum diffusion time is obtained when a notification is flooded through all the participants of the system.

Considering naive flooding as a routing algorithm for subscriptions and notifications that is allowed to be as general as possible in the derivation of the two functions. In other words, the described model represents the worst-case scenario of a wide range of NS implementation solutions.

3.2.2. Expressing f(t) and g(t)

The probability that a broker receives a subscription/notification $f(t)$ and $g(t)$ can be represented as the number of brokers in a NS that receive a subscription/notification, at time t. Message diffusion through a flooding algorithm follows an epidemic-like behaviour, that is, each participant transmits a subscription/notification only to a subset of other participants that in turn forward it to other participants. Thus, the following analysis is inspired by the mathematical theory of epidemics [14], often used in the context of a gossip-based multicast algorithm to predict the average percentage of participants that will be reached by a multicast message [15].

Let $f(t)$ be the number of NS participants that have received a subscription at time t and N the overall number of brokers in the system. The variation in time of this function is proportional to the number of brokers that have not received the subscription yet:

$$\frac{\mathrm{d}f(t)}{\mathrm{d}t} = \alpha(t)\,(N - f(t))$$

The value of $\alpha(t)$ is at the same time proportional to $f(t)$, because the more participants in the system that are aware of the subscription, the faster it spreads. Hence,

$$\frac{\mathrm{d}f(t)}{\mathrm{d}t} = s\frac{f(t)}{N}(N - f(t))$$

We refer to the constant s as the *subscription update rate*, because it represent the speed of diffusion of an update. s depends on parameters of the NS, such as the time required for a single update, the number of participants updated by a single participant, the time required to process a subscription within each participant.

Solving the above differential equation leads to the following solution:

$$f(t) = \frac{1}{1 + c\,\mathrm{e}^{-st}}$$

where c is an arbitrary constant. Since at $t = 0$ only one broker is aware of the subscription (the one that sends it), we can impose $f(0) = 1$ and subsequently obtain $c = N - 1$. The final expression for $f(t)$ is then

$$f(t) = \frac{1}{1 + (N - 1)\,\mathrm{e}^{-st}} \tag{3}$$

Following a similar procedure, we obtain the following expression for $g(t)$:

$$g(t) = 1 - \frac{1}{1 + (N - 1)\,\mathrm{e}^{-rt}} \tag{4}$$

where r is the *notification update rate*. In general $r \neq s$ because the different algorithms used for routing subscriptions and notifications can result in different diffusion rates. The plots for $f(t)$ and $g(t)$ are shown in Figure 7.

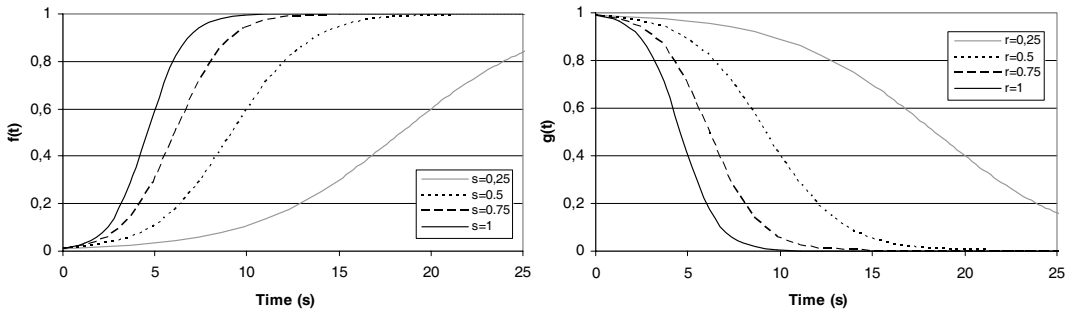


Figure 7. Analytical trends of $f(t)$ and $g(t)$.

s and r are two important parameters that significantly affect the behaviour of the system. Unfortunately, an analytical evaluation is not a trivial task. From an experimental measure in a system with 100 nodes configured in a dense topology where all nodes are all physically close to each other, we obtained update rates of $s = 0.98$ and $r = 1.2$. Details are given in Section 4. From the analytical study in Section 3.3, it turns out that a system with update rates greater than 1; 5 can be considered 'fast', in the sense that with average subscription intervals it can ensure at least 80% of the notification.

3.2.3. *Expressing d*

The next step is substituting the above expressions into Equation (1) to obtain a general expression for d that depends only on the external operative parameters of the NS.

First we calculate the integrals of $f(t)$:

$$\int_0^{T_{\text{sub}}} f(t)\,\mathrm{d}t = s\,T_{\text{sub}} + \frac{1}{s}\ln\frac{1+(N-1)\,\mathrm{e}^{-sT_{\text{sub}}}}{N}$$

With $1/(1+(N-1)\,\mathrm{e}^{-sT_{\text{sub}}}) \simeq 1$, we obtain

$$\int_0^{T_{\text{sub}}} f(t)\,\mathrm{d}t = sT_{\text{sub}} + \frac{1}{s}\ln\frac{1}{N}$$

Analogously,

$$\int_0^{T_{\text{diff}}} g(t)\,\mathrm{d}t = (1-r)T_{\text{diff}} - \frac{1}{r}\ln\frac{1}{N}$$

Substituting this expression into Equation (1), results in the following expression for d:

$$d = 1 - \frac{(1-s)T_{\text{sub}} + rT_{\text{diff}}}{T_{\text{ON}}} + \frac{1}{T_{\text{ON}}}\left(\frac{1}{s} - \frac{1}{r}\right)\ln\frac{1}{N} \tag{5}$$

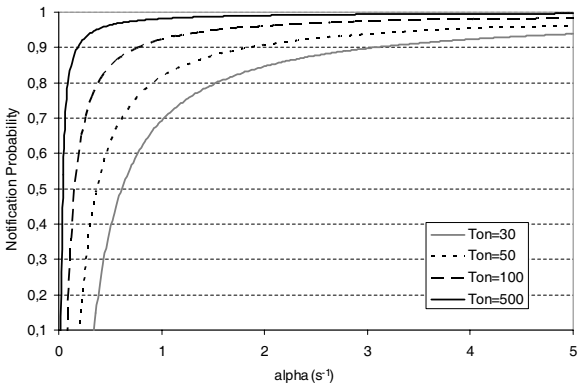


Figure 8. Notification probability versus update rate.

From the above definitions of T_{sub} and T_{diff} it is clear that it should be $f(T_{\text{sub}}) = 1$ and $g(T_{\text{diff}}) = 0$. These two values cannot be determined directly from the analytical expressions of $f(t)$ and $g(t)$ because the functions only asymptotically tend to 1 and 0, respectively. However, considering the meaning of the functions, representing the fraction of participants reached by a message, it makes sense to assume only discrete values for them. In particular, the minimal value for the fraction can be $1/N$, then T_{sub} and T_{diff} can be chosen approximately as any values such that $f(T_{\text{sub}}) \geq 1 - 1/N$ and $g(T_{\text{diff}}) \geq 1/N$. Considering equalities, we obtain the following approximate expressions:

$$T_{\text{diff}} \simeq \frac{1}{r} \ln(N-1)^2$$

$$T_{\text{sub}} \simeq \frac{1}{s} \ln(N-1)^2$$

This allows us to rewrite d only in terms of s, r and N:

$$d \simeq 1 - \frac{1}{T_{\text{ON}}} \left(\frac{1}{s} + \frac{1}{r} \right) \ln N \tag{6}$$

3.3. Discussion

In the following we discuss the analytical values obtained by the application of Equation (6), studying the influence of the various operative parameters over the notification probability. We will consider, for simplicity, a single parameter α for the update rates (i.e. $\alpha = r = s$). Subsequently, we assume a single stabilization time T_α for both subscriptions and notification.

Figure 8 plots the notification probability against the update rate, with different fixed values of T_{ON} and $N = 100$. The sensitivity of the system to changing values of α strongly depends on the expected value of T_{ON}. With long subscription times ($T_{\text{ON}} > 500$), a slowly responding system ($\alpha < 0.2$) can offer an acceptable level of performance ($d > 0.9$). On the other hand, rapidly changing subscriptions

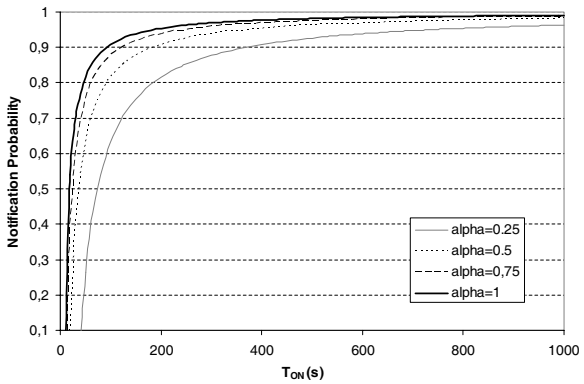


Figure 9. Notification probability versus subscription interval.

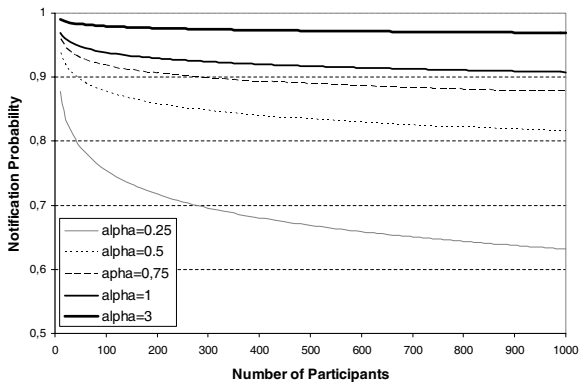


Figure 10. Notification probability versus number of participants.

($T < 50$) require a vary fast responding system. When $\alpha < 1$, d follows a steep curve. This means that small variations in α result in significant changes in d. However, the plot shows that with values of T_{ON} starting from about 100 s, the system reaches relatively high values for d with α higher than about 0.5.

This is more evident in the plot given in Figure 9, showing the notification probability against the variation of T_{ON}, for different values of α and $N = 100$. Moreover, this plot demonstrates that the more critical behaviour, independently of α, occurs when T_{ON} is lower than 60 s. In this case, the system is highly unstable, requiring careful fixing to achieve acceptable values of d.

Finally, Figure 10 plots the influence of the number of participants N over the notification probability, with different values of α (while T_{ON} is fixed at 150). This plot shows that for α higher

than 1.5, the system can be considered 'fast' because on increasing the number of participants, the notification probability is always higher than 0.9.

Overall, we can say that the analytical model presented in this section allows us to make important high-level considerations about the general behaviour of a NS. However, a further step is required to fully consider such results trustworthy, i.e. achieve an experimental confirmation. This is the subject of the following section.

4. SIMULATION STUDY

In this section we present the experimental results derived from a simulation of the execution of a real pub/sub system. The objective of the study is to evaluate the actual fraction of the delivered notifications and compare it with the analytical result obtained from the model.

4.1. Simulation details

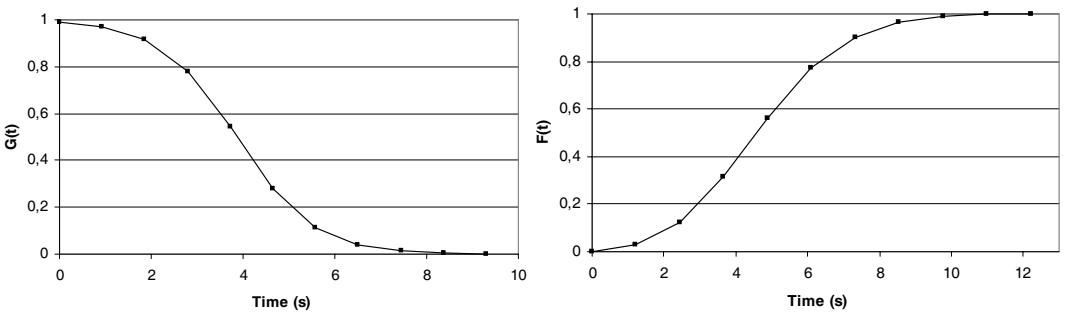
We carried out our experiments by implementing a prototype of a distributed NS made up of a set of distributed brokers, communicating through point-to-point application-level connections. The system is based on the content-based routing algorithm (CBR) for acyclic peer-to-peer topologies introduced in SIENA [1]. The key idea is to diffuse subscriptions in order to build paths for routing events, so that parts of the network with no interested subscribers are excluded from event diffusion. Simulations were performed by running the NS prototype on top of the J-Sim [16] real-time network simulator. We give a more detailed explanation both of the CBR algorithm and of our prototype implementation in the following section.

Network-level topologies are generated using the Georgia Tech ITM topology generator [17]. All topologies follow the Transit-Stub model. The application-level network (i.e. the distribution of the distributed brokers over the network nodes and the links between them) is self generated by our prototype and follows a random topology that is not influenced by the underlying network topology. This reproduces the typical shift between application level and network level that occurs in the real deployment of overlay networks. Experiments featured 100 participants (brokers) deployed above networks with 100, 250 and 500 nodes. We considered different deployment scenarios in order to alter the values of the update rate.

4.2. Simulation results

4.2.1. Update rates

Figures 11(b) and 11(a) show the results of the experiments over subscription and event diffusion times, with a 100-broker NS deployed over a 100-node network. In particular, Figure 11(b) plots the fraction $F(t)$ of brokers at time t that receive a new subscription issued at time 0, and Figure 11(a) plots the fraction $G(t)$ of brokers that *have not* received a notification issued at time 0. In practice, these curves are the experimental results of the $f(t)$ and $g(t)$ functions, as defined in the previous section. The shape of the curves coincides with those in Figure 7, confirming that epidemics models are accurate at representing information diffusion processes.



(a) (b)

Figure 11. Experimental values of T_{diff} and T_{sub} for 100 participants over a 100-node network.

Table I. Experimental values for update rates.

Network nodes	s	r	T_{sub}	T_{diff}
100	0.97	1.21	12.20	9.95
250	0.62	0.80	19.06	14.03
500	0.47	0.58	25.44	20.44

Plots were obtained as follows (we consider only T_{sub} as the same can be applied for T_{diff}). We let a subscription start from a random broker and be delivered to all the other brokers. This is obviously the worst case scenario that can be encountered for routing. In turn, each broker acts as the source for the subscription. The maximum time required for *all* brokers to receive the subscription, considering all the possible source brokers, corresponds to T_{sub}. Experiments were repeated five times over different network topologies. The same experiments were carried out with networks of 250 and 500 nodes, producing the same trends, though obviously with different values for T_{sub} and T_{diff}.

The coincidence of the analytical and experimental curves allows us to evaluate the update rates for subscriptions and notifications. These are the values of s (respectively r) that when substituted into Equation (3) (respectively (4)) lead to a curve that has the same integral as the one obtained experimentally. The results are reported in Table I.

Note the slight difference between the update rates for subscriptions and notifications. In other words, notification routing turns out to be faster than subscription routing, though they both run on the same topology and travel to all the brokers. This is due to the fact that at each hop, the subscription routing algorithm has to update the local routing data structures, taking more time than the matching operation performed by the notification routing to determine the next hop. Considering that, along its execution, the NS is supposed to manage thousands of subscriptions, one can expect the update rate to be subject to changes over time.

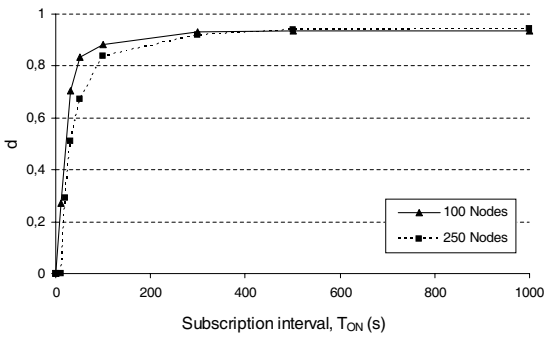


Figure 12. Experimental values of d.

4.2.2. *Notification probability*

Figure 12 shows the number of notifications actually achieved for notifications published at a broker while another broker in the system has an active, matching subscription. These results were obtained as follows: we generated a subscription on a randomly chosen broker, and after a time T_{ON}, a corresponding unsubscription. During the subscription interval, a notification was produced in another random broker. The exact publication time was randomly chosen and followed a uniform distribution inside the subscription interval. We repeated this process over 250 different random subscriber–publisher couples and executed five runs of the experiment, each on a different network topology. The whole process was repeated for different values of T_{ON} and over 100 and 250 network nodes, obtaining the curves depicted in Figure 12.

The number of notification losses surprisingly does not tend to 0, even for high values of T_{ON}. This demonstrates that the notification loss phenomenon is an important issue in pub/sub systems.

For a final validation of our analytical model, we have to compare the curve obtained experimentally with the one resulting from Equation (1) by substituting the values for s and r obtained from a previous experiment. The percentage error between the experimental curve and the analytical one is shown in Figure 13. Negative values of the error mean that the analytical value is higher than the experimental value. The plot shows that, excluding lower values of T_{ON} for which the system unpredictability results in unstable error values, the error is within a low 6%. This confirms the overall reliability of the analytical evaluations, except for the smallest values of T_{ON} that are more sensitive to small variations in the results.

Thus, we have shown that our analytical model is able to predict the actual behaviour of a NS, with a small percentage error. Application of the model with values of s and r that are derived experimentally gives a general estimation of the probability of notification that accurately reproduces the experimental results.

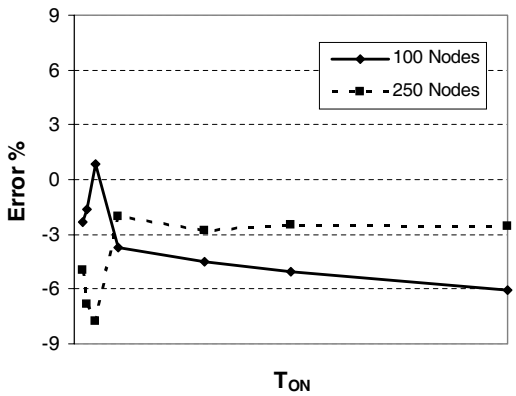


Figure 13. Comparing simulation and analysis: evaluation error.

5. RELATED WORK

To the best of our knowledge, the only other research contribution examining a formal definition of the semantics of a pub/sub system has been given by Mühl in his PhD thesis [8]. The thesis presents a framework for specifying a general pub/sub computation and for proving the formal correctness of an important class of diffusion algorithms (*content-based routing algorithms*). Pub/sub computation is abstracted by a sequence of global states interleaved by operations. The characterization of the correct computations of the system is given with respect to the global state. At this level of abstraction, the effect of the delay in the execution of operations is not represented: Safety and Liveness properties are defined as if any subscription update and notification takes effect instantly.

When defining the framework for content-based routing, the author shows that the assumptions may never be satisfied when more update processes occur concurrently. The Liveness property is then shown to be guaranteed only for those subscriptions that have been propagated in the whole system (i.e. that are stable, in our terminology).

Though the conclusions are obviously similar, our approach is quite different compared with [8]. First of all, the main objective of the work is different: we propose a model to analytically characterize the 'quality' of a generic pub/sub system, while Mühl proposes a framework that formally shows the correctness of a class of diffusion algorithms. Besides, our model presents two characterizing aspects: (i) explicitly introducing the pub/sub system in the model; and (ii) considering the notion of time in the specifications of the system. These two aspects allow us to point out the non-deterministic behaviour of pub/sub systems at a specification level, i.e. independently from any specific implementation such as the deployment type or the diffusion algorithm. On the other hand, we do not give any formal proof but provide only a probabilistic analysis.

Concerning the mathematical analysis itself, it was inspired from the mathematical theory of epidemics [14], recently applied to model information diffusion phenomena like probabilistic reliable multicast [15,18,19] or mobile *ad hoc* networks [20].

6. CONCLUSIONS

In this paper we presented a formal model for the general specification of a pub/sub system. With respect to another similar research contribution [8], our model gives an implementation-independent characterization of pub/sub semantics that points out the unpredictable behaviour deriving from the inherent decoupling among participants.

Our model presents two characterizing aspects: (i) explicitly introducing the pub/sub system in the model rather than considering only the semantics at processes; and (ii) considering the notion of time in the specifications of the system. This characterization allows us to point out the non-determinism of pub/sub systems at the specification level, i.e. independently from any implementation specific issues such as the deployment type or the diffusion algorithm. On the other hand, we do not exploit the specification to give formal proof of the pub/sub system algorithms but only provide a simple probabilistic analysis.

In our opinion this analytical performance model can be used as a rule of thumb for users and designers of a pub/sub system to predict the behaviour of their applications, easily identifying the critical implementation aspects that influence the overall performance. This can represent a more practical solution than other complex and time-consuming analyses (such as simulations) that, due to the many dynamic aspects involved, are not guaranteed to give a precise performance assessment.

REFERENCES

1. Carzaniga A, Rosenblum DS, Wolf AL. Design and evaluation of a wide-area notification service. *ACM Transactions on Computer Systems* 2001; **3**(19):332–383.
2. Banavar G, Chandra T, Mukherjee B, Nagarajarao J, Strom RE, Sturman DC. An efficient multicast protocol for content-based publish–subscribe systems. *Proceedings of the 19th International Conference on Distributed Computing Systems*, Austin, TX, 31 May–4 June 1999. IEEE Computer Society Press: Los Alamitos, CA, 1999; 262–272.
3. Cugola G, Di Nitto E, Fuggetta A. The JEDI event-based infrastructure and its applications to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering* 1998; **27**(9):827–850.
4. Segall B, Arnold D. Elvin has left the building: A publish/subscribe notification service with quenching. *Proceedings of the 1997 Australian UNIX and Open Systems Users Group Conference*, 1997.
5. Preotiuc-Pietro R, Pereira J, Llirbat F, Fabret F, Ross K, Shasha D. Publish/subscribe on the Web at extreme speed. *Proceedings of the 26th International Conference on Very Large Databases (VLDB 2000)*, Cairo, Egypt, 10–14 September 2000; 627–630.
6. Pietzuch P, Bacon J. Hermes: A distributed event-based middleware architecture. *Proceedings of the 1st International Workshop on Distributed Event-Based Systems (DEBS'02)*, 2002; 611–618.
7. Oki B, Pfluegel M, Siegel A, Skeen D. The Information Bus: An architecture for extensive distributed systems. *Proceedings of the 14th Symposium on Operating Systems Principles*, December 1993. ACM Press: New York, 1993; 58–68.
8. Muhl G. Large-scale content-based publish/subscribe systems. *PhD Thesis*, Technical University of Darmstadt, 2002.
9. Ratnasamy S, Handley M, Karp R, Shenker S. Application-level multicast using content-addressable networks. *Proceedings of ACM SIGCOMM* 2001; **8**(4).
10. Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H. Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking* 2003; **11**(1):17–32.
11. Rowston A, Druschel P. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *Proceedings of Middleware '07*, 2001.
12. Tanenbaum A, van Steen S. *Distributed Systems: Principles and Paradigms*. Prentice-Hall: Englewood Cliffs, NJ, 2002.
13. Baldoni R, Contenti M, Virgillito A. The evolution of publish/subscribe systems. *Proceedings of the 9th IEEE Workshop on Future Trends of Distributed Computing* (*Research and Position Papers*, vol. 2584), Schiper A, Shvartsman AA, Weatherspoon H, Zhao BY (eds.). Springer: Berlin, 2003.

14. Demers A, Greene D, Hauser C, Irish W, Larson J. Epidemic algorithms for replicated database maintenance. *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, 1987; 1–12.
15. Birman KP, Hayden M, Ozkasap O, Xiao Z, Budiu M, Minsky Y. Bimodal multicast. *ACM Transactions on Computer Systems (TOCS)* 1999; **17**(2):41–88.
16. J-Sim. http://www.j-sim.org [January 2005].
17. Zegura EW, Calvert KL, Bhattacharjee S. How to model an internetwork. *IEEE Infocom* 1996; **2**:594–602.
18. Eugster PTh, Guerraoui R. Probabilistic multicast. *Proceedings of the 3rd IEEE International Conference on Dependable Systems and Networks (DSN 2002)*, 2002; 313–322.
19. Kermarrec A-M, Massoulié L, Ganesh AJ. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems* 2003; **14**(3):248–258.
20. Khelil A, Becker C, Tian J, Rothermel K. An epidemic model for information diffusion in MANETs. *Proceedings of the 5th ACM International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems*, 2002; 54–60.

