Providing Probabilistic Latency Bounds for Dynamic Publish/Subscribe Systems

M. Adnan Tariq, Boris Koldehofe, Gerald G. Koch, and Kurt Rothermel

IPVS – Distributed Systems, Universität Stuttgart
{firstname.lastname}@ipvs.uni-stuttgart.de

Abstract. In the context of large decentralized many-to-many communication systems it is impractical to provide realistic and hard bounds for certain QoS metrics including latency bounds. Nevertheless, many applications can yield better performance if such bounds hold with a given probability. In this paper we show how probabilistic latency bounds can be applied in the context of publish/subscribe. We present an algorithm for maintaining individual probabilistic latency bounds in a highly dynamic environment for a large number of subscribers. The algorithm consists of an adaptive dissemination algorithm as well as a cluster partitioning scheme. Together they ensure i) adaptation to the individual latency requirements of subscribers under dynamically changing system properties, and ii) scalability by determining appropriate clusters according to available publishers in the system.

1 Introduction

Publish/subscribe is a well-known and popular communication paradigm for building distributed applications such as stock exchange, traffic monitoring or person tracking. It provides a decoupling of producers of information, called publishers, from consumers of information, called subscribers. Without knowledge of the actual source of information, subscribers specify their interests in the form of subscriptions and are notified about the corresponding published events.

In the past, most research has focused on providing expressive and scalable publish/subscribe systems. However, innovative Internet applications such as distributed online games have other requirements like time bounded processing and delivery of events. Moreover, in business applications, complex event processing [13] may be used to perform the transition from basic events, e.g. sensor readings, to complex events that match the semantics of the application, e.g. the detection of a fire. If such transitions need to be performed in a timely manner, delayed events may not only be useless, but also lead to wrong computations by the application.

Reserving resources along the communication links can guarantee end-to-end QoS between subscribers and publishers [5]. However, this is not always viable since reservation protocols are typically not available on a global scale and reservation in the context of heterogeneous network environments is even harder.

An alternative approach is to observe the behavior of the underlay and then provide a QoS bound with a probabilistic reliability derived from the observations. This reliability depends on the value chosen for the QoS bound and its probability distribution.

Klaus David and Kurt Geihs (Eds.): Kommunikation in Verteilten Systemen (KiVS), LNCS 1431-472X, pp. 155–166, 2009.

The original publication is available at www.springerlink.com: http://www.springerlink.com/content/978-3-540-92666-5

Although the characteristics of the underlay may change dynamically over time and so affect the probabilistic reliability, providing probabilistic bounds is still useful in three ways. First, the end-to-end latency distribution can be fairly stable, e.g. in the Internet it can be modeled with a certain probability distribution with small variation [8,17]. Second, probabilistic reliability allows the system to overstep a given QoS bound to a determined extent while not violating the agreement with the user. Third, even a small reduction of the demanded probabilistic reliability yields a better probabilistic bound and, with this, a significant performance improvement. For instance, assuming that the probability distribution of message latency is stable, one can derive that an event will be propagated within 200*ms* with a probability of 95%, while a more useful latency bound of 100*ms* can be achieved with a probability of 90%.

In this paper we address how end-to-end latency requirements of individual subscribers in a content-based publish/subscribe context can be satisfied by accounting for the latency distribution of communication links. We propose an algorithm that adapts to the requirements of subscribers and maximizes the set of subscribers and publishers it can support. First, we introduce an event dissemination algorithm in a restricted content-based model with constraints on the set of publishers that adapts to dynamic changes in the underlay (cf. Section 3). Then we show how to generalize the algorithm to match the content-based publish/subscribe model by providing a clustering scheme (cf. Section 4). We also present an experimental evaluation (cf . Section 5) of the algorithm performance under dynamic behavior.

2 Probabilistic latency bounds in content-based publish/subscribe

We consider the content-based publish/subscribe model of communication in a distributed system consisting of an unbounded and dynamic set of peers. We assume that peers can leave or join arbitrarily often and they can fail temporarily or permanently. Furthermore, we assume that the underlying communication environment is heterogeneous with different link properties related to delay and bandwidth; however each peer is connected to the network through a single interface. Each peer is assumed to have a unique identity, which can be used to establish a logical point-to-point connection, forming an overlay network. Logical links are created and maintained by a probabilistic membership service such as [16], which helps peers to bootstrap and prevents the partitioning of the network. On top of the membership service, peers are organized in a single spanning tree, where the links of the spanning tree are embedded within the links of the membership service overlay.

Peers contribute in the publish/subscribe system in one of two roles: publishers or subscribers. Publishers serve as information sources and the publish/subscribe overlay needs to ensure that subscribers receive all the relevant messages. According to the content-based publish/subscribe model, an event consists of a set of attributes and associated values. We use Ω to denote the set of all attributes that exist in the system. Each publisher p periodically propagates an advertisement $Adv(p) \subseteq \Omega$. Thereby p announces potential future events with any desired attribute set taken from the set $PS^{-\emptyset}_{Adv}(p) := PS_{Adv}(p) - \{\emptyset\}$, with $PS_{Adv}(p)$ being the power set of $Adv(p)$. For example, a publisher p with $Adv(p) = \{temperature, humidity\}$ can publish events consisting

of $\{temperature\}$, $\{humidity\}$ or $\{temperature, humidity\}$. Subscribers issue subscriptions which are expressed as a conjunction over ranges of attribute values, e.g. ($color = red \wedge temperature \in [20,25]$). Beyond typical content-based publish/subscribe systems, subscribers can specify upper latency bounds with their subscriptions, together with a minimum probability that these bounds are met.

Peers observe two QoS metrics: traffic (in terms of bandwidth) and latency. Each peer maintains a traffic specification of its network interface, which specifies the number of event messages that can be propagated per time unit. Traffic specifications place constraints on the creation of overlay links. Let $T_S(n)$ and $T_C(n)$ denote the traffic specification and the current traffic characteristics of peer n, respectively, and let $r(n,s)$ specify the message rate for any new connection to peer s. Then, peer n will accept the new connection only if $T_C(n) + r(n,s) \leq T_S(n)$ is satisfied. Latency is modeled probabilistically in our system: each peer maintains information about the latency distribution for each of its overlay links, assuming that all messages are of similar size. In practice, information about the latency distribution and traffic characteristics can be collected by relying on cross-layer sampling services [15].

In addition to latency characteristics of overlay links to neighbors, each subscriber peer also maintains information about the latency distribution of end-to-end paths to distant peers, which we call *path latency characteristics* (*plc*). Subscribers maintain their *plc* recursively: after receiving the *plc* of a direct predecessor n in the spanning tree, a subscriber s combines it with latency characteristics of the overlay link between n and s, and notifies its successors on the tree about its updated *plc*.

A subscriber can use the path latency characteristics to determine the validity of its *latency specification* (i.e. probabilistic upper latency bound) for an end-to-end path to a publisher, like the one shown in *Figure* 1. If the latency characteristics fail to match the individual latency specification, the subscriber has to take steps to increase the *plc*. Our approach fulfills the individual latency specifications of subscribers by placing them at an appropriate position with respect to relevant publishers in the overlay network.

Satisfaction of latency specifications in content-based publish/subscribe in a scalable manner is highly challenging. First, providing an optimal solution to satisfy the QoS demands of all subscribers with the same interest in the presence of traffic constraints is NP hard and is known as "delay-constraint minimum cost routing" problem [14]. Second, in publish/subscribe systems, the selectivity of subscriptions (i.e. the ratio of the total number of events that match the subscription) can vary widely. Less selective subscribers should obviously be placed before more selective ones, but at the same time peers with low traffic specifications should be behind subscribers with high specifications. A trade-off is necessary if subscribers with high traffic specifications are highly selective or subscribers with low specifications are less selective. Third, one must also consider that multiple publishers' trees may be embedded in the same overlay network and hence a subscriber may need to consider placement with respect to many publishers.

In this paper we approach these challenges by looking first at the constraints on subscriber selectivity by extending Siena's content-based routing [6]. We propose adaptation mechanisms that maximize the number of satisfied subscribers. In a first step, we impose constraints on the advertisements of publishers in order to reduce the multiple-

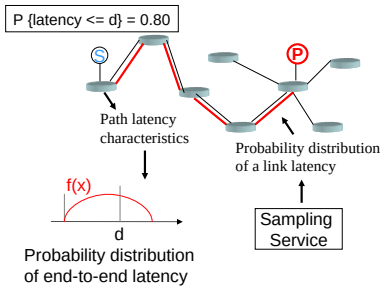

Fig. 1. QoS Model

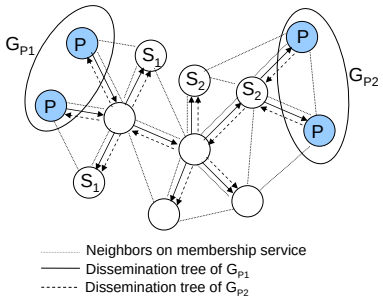

Fig. 2. Spanning trees created on top of membership service with embedded dissemination trees

publisher problem. Finally, we release these constraints and propose an approach to group publishers in clusters according to their advertisements and this way tackle the multiple-publisher problem.

3 Adaptation to probabilistic latency requirements

In a first approach to adapt publish/subscribes system to probabilistic latency specifications we introduce the constraint that publishers have either completely overlapping or disjoint advertisements, i.e., the set of publishers can be partitioned into disjoint groups of publishers $G_P = \{p_1, p_2, \ldots, p_m\}$ so that $\forall p_i, p_j \in G_P : Adv(p_i) \subseteq Adv(p_j) \vee Adv(p_j) \subseteq Adv(p_i)$. Hence, publications that match a subscription s are issued by a single group of publishers further denoted as $G_P(s)$. We write $s \prec G_P(s)$ to denote that s can be satisfied by $G_P(s)$. We assume that each subscriber has only one subscription and hence will not differentiate between the terms subscriber and subscription.

We now present an adaptive algorithm that maximizes the number of subscribers whose latency specifications can be satisfied without violating their traffic constraints. The algorithm, on top of the spanning tree, maintains a separate dissemination tree for each group of publishers G_p (cf. Figure 2). To construct the dissemination tree and to match events with subscriptions, we extend Siena [6] approach to take into account the *path latency characteristics* (*plc*) for a publisher. A dissemination tree is created by a publisher flooding its advertisement along the spanning tree. Each advertisement includes the *plc* with respect to the publisher. A peer that receives an advertisement sets up the path for the dissemination tree, updates the *plc* and forwards the advertisement along with the updated *plc* down the dissemination tree.

In the following we show how to adapt the dissemination trees by relying on two strategies. The *reactive adaption* uses local adaptations, to find an appropriate position for each subscriber in its relevant dissemination tree such that its individual latency specification is met. The *proactive adaptation* periodically runs an optimization algorithm and tries to enhance the dissemination trees, so that a large number of subscribers can be satisfied.

Algorithm 1 Placement Strategies

Require: A subscriber s whose latency specification $l(s)$ should be satisfied
Ensure: Peer n which can satisfy the latency specification of s.

```
 1: noOfTries ← 0 // Number of tries to find a suitable position
 2: childForInsertion [ ] ← ∅
 3: for all n ∈ N(s) do // N(s) = {n ∈ Neighbors | (s,n) ∉ SpanningTree}
 4:     if (T_S(n) − T_C(n) > r(n,s)) ∧ (l(s) = satisfied ) then
 5:         create link (n,s) on spanning tree and remove existing link.
 6:         break
 7: for all z_i ∈ {z_m, z_{m−1}, ... z_0 | z_m = Grandparent(s) ∧ ∀_i z_{i−1} = parent(z_i) ∧ z_0 ∈ G_P}  do
 8:     if l(s) = satisfied then
 9:         if T_S(z_i) − T_C(z_i) > r(z_i,s) then
10:             connect to z_i
11:         else // Try to insert s between z_i and one of its children
12:             for all k_i ∈ K  do // K = {k_i | k_i ∈ child(z_i)}
13:                 if ¬(k_i ≺ G_p(s)) ∧ ¬(subtree(k_i) ≺ G_p(s)) then
14:                     childForInsertion ← k_i // insertion is always possible
15:                 else if k_i ≺ G_p(s) ∧ ¬(subtree(k_i) ≺ G_p(s)) then
16:                     if subscription of k_i is not violated then childForInsertion ← k_i
17:                 else // subtree(k_i) ≺ G_p(s)
18:                     childForInsertion ← k_i (insert with probability)
19:             if |childForInsertion| > 0  then // more than one child can be used for insertion
20:                 Select child with more selective subscription and low traffic specs (T_S(k_i))
21:                 add links (z_i,s) and (s,k_i) and remove (z_i,k_i)
22:                 If T_C(s) + r(s,k_i) > T_S(s) then s will leave one of its children to the existing parent
23:                 break
24: if l(s) ≠ satisfied then
25:     noOfTries++
26:     backoff for noOfTries ∗ΔT seconds
27:     Start from random location
```

3.1 Reactive and proactive adaptations

A new subscriber s arriving in the system, or an existing subscriber (because of dynamic behavior such as the crash or departure of a parent node, or changes in the *plc*), triggers the reactive algorithm to find a new position within the dissemination tree of $G_P(s)$. The reactive algorithm uses the following placement strategies (cf. *Algorithm* 1):

Local transformation (*lines 3-6*): If changes to the underlying QoS violate the latency specification, peers first try to connect to another parent among their neighbors of the member service overlay.

Bottom-up strategy (*lines 7-22*): If the local transformation is unsuccessful, then the subscriber needs to connect to a suitable parent higher in the dissemination tree, i.e. closer to the publishers. The subscriber s follows the reverse path formed by the advertisements. Once a suitable parent z_i is found, s will connect directly, if the traffic specification of z_i allows. Otherwise, s tries to be inserted between z_i and one of its child peers k_i.

If the peers in the subtree of k_i are subscribed for $G_p(s)$ (*lines 17-18*), then s is inserted probabilistically. The reason is that the number and latency specification of the subscribers are not locally available, so the changes in the link behavior may trigger a lot of simultaneous adaptations, which may clutter the network. The probability of insertion decreases with the number of unsuccessful tries performed by s to find an

Algorithm 2 Proactive Algorithm

Require: Peer n performing proactive algorithm.

1: childToPromote [] $\leftarrow \emptyset$
2: $V = \{G_{p1}, G_{p2} \cdots G_{pm} | \ \forall_k \ k \in child(n) \wedge \ k \ is \ subscriber \Rightarrow G_p(k) \in V\}$
3: **for all** $v_i \in V$ **do**
4: **for all** $k_i \in K$ **do** // $K = \{k_i \in child(n) \wedge \ k_i \prec v_i\}$
5: $W(k_i)$ = Assign weight according to selectivity of subscription and traffic specs $T_S(k_i)$
6: **if** $\forall_{k_j \in K, j<i} \ W(k_i) > W(k_j)$ **then**
7: childToPromote[v_i] = k_i
8: **for all** $k_i \in$ childToPromote **do**
9: promoteToParent(k_i,z_i) // Performs bottom-up strategy (*Algorithm* 1) to connect to parent

appropriate position and with the level of node k_i in the dissemination tree (details are in Section 3.2).

Random Connect (*lines 23-26*): If the latency specification of s cannot be satisfied after reaching one of the publishers, it will perform an exponential backoff and tries to find its position by connecting to a random peer. However, s leaves the existing parent only when a suitable peer is found that can satisfy its latency specifications.

The *proactive algorithm* runs periodically and pushes the subscribers with good traffic specifications and less selective subscriptions near the publishers. The goal is to improve the overall quality of the dissemination tree so that more subscribers with their individual latency specifications can be satisfied. The algorithm is performed by every non-leaf peer for the immediate child subscribers on each dissemination tree (cf. *Algorithm* 2).

3.2 Algorithm properties

The algorithm design addresses two issues: compliance to individual latency specifications and scalability.

Clearly, latency specifications can only be fulfilled if there exists a suitable position for the peer in the overlay. In cases where an individual subscription can be fulfilled by a large number of neighbors there is a high chance to find an appropriate position by performing the local adaptation strategy. If such neighbors are not at hand, the bottom-up strategy is of benefit, since peers closer to the publishers can satisfy higher requirements. Nevertheless, it is possible that a subscriber will not be able to satisfy its latency specification. In this case the random selection ensures that it will eventually find a suitable position.

Scalability is related to the number of satisfied subscribers, the overhead of message forwarding and finally the cost of performing adaptations. Latency specifications of a large number of subscribers can be satisfied because the proactive maintenance yields fat trees, where subscribers with good traffic specifications are pushed close to the publishers. Furthermore, the notification forwarding overhead is reduced by the placing subscribers near their relevant publishers, saving intermediate peers from forwarding irrelevant notifications. The subscription forwarding overhead is reduced by placing less selective subscriptions close to the relevant publishers.

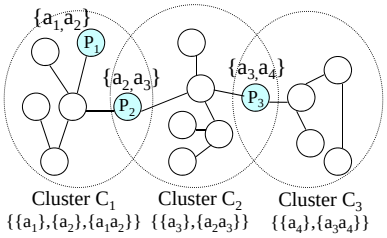

Fig. 3. Cluster management for a system with four attributes $\Omega = \{a_1, a_2, a_3, a_4\}$ and three publishers. $PS^{-\emptyset}_{Adv}(p_2) = \{\{a_2\}, \{a_3\}, \{a_2, a_3\}\}$, so p_2 publishes notifications with attribute $\{a_2\}$ in C_1 and with attributes $\{\{a_3\}, \{a_2, a_3\}\}$ in C_2.

Another important aspect is the cost of performing adaptations. The application of a bottom-up strategy and local transformation reduce the management overhead of the peers higher in the dissemination tree. Furthermore, the algorithm reduces the number of concurrent adaptations during failures: a subscriber s whose latency specification is violated first tries to find a position satisfying also all the relevant subscribers in its subtree. A probabilistic approach is used, which balances two factors—the position of the subscriber in the dissemination tree and the number of unsuccessful attempts and is given by $e^{-level(s)*noOfTries/(Normalizationparameter)}$. The higher the subscriber is located in the tree, the higher is the probability to move to a position that satisfies all peers in its subtree and prevents the start of many simultaneous adaptation algorithms. However, simulation results have shown that due to saturation near the publishers the subscriber higher in the tree may not be successful, so the number of tries should be limited.

4 Cluster management

In this section, we extend the adaptive dissemination algorithm of Section 3 to match the requirements of a generic publish/subscribe system without constraints on publishers. This implies that the advertisements of different publishers overlap, and a subscription can be satisfied by more than one publisher's group (the multi-publisher problem). In the following we present how clustering of publishers leads to an appropriate number of groups with which publishers as well as subscribers need to be associated.

Let C be a cluster of publishers and let CS_C denote the set of all attribute sets that are published in the cluster. CS_C determines the potential subscribers of the cluster. In *Figure* 3, for example, cluster C_1 has $CS_{C_1} = \{\{a_1\}, \{a_2\}, \{a_1, a_2\}\}$, and subscribers with subscriptions consisting of either of these attribute sets will join C_1.

The basic idea is to prevent the presence of these attribute sets in additional clusters, so that subscribers will not have to be associated with multiple clusters and thus preserve scalability for subscribers. Therefore, the cluster management algorithm has to consider the intersections of $PS^{-\emptyset}_{Adv}$ of publishers rather than $Adv(p)$. Let $M := \{C_1, C_2, ...\}$ be the set of all clusters currently in the system. The cluster management algorithm preserves the invariant that the attribute sets of all clusters $C \in M$ are disjoint, i.e. $\forall C_i, C_j \in M : i \neq j \Rightarrow CS_{C_i} \cap CS_{C_j} = \emptyset$. When a publisher p arrives in the system, the overlap between its $PS^{-\emptyset}_{Adv}(p)$ and all CS_{C_i} in M is calculated. The following cases are distinguished:

No overlap with any CS_C: The publisher creates a new cluster C_{new} with $CS_{C_{new}} = PS^{-\emptyset}_{Adv}(p)$ and advertises the cluster.

$PS^{-\emptyset}_{Adv}(p)$ is included in an existing cluster: The publisher merges in the corresponding cluster.

Existing clusters are included in $PS^{-\emptyset}_{Adv}(p)$: The publisher creates a new cluster C_{new} with $CS_{C_{new}} = PS^{-\emptyset}_{Adv}(p)$ and advertises the cluster; all clusters whose CS_C is a subset of $PS^{-\emptyset}_{Adv}(p)$ merge in the new cluster.

Partial overlap: The publisher joins every cluster whose CS_C overlaps with $PS^{-\emptyset}_{Adv}(p)$, and for the remaining elements of $PS^{-\emptyset}_{Adv}$ it creates a new cluster $CS_{C_{new}} = PS^{-\emptyset}_{Adv}(p) - \bigcup_i CS_{C_i}$. Joining a cluster C means that p publishes events with an attribute set from the intersection $CS_C \cap PS^{-\emptyset}_{Adv}(p)$ exclusively in cluster C. The publisher with the most general advertisement (i.e. that has the highest share in the CS_C of the cluster) becomes the cluster head.

With the resulting clustering, a subscription can always be satisfied by publishers of just one single cluster, and we can directly apply the adaptive dissemination algorithm of *Section 3* to each cluster. However, if each publishers p would select the attributes in $Adv(p)$ uniformly at random from Ω, a large set of publishers can create many disjoint clusters, each cluster only serving a small fraction of the involved publishers' $PS^{-\emptyset}_{Adv}(p)$. This is beneficial for subscribers because clusters publish only a small variety of events and subscribers receive only a small amount of false positives (i.e. events that do not match their individual subscription). Publishers, however, might have to be associated with a high number of small clusters.

In order to preserve scalability of the system, i.e. to avoid that publishers would need to connect to a number of existing clusters that will grow exponentially with $|Adv(p)|$, we use an inherent property of the cluster management algorithm: It merges a pair of clusters C_1 and C_2, if CS_{C_1} is included in CS_{C_2} or vice versa. If a publisher p realizes that it has to connect to a large number of clusters, it generalizes its advertisement by adding an appropriate attribute to $Adv(p)$ so that another cluster can merge in the newly created cluster of p. This way, the arrival of a new publisher does not result in a growth of $|M|$ in the system. Of course, the effect of this solution has to be balanced with the subscribers' interest that clusters have a small CS_C in order to limit the number of false positives. This can be achieved with a threshold for the number of clusters that p has to connect, before the solution is applied, and by changing the number of attributes that p is allowed to assume in order to generalize its $Adv(p)$. For further scalability, cluster heads do not advertise CS_C which is often a large proper subset of $PS^{-\emptyset}_{Adv}(p)$. Instead, the cluster head advertises its $Adv(p)$ together with the sets that are in $PS^{-\emptyset}_{Adv}(p)$ but excluded from CS_C. The exclusion of a set amounts to the exclusion of its power set so that the exclusion information in an advertisement is kept rather small, and the algorithm's tendency to merge clusters also prevents the exclusion of too many subsets.

5 Evaluation

This section evaluates the self-adaptation algorithm with respect to convergence and stability in the presence of failures and churn, using metrics similar to [2]. The evaluations were performed using PeerSim [12], a large-scale P2P simulator.

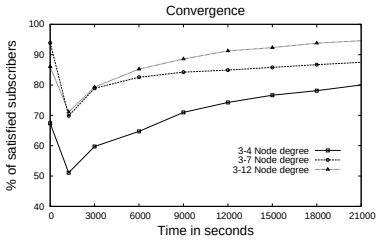

Fig. 4. Convergence

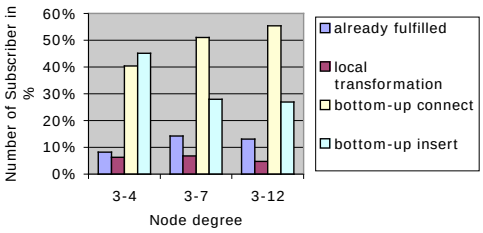

Fig. 5. Adaptations

No routing protocol was implemented at the underlay level. Instead, latencies between the routers were assigned based on the King [10] methodology, which estimates the latency between any pair of Internet hosts by measuring the latency between nearby authoritative DNS servers. The latencies between the routers are in the range $[1, 500]$ms with reliability factor of 90% to 100%.

All the simulations are performed for $1,024$ peers. The number of neighbors on the underlying membership service is assigned randomly between 10 to 15. The initial delay for the *Random Connect* strategy is 3 seconds and the delay for *proactive algorithm* is 5 seconds. The latency requirements from the subscribers are in the range [120, 260]ms with low probabilities. For simplicity, the traffic characteristics of a peer is simulated by the node degree.

5.1 Algorithm convergence

We analyze the algorithm's behavior to converge in a limited number of adaptations under static conditions. Convergence means that if the number of subscribers with individual latency requirements remains constant, the system will eventually converge to a stable state where a considerably large percentage of the subscribers is satisfied and no more adaptations are performed. We evaluated the algorithm with different traffic specifications of peers, i.e with possible node degrees in $[3,4]$, $[3,7]$ and $[3,12]$, with all other parameters left unchanged. Starting with 10 publishers with disjoint advertisements, a new peer chosen uniformly at random (among the 1024 peers), subscribed with its individual latency requirements at each simulation step until all $1,024$ peers in the system were either subscribers or publishers. Figure 4 shows the convergence behavior of the algorithm with respect to traffic specifications of peers. The lower percentage of satisfied subscribers in the first 1,024 simulation steps is due to the fact that new subscribers are constantly arriving in the system and are not immediately satisfied. Figure 4 also shows the effect of traffic specifications on the satisfaction of subscribers: as peers with high traffic specifications are pushed up by the proactive algorithm, the dissemination tree becomes fat and hence more subscribers can be satisfied. Figure 5 shows the percentage of subscribers (out of the total number of satisfied subscribers) satisfied by each adaptation strategy. Only $8-14$% of the subscribers are satisfied without any adaptation, which shows the effectiveness of the algorithm in increasing the number of satisfied subscribers. Similarly, low traffic specifications result in more insertions during the bottom-up strategy as visible in the case of a node degree $[3,4]$.

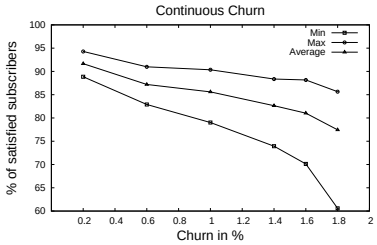

Fig. 6. Continuous churn

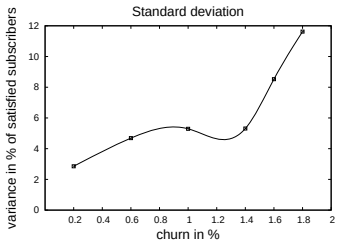

Fig. 7. Standard deviation for continuous churn

5.2 Algorithm stability

We have evaluated the algorithm's reaction to the dynamic changes in the system state and its convergence to a stable state.

Continuous churn: Simulations were performed with peers having a node degree in $[3, 7]$. In each simulation, 512 subscribers were pushed into the system and the system was allowed to converge to a stable state. Afterwards, continuous churn was introduced in the system. The percentage of churn was relative to the total of all peers in the system, e.g. a churn of 1.8% means that at every simulation step, 9 peers subscribed to and 9 peers unsubscribed from the system. Figure 6 shows the minimum, average and maximum percentage of satisfied subscribers in the system with increasing churn. Figure 7 shows the corresponding standard deviation, which gives an indication about the stability of the system. It is evident from the figures, that at higher churn rate the system is more unstable, because a higher number of potential subscribers are looking for a position. The reason being that the subscribers who canceled their subscriptions are still in the system occupying their previous positions, which makes it difficult for the new subscribers to find the position.

Massive churn: This scenario evaluates the behavior of the algorithm in the case of sudden rapid churn. The simulation settings were the same as in above experiments. Figure 8 shows that there is no relation between the percentage of churn and the course of the corresponding curve. For example, a curve with higher rapid churn might converge faster than others. The reason is that the algorithm relies on random interaction between peers and hence may converge to a different state on different runs. However, the system can tolerate and recover from massive occurrence of churn gracefully.

Further measurements on continuous and massive failures shows similar results. These results are omitted due to space constraints.

6 Related work

Over the last decade many content-based publish/subscribe systems [6,9,11,3] have evolved. Most systems focus on increased scalability of the system by reducing the cost of subscription forwarding and event matching. Clustering has also been identified as a technique to achieve scalability [4,1,7]. Clustering of subscribers can be achieved in two ways, either based on similarity of subscriptions or by partitioning the event

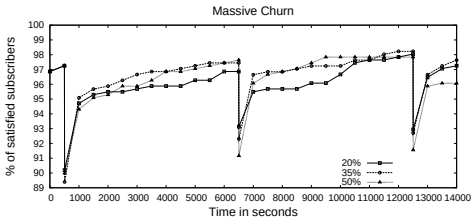

Fig. 8. Massive churn

space. Kyra [4] partitions the event space into clusters, brokers are assigned to each cluster and subscriptions are moved to relevant brokers. However, their approach is not dynamic, and when a broker joins or leaves, the whole partitioning needs to be recomputed. Additionally, no event partitioning criteria is specified. In [7], a direct mapping of the containment graph of subscriptions is mapped to a tree structure. This results in as many trees as there are subscriptions that are not contained in any other subscription. Similarly, [1] builds a tree for every attribute of the event space, and a subscriber can join any tree for which it has specified an attribute filter. Publishers publish to each tree with a matching attribute, which results in a huge number of unnecessary messages on each tree. Apart from the stated drawbacks, none of the approaches address issues related to QoS fulfillment.

Although it is easy to add QoS semantics into subscriptions [5], only few systems actually cope with satisfying QoS specifications in a decoupled environment like publish/subscribe, but rather deal with QoS as a metric for performance comparisons. To the best of our knowledge only two systems address issues related to satisfying individual latency requirements of subscribers. Indiqos [5] addresses delay requirements of individual subscribers, but relies on network reservation protocols, which limits its scalability. In [17] bounded delays on event delivery by employing message scheduling strategies at each broker are considered. They use a QoS model similar to ours. However, a static broker topology is assumed.

7 Conclusion

In this paper we have shown, in the context of pub/sub, how to deal with probabilistic latency bounds in large dynamic network environments. In particular, we apply subscription-centered adaptation to ensure an appropriate arrangement of subscriptions with various selectivity and to maintain high-capacity dissemination trees. Additionally, scalability is ensured by a publication centric clustering of the overlay. The evaluation shows that the algorithm performs well with respect to the fulfillment of individual latency specifications and is robust in a very dynamic setting. The described algorithm is currently practically applied to support a gaming application, in SpoVNet [15] project.

In the future, we will extend our partitioning schemes to reducing the management overhead due to advertisements. Moreover, we plan to investigate the impact of further QoS metrics and possible synergies with the current adaptation schemes.

8 Acknowledgment

This work is partially funded by Landesstiftung Baden-Württemberg under the initiative BW-FIT. Furthermore, we would like to thank Manuela Antonovic for her contributions towards the initial ideas of this paper.

References

1. E. Anceaume, A. K. Datta, M. Gradinariu, G. Simon, and A. Virgillito. Dps: Self-* dynamic reliable content-based publish/subscribe system. Technical report, IRISA, France, 2004.
2. Roberto Baldoni, Roberto Beraldi, Leonardo Querzoni, and Antonino Virgillito. Efficient publish/subscribe through a self-organizing broker overlay and its application to SIENA. *The Computer Journal*, 50:444–459, 2007.
3. Jorge A. Briones, Boris Koldehofe, and Kurt Rothermel. SPINE: Publish/subscribe for Wireless Mesh Networks through self-managed intersecting paths. In *International Conference on Innovative Internet Community Systems*. IEEE Computer Society, June 2008.
4. Fengyun Cao and Jaswinder Pal Singh. Efficient event routing in content-based publish-subscribe service networks. In *INFOCOM*, 2004.
5. Nuno Carvalho, Filipe Araujo, and Luis Rodrigues. Scalable QoS-based event routing in publish-subscribe systems. In *Proceedings of the Fourth IEEE International Symposium on Network Computing and Applications*. IEEE Computer Society, 2005.
6. Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 2001.
7. Raphaël Chand and Pascal Felber. Semantic peer-to-peer overlays for publish/subscribe networks. In *Euro-Par*, pages 1194–1204, 2005.
8. A Corlett, D.I. Pullin, and S. Sargood. Statistics of one-way internet packet delays. Presentation at 53rd IETF, March 2002.
9. Ludger Fiege, Mariano Cilia, Gero Mühl, and Alejandro Buchmann. Publish-subscribe grows up: Support for management, visibility control, and heterogeneity. *IEEE Internet Computing*, 10:48–55, 2006.
10. Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: estimating latency between arbitrary internet end hosts. 2002.
11. Abhishek Gupta, Ozgur D. Sahin, Divyakant Agrawal, and Amr El Abbadi. Meghdoot: Content-based publish/subscribe over p2p networks. In *Intl. Middleware Conference*, 2004.
12. Márk Jelasity, Alberto Montresor, Gian Paolo Jesi, and Spyros Voulgaris. Peersim: A peer-to-peer simulator. http://peersim.sourceforge.net/.
13. David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., 2001.
14. Ariel Orda and Er Sprintson. QoS Routing: the precomputation perspective. In *Infocom*, 2000.
15. The SpoVNet Consortium. Spontaneous Virtual Networks: On the road towards the Internet's Next Generation. *it - Information Technology*, 50(6), December 2008.
16. Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. Cyclon: Inexpensive membership management for unstructured P2P overlays. *J. Network Syst. Manage.*, 2005.
17. Jinling Wang, Jiannong Cao, Jing Li, and Jie Wu. Achieving bounded delay on message delivery in publish/subscribe systems. In *International Conference on Parallel Processing*, 2006.