Measuring Notification Loss in 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
email: {baldoni,beraldi,tucci,virgi}@dis.uniroma1.it

Abstract

A publish/subscribe communication system (PSS) realizes a many-to-many anonymous interaction among its participants. Producers of information (publishers) issue notifications to the PSS. These are delivered by the PSS to all subscribers that declared interest in it. However, this decoupled form of interaction introduces delays between i) the production of a notification and its delivery to subscribers (diffusion delay) and ii) the declaration of interest by a subscriber and its registration in the PSS (subscription/unsubscription delay). Such delays could lead to notification loss scenarios where an event is not delivered to an intended subscriber even though it was issued when the subscription was active. This paper studies this notification loss phenomenon by presenting a simulation study of a PSS and an analytical model. The latter measures the percentage of notifications guaranteed by a PSS implementation to a subscriber. To our knowledge this is the first paper that addresses such a QoS issue. The model is based on a formal framework of a distributed computation. The framework abstracts the PSS through the two delays, defining safety and liveness properties that precisely characterize the semantics of the PSS.

1. Introduction

Communication systems following the publish/subscribe (pub/sub) paradigm have experienced a relevant gain in popularity during the last 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 form of *notifications*), that 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 from publisher, but they are indirectly addressed through the content of notifications. That is, subscribers express their interests by issuing *subscriptions* for specific notifications, independently from the publishers that produces them, and then they are *asynchronously* notified for all notifications, submitted by any publisher, that match their subscriptions.

Since pub/sub has been largely recognized as an effective approach for information diffusion, lots of pub/sub based systems, both research contributions [3, 2, 4, 9, 8] and commercial products [7], has been presented and are actually used in several application contexts. From the research side, on one hand, a lot of work has been done in this field from the software engineering community, focusing on scalability, efficient information delivery or efficient and expressive information matching. On the other hand, only few contributions exist [6] that define, for example, which is the computational model underlying a PSS and, most important, which are the delivery guarantees that a PSS has to ensure to applications. This step is necessary for carrying out, for example, an analytical study of the performance of a PSS which is the base of a rigorous QoS policy. According to PSS users, the lack of this rigorous approach is currently one of the main pitfall of a PSS which limits its applicability, for example, to mission critical systems.

In this paper we propose a computational model based on a PSS where the latter 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, namely the subscription delay (denoted T_{sub}) and the diffusion delay (denoted T_{diff}), which characterize, respectively (i) the non-atomicity of the subscription/unsubscription storage (ii) the non-instantaneous diffusion of a notification. These delays depend of course on the implementation of the PSS (e.g. centralized, network of brokers, etc.) and model all the delays that could arise in the processing of subscriptions and notifications from the PSS, due both to computation and to network.

This model produces a global history of the computation, on 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 from the delays. Then we propose a liveness property which states when a notify event belongs to the history: this is affected by the interval 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 publish [1]. In other words our liveness prop-

1 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 PSS operations which take time to take effect, this sentence should be reworded as follows "under some timing conditions something good happens".

erty gives a timing condition implying, given a publish event, the presence of a corresponding notify event in the global history.

This liveness condition gives us the opportunity to define a measure of the notification loss of a PSS. More specifically, 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 at t. Therefore, $1 - d$ represents the percentage of notification losses[2]. The system behaves ideally if d is equal to 1 (each notification issued at time t is delivered at all matching subscriptions active at t). We study d as a function of the subscription delay and of the diffusion delay.

The paper presents a simulation study of a distributed PSS (similar to the Siena [3] system) deployed over large-scale wide-area networks. The simulation has been carried out by integrating J-SIM real-time network simulator (developed by The Ohio State University [5]) and GT-ITM network topology model (developed by Georgia Tech [10]). Results allow to estimate the values of T_{diff}, T_{sub} and d in realistic network settings.

We finally propose an analytical model for a PSS that starting from an estimate of T_{diff} and T_{sub} is able to determine d (note that T_{diff} and T_{sub} can be estimated on-the-fly by a system manager). Even though the granularity of this model is coarse, we believe that it can be very useful from two perspectives: i) the designer of a PSS can quickly evaluate the overall performance of the system; ii) users can predict the probability of receiving a notification. We show that the model confirms the results obtained by the simulation.

The paper is structured as follows: Section 2 introduces the formal framework, Section 3 presents the simulation study and the analytical model and Section 4 concludes the paper.

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 through a publish/subscribe communication system (PSS). Processes are decoupled in the sense that they never communicate directly within each other but only through the PSS.

2.1. Notifications and Subscriptions

Processes can act both as producers and consumers of information, taking on the role of *publishers* and *subscribers*, respectively. We consider the information produced and consumed in form of *notifications*, made up of a set of attribute-value pairs. Each attribute has a *name*, a simple character string, and a *type*, one of the common primitive data types defined in programming languages or query languages (e.g. integer, real, string, etc.).

2 Note that we assume an underlying reliable communication system, so a notification loss is related to the inability of the PSS system to deliver instantaneously information from a publisher to a subscriber.

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 to all publications declared through the *filter* ϕ. A filter ϕ is a query expression composed by 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 an information x matches a filter ϕ is called $matching$ ($x \sqsubset \phi$). We say that x matches a subscription σ if it matches $\sigma.\phi$ ($x \sqsubset \sigma.\phi$).

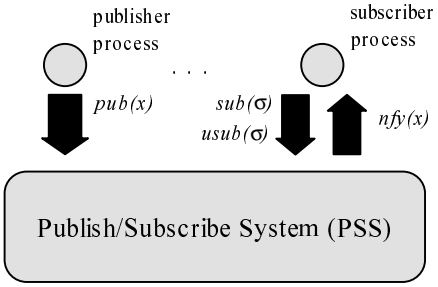


Figure 1. A publish/subscribe system interaction

2.2. Process-PSS Interaction

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

We denote as $op = \{sub(\sigma), usub(\sigma), pub(x), ntfy(x)\}$ respectively 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 (Figure 1).

Then, the operations $sub(\sigma), usub(\sigma), pub(x)$ are issued by a process and executed by the PSS, while $ntfy(x)$ is issued by the PSS 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 PSS. Note that the PSS issues $ntfy(x)$ on the set of processes computed after the matching operation.

2.3. 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 specifications.

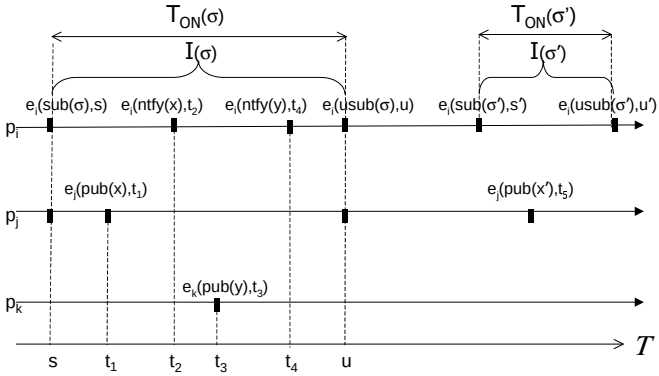


Figure 2. Global History respecting Safety

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 publish and subscribe system, 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 occurred 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 $t_1 < t_2 < \ldots < t_m$). 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 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 time interval as $T_{ON}(\sigma)$. A subscription interval is defined also 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 be consequently infinite. Figure 2 shows an example of global history of three processes, with two subscription intervals $I(\sigma)$, $I(\sigma')$ and their corresponding T_{ON}.

2.4. Safety properties

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

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

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 that ensures the notify events are not invented by a process. This is taken into account by the Validity property which states as follows:

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

P2: Validity

The computation in Figure 2 also respects Validity: then 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 are defined, it is interesting to understand under which conditions a notify event should be generated, i.e. to define a Liveness property. As just said the global history in Figure 2 satisfies safety. However supposing x' matches σ', should we expect that the PSS 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 PSS is physically built. This is actually the aim of the following section.

2.5. PSS Implementation Parameters

The PSS 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 PSS, there exists an actual PSS physical implementation (e.g. centralized, distributed, network of brokers etc.) that performs the desired functionality. In order to capture the behavior of *any* PSS 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

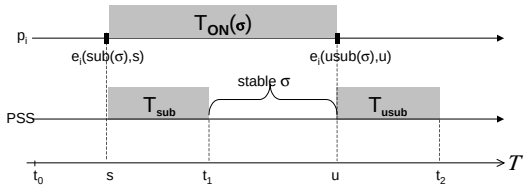


Figure 3. Subscription/unsuscription delays

publish operation issued by a process. These parameters model the time required for the internal processing at the PSS and the network delay elapsed to propagate subscriptions and notifications, in a distributed implementation. Let us finally assume that any message sent by a processes of a PSS implementation uses reliable channels.

Subscription/unsubscription delays. When a process issues a subscribe/unsubscribe operation, the PSS is not immediately aware of the occurred event. In other words, at an abstract level, the registration (resp. cancellation) of a subscription takes a certain amount of time to be stored into the PSS. This time encompasses for example the update of the internal data structures of the PSS and the network delay due to the propagation of the operation among all the entities constituting the PSS. 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 into the PSS. As an example, in a distributed implementation of a PSS, this means *each entity* implementing the PSS after this threshold of time is aware of the registration/cancellation operation.

We denote such delay as T_{sub} for subscribe operations and as 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 $s < t \leq s + T_{sub}$ [3]. The same holds for unsubscribe operations, i.e. an unsubscribe operation, issued at time u, takes effect at a time t' such that $u < t' \leq u + T_{usub}$.

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

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

Diffusion delay. As soon as a publication is issued, the PSS 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 PSS implementation, the diffusion can be performed in several ways, using for example a routing protocol on a set of distributed brokers. Without entering implementation details, we can say that this operation takes a certain amount of time *during* which the PSS computes and issues notify operations to interested subscribers, i.e. diffusion takes a non-zero 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. $\sigma \in S(t)$. Then the PSS 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 PSS, 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 T_{diff}.

To clarify the meaning of the diffusion delay see Figure 4. For sake of simplicity and without loss of generality we assume that the communication delay between a process and the PSS is zero. This implies that (i) a notification published by a process immediately gets the PSS, and (ii) if PSS 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 PSS, during T_{diff}, notifies the interested subscribers. Supposing that $\{p_j, p_k, p_h\}$ is the set of interested subscribers then $T_{diff} = max\{\Delta_j, \Delta_k, \Delta_h\} = \Delta_h$. Let us remark that each generic interested subscriber p_i is notified in 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 PSS's state. However, *how and when* the state is considered, is implementation dependent. Moreover, *the state can change during the diffusion*. In the following these aspects will be clarified.

2.6. Liveness Property

The concept of "interested subscriber" has been till now considered quite intuitively. The desirable PSS behavior 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.

3 In our framework we reasonably assume for each subscriptions σ we have $T_{ON}(\sigma) > T_{sub}$.

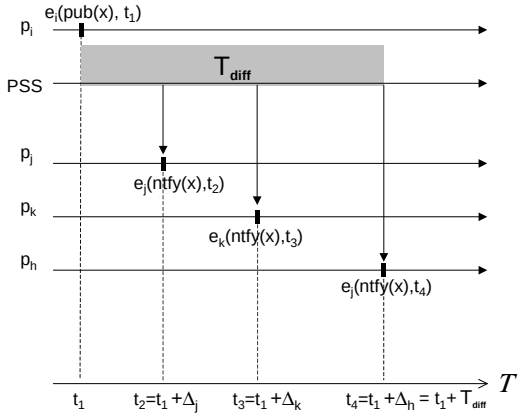


Figure 4. Example of Diffusion

$x \sqsubset \sigma.\phi$ and $\sigma.s \leq t \leq \sigma.s + T_{ON}(\sigma)$. However the PSS system is surely aware of the subscription σ by p_i only when the subscription becomes stable, i.e. at time $\sigma.s + T_{sub}$. Then, at first check, an interested subscriber seems to be a process whose subscription is stable (i.e. belonging to the PSS state), at the moment in which the matching information is published, i.e. $\sigma.s + T_{sub} \leq t \leq \sigma.s + T_{ON}(\sigma)$.

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 behavior of the system. Let us point out 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 $\sigma.s + T_{sub} \leq t \leq \sigma.s + T_{ON}(\sigma)$. However, if PSS *is able to notify* x at p_i *only* at a time $t' = t + \Delta_i$ such that $t' > \sigma.s + T_{ON}(\sigma)$ then p_i will discard x as it is not longer interested to x.

Then, the definition of a liveness property, that states exactly to which subscribers a publication is notified to, must be necessarily defined considering both the subscription/unsubscription delays and the diffusion delay. This property can be stated as follows:

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

P3: Liveness

This property states that the delivery of a notification can be guaranteed only for those subscribers that maintain their subscriptions stable for the entire time taken by notification diffusion (diffusion delay). In other words, Liveness property defines the PSS 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 on the system state and on the diffusion time of an information through the PSS, 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 contains also $e_h(ntfy(x), t_3)$. This depends on the fact that (i) the subscribe operation for subscription σ_2 has taken effect *before* the $\sigma_2.s + T_{sub}$, (ii) PSS has made the diffusion relying on a state containing σ_2, and (iii) the diffusion to p_h has completed before $t_1 + T_{diff}$. Note that such "lucky" conditions *may* occur but the probability of its occurrence is not equal to one.

2.7. On the liveness specification in dynamic systems

As pointed out in [1], understanding and comparing different publish/subscribe systems is quite a difficult task, due to informal and different semantics. From this, it 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 [6], where the author defines safety properties which are actually similar to the ones defined in Section 2.4. However defining "no bad thing can happen" is the easy part of the job in dynamic distributed systems (such as publish/subscribe 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. [6] defines liveness 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 PSS 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 the assumption on a minimum number of correct processes in the system (i.e., processes that never fail). If we make a parallel with a

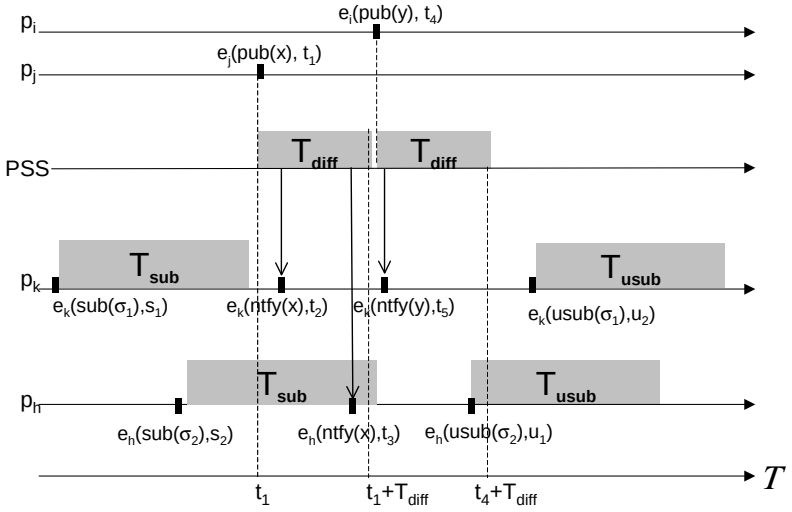


Figure 5. Global History with not expected notify events

PSS, this means to make an assumption on the minimum number of subscribers that never disconnects. It is clear that in a PSS such an assumption does not make any sense[4]. A disconnected process *is not a bad process* as "disconnection" is a matter of life in a PSS and not an undesirable event to cope with. A liveness specification for this dynamic context should capture this normal behavior.

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 PSS). This presence depends of three delays ($T_{diff}, T_{sub}, T_{usub}$) which abstract the dynamic behavior of the computation and the PSS implementation. Thanks to the latter point our Liveness condition can also be used to compare different PSS implementations. To explain this point, consider the following example. Suppose to have two different PSSs 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 satisfies liveness as defined in section [6], 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 [6]. However, differently from [6] when the three delays are finite (typical practical case) some subscription (satisfying conditions stated in the Liveness property) must be notified.

4 Defining a liveness that gives guarantees if and only if a process never disconnect is the equivalent of not giving any guarantee.

3. 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, *notification loss* is the probability that x is not notified to p (i.e., $1 - d$).

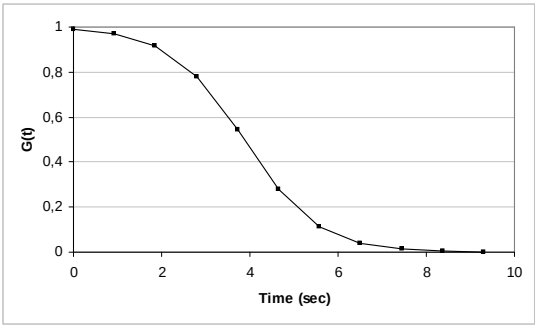
In this Section we first provide numerical results for T_{diff}, T_{sub} and d obtained through the simulation of a PSS in a specific network setting. Then, we present a simple and general analytical model for the computation of d.

3.1. PSS Simulation Study

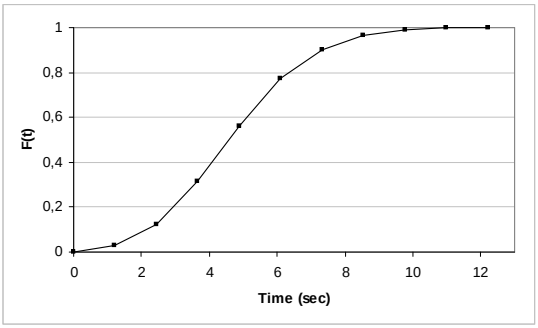
We carried out our experiments by implementing the prototype of a distributed PSS made up of a set of distributed brokers, communicating through point-to-point application-level connections. The system is based on the content-based routing algorithm (CBR) for acyclic peer-to-peer topologies introduced in Siena [3]. 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. Each broker has to maintain a routing table, that represents a local view of the global subscription distribution.

The CBR algorithm limits subscription propagation exploiting containment relationships among subscriptions. A subscription σ_1 is contained in another σ_2 if σ_2 matches all the notifications for σ_1. A subscription σ that is not present in the system is propagated by its subscriber throughout the entire network. If the same subscriber issues a subscription contained in σ, it does not need to route it again.

The CBR algorithm for event diffusion works as follows: each time an event e is received by a broker B_i either from



(a)



(b)

Figure 6. Experimental values of T_{diff} and T_{sub} with respect to $B(t)$

its local publishers or from a link, it matches e against all local subscriptions and then forwards e *only* through links which can lead to potential e's subscribers. Forwarding links are determined from the routing table, by matching the event against the entries contained in it.

Simulation Results Simulations were performed by running the PSS prototype on top the J-Sim [5] real-time network simulator, providing an accurate representation of the behavior of the entire network stack. Experiments featured 100 brokers and 100 network nodes. Network-level topologies are generated using the Georgia Tech ITM topology generator [10] and 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. The fact that the application-level topology is oblivious of the underlying network topology reproduces the common real-world situation. We stress the fact that this technique was never used in previous PSS simulation studies [3, 6], which did not include a representation of the network level.

Figures 6(a) and 6(b) respectively show the plots of the percentage $B(t)$ of brokers at time t reached by a new publication or by a new subscription issued at time 0. The time when *all* brokers receive the notification/subscription corresponds to

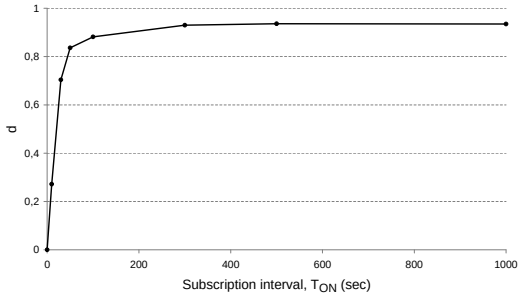


Figure 7. Experimental values of d

T_{diff}/T_{sub}, that in this case are roughly 8 seconds and 12 seconds, respectively.

Figure 7 shows the number of notifications actually obtained for events published during a subscriber's subscription interval. 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, an event is published in another random broker. The exact publication time is randomly chosen and follows a uniform distribution inside the subscription interval. We repeated this 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}, obtaining the curve depicted in Figure 7.

3.2. Analytical Model

The analytical model rests on the following assumptions:

1. the process p issues the subscription σ for a period $T_{ON} \geq T_{sub} + T_{diff}$;
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.

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 PSS's configuration after this time interval. Removing such hypothesis, however, would only increase the value d (i.e., the notification probability we calculate is a lower bound of the actual probability).

Let t_{sub} be the time p issues the sub operation, t_{usub} the time when p issues the corresponding unsubscription and t_{pub} the time the notification x is published. Then, the PSS guarantees the delivery of any publication occurring at a time t_{pub} such that $t_{sub} + T_{sub} \leq t_{pub} \leq t_{usub} - T_{diff}$. This means in fact that the publication was issued when the subscription

Figure 8. A sketch of $P(t)$, the probability that a pub issued in the interval T_{ON} is notified to an interested subscriber.

was stable and there was enough time for information diffusion to be completed before p issues the unsubscription. Moreover, for the publications such that $t_{sub} < t_{pub} < t_{sub} + T_{sub}$ as well as for those with $t_{usub} - T_{diff} < t_{pub} < t_{usub}$ there is also some probability for being notified.

For example, let us consider a distributed implementation of the PSS 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 $t < T_{sub}$, the portion of the PSS 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 $f(\tau)$ the probability density function that the PSS notifies x to p, given that x was issued at time $t_{pub} = t_{sub} + \tau$, where $0 \leq \tau \leq T_{sub}$. Clearly, $f(\tau)$ must be a monotonically increasing function with $f(0) = 0$[5] and $f(T_{sub}) = 1$. In our experiments, $f(\tau)$ corresponds to the function plotted in Figure 6(a).

Also, $g(\tau)$, where $0 \leq \tau \leq T_{diff}$, is the probability density function that a notification published at a time $t_{pub} = t_{usub} - T_{diff} + \tau$ 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 $g(T_{diff}) = 0$. In our experiments, $g(\tau)$ corresponds to the function plotted in Figure 6(b).

Figure 8 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 PSS.

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

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

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

that can also be rewritten as

$$d = \frac{T_{ON} - T_{diff} - T_{sub}}{T_{ON}} + \frac{1}{T_{ON}} \left(\int_0^{T_{sub}} f(t)\,\mathrm{d}t + \int_0^{T_{diff}} g(t)\,\mathrm{d}t \right) \quad (2)$$

Note that d is directly proportional to the area of the curve depicted in Figure 8. Clearly, if $T_{diff} = T_{sub} = 0$ then the PSS behaves as an ideal system with $d = 1$ (all publications are immediately notified).

3.3. Analytical Results

In order to provide some numerical results, the functions $f(\tau)$ and $g(\tau)$ have to be specified. It is expected that the actual shape of the curves reflects, respectively, the update mechanisms used internally by the PSS and the diffusion mechanism. For the sake of simplicity we will consider the following definitions:

$$f(\tau) = \left(\frac{\tau}{T_{sub}}\right)^{\frac{1}{r}} \quad (3)$$

$$g(\tau) = \left(\frac{T_{diff} - \tau}{T_{diff}}\right)^{\frac{1}{r}} \quad (4)$$

where considering $r > 0$. r is a parameter representing the rapidity of information propagation inside a PSS. If $r \to 0$ a piece of information issued at time t is seen by all brokers at time $t + T_{sub}$ (resp. $t + T_{diff}$). If $r \to \infty$, then a piece of information issued at time t is seen by all brokers at time t (ideal behavior).

Considering these simple expressions for $f(\tau)$ and $g(\tau)$ has the advantage of highly simplifying the analysis, at the same time not disproving the significance of the numerical results, since, as it is clear from the expression of d, the delivery probability depends only from the area defined by function $P(t)$, not by its shape. The above analytical expressions of $f(\tau)$ and $g(\tau)$ are thus equivalent to the actual functions if they provide the same area: for a real application of our model it is sufficient to calculate the value of r that provides the same area.

According to the above assumptions:

$$\int_0^{T_{sub}} f(\tau)\,d\tau = \frac{1}{T_{sub}^{\frac{1}{r}}} \int_0^{T_{sub}} \tau^{\frac{1}{r}}\,d\tau = \frac{r}{r+1} T_{sub}$$

Similarly,

$$\int_0^{T_{sub}} g(\tau)\,d\tau = \frac{r}{r+1} T_{diff}$$

Hence,

$$d = \frac{T_{ON} - T_{diff} - T_{sub}}{T_{ON}} + \frac{r}{r+1} \frac{T_{sub} + T_{diff}}{T_{ON}}$$

5 Actually, the value is slightly higher than 0, because there is the probability that x is issued by p itself.

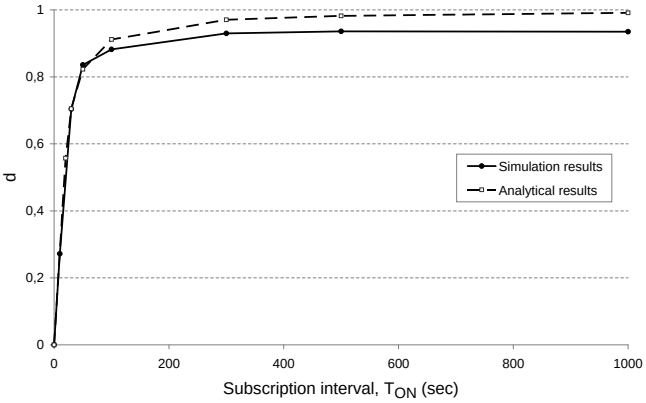


Figure 9. Comparing simulation and analysis

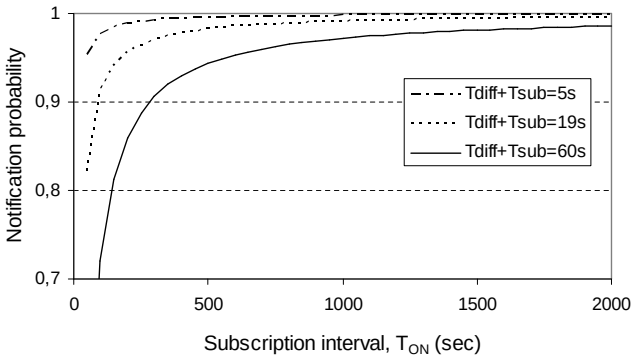


Figure 10. Notification probability as a function of the subscription interval T_{ON}, for r=1.14 and different values of $T_{sub} + T_{diff}$

that can be rewritten as

$$d = 1 - \frac{1}{r+1}\frac{T_{diff} + T_{sub}}{T_{ON}}$$

Note that, for $T_{ON} = \infty$, or $\Delta_{diff} + \Delta_{sub} = 0$, or $r = \infty$, $d = 1$ (Recall that we have assumed $T_{diff} + T_{sub} \leq T_{ON}$).

Figure 9 shows a comparison between the curve obtained from the experiment and the one calculated analytically by applying Equation 3. The values of T_{sub} and T_{diff} are 8 and 12 seconds (obtained from Figures 6(b) and 6(a)). The value of r used is 1.14. This is the value that inserted in the analytical expression of $f(\tau)$ and $g(\tau)$ in Equations 1 and 2 provides a function which returns the same value of the integral of the functions that interpolate the plots shown in Figures 6(b) and 6(a) respectively. The results of the comparison clearly show the similarity between the experimental and the analytical curves, especially for low values of T_{ON}, where the two curves completely overlap. For growing values of T_{ON} differences between the two plots are within 5%.

Let us finally remark that a generic PSS system is characterized by the two values of T_{sub} and T_{diff} that give a bound on the time required to propagate subscriptions and notifications, and by the parameter r, which summarizes the way information is propagated inside the network. As an example of application of the model, Figure 10 reports how the notification probability varies as a function of time for different values of $T_{sub} + T_{diff}$ and r = 1.14. The curve can be used to predict how the probability is affected by the two delays.

4. Conclusions

In this paper we presented a formal framework for the general specification of the delivery guarantees offered by a publish/subscribe system. With respect to other contributions [6], our framework gives timing conditions under which a notify event is generated. The timing conditions change according to the PSS implementation.

The framework has been used to devise a simple analytical model of a PSS (the first to the best of our knowledge). Results of such a model can be used as rule of thumb for users and designers of a PSS to predict the behavior of their applications with respect to the notification loss due to concurrent and independent execution of publishers and subscribers and to the non-instantaneous delivery of a notification. The results obtained from a deep, time-consuming simulation analysis, which integrates two complex public domain simulation tools (J-Sim and GT-ITM) plus a home-brewed PSS prototype, confirm those obtained through our analytical model.

As far as future work is concerned, we are planning to complicate the model along two directions. Firstly, we want to analyze the relation between r, T_{diff} and T_{sub}. Secondly, we want to study the effect of the persistence of a publication (i.e., a publication is stored in the system for a given lifetime) within the PSS both on the formal framework and on the analytical model. We are particularly interested in capturing the relation between the persistence and the notification loss.

References

[1] R. Baldoni, M. Contenti, and A. Virgillito. The Evolution of Publish/Subscribe Systems. In André Schiper and Alexander A. Shvartsman and Hakim Weatherspoon and Ben Y. Zhao, editor, *Future Trends in Distributed Computing, Research and Position Papers*, volume 2584. Springer, 2003.

[2] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. Strom, and D. Sturman. An Efficient Multicast Protocol for Content-based Publish-Subscribe Systems. In *Proceedings of International Conference on Distributed Computing Systems*, 1999.

[3] A. Carzaniga, D. Rosenblum, and A. Wolf. Design and Evaluation of a Wide-Area Notification Service. *ACM Transactions on Computer Systems*, 3(19):332–383, Aug 2001.

[4] G. Cugola, E. D. Nitto, and A. Fuggetta. The JEDI Event-Based Infrastructure and its Applications to the Development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27(9):827–850, September 1998.

[5] J-Sim. http://www.j-sim.org, 2003.

[6] G. Muhl. *Large-Scale Content-Based Publish/Subscribe Systems*. Phd thesis, Technical University of Darmstadt, 2002.

[7] B. Oki, M. Pfluegel, A. Siegel, and D. Skeen. The Information Bus - An Architecture for Extensive Distributed Systems. In *Proceedings of the 1993 ACM Symposium on Operating Systems Principles*, December 1993.

[8] R. Preotiuc-Pietro, J. Pereira, F. Llirbat, F. Fabret, K. Ross, and D. Shasha. Publish/Subscribe on the Web at Extreme Speed. In *Proc. of ACM SIGMOD Conf. on Management of Data*, Cairo, Egypt, 2000.

[9] B. Segall and D. Arnold. Elvin Has Left the Building: A Publish /Subscribe Notification Service with Quenching. In *Proc. of the 1997 Australian UNIX and Open Systems Users Group Conference*, 1997.

[10] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to model an internetwork. In *IEEE Infocom*, volume 2, pages 594–602, 1996.