Adaptive Gossip-Based Broadcast*

L. Rodrigues
U. de Lisboa
ler@di.fc.ul.pt

S. Handurukande
EPFL
sbandara@lpdmail.epfl.ch

J. Pereira
U. do Minho
jop@di.uminho.pt

R. Guerraoui
EPFL
rachid.guerraoui@epfl.ch

A.-M. Kermarrec
Microsoft Research
annemk@microsoft.com

3rd July 2003

Abstract

This paper presents a novel adaptation mechanism that allows every node of a gossip-based broadcast algorithm to adjust the rate of message emission 1) to the amount of resources available to the nodes within the same broadcast group and 2) to the global level of congestion in the system. The adaptation mechanism can be applied to all gossip-based broadcast algorithms we know of and makes their use more realistic in practical situations where nodes have limited resources whose quantity changes dynamically with time without decreasing the reliability.

1 Introduction

Gossip-based broadcast algorithms [1], also called "epidemic" or "probabilistic" broadcast algorithms, do have inherent scalability properties that make them very appealing for disseminating information among a large number of nodes. The underlying idea is very intuitive: every node that receives a message, buffers it, and then forwards it (i.e., gossips it) a certain number of times, each time to a randomly selected subset of processes.

Nevertheless, the applicability of gossip-based broadcast algorithms in a practical setting is limited by their assumption that enough buffering resources exist on all nodes. Indeed, in order to operate in a reliable manner, the nodes participating in the broadcast must be equipped with enough resources to ensure that messages are gossiped a sufficient number of times. If a node does not have enough resources, it may drop a large number of messages that are being forwarded. If several nodes do not have enough resources, reliability might end up being drastically impacted. One might consider calibrating, a priori, the transmission rate of the senders according to the resources available at every

*This work has been partially supported by the FCT project RUMOR (POSI/ 40088/ CHS/ 2001), Microsoft Research Grants (2001-39,2001-47) and Swiss National Science Foundation. Selected section of this report were published in the proceedings of the Proceedings of the International Conference on Dependable Systems and Networks (DSN), pp. 47-56, San Francisco, California, USA, June, 2003.

node. The static flavor of this naive solution makes it unrealistic in a practical scheme, as we explain below.

A large scale publish-subscribe application illustrates the above problem. Nodes may assume one of two roles: publishers which broadcast information and subscribers which register interest in receiving certain types of information.[1] Gossip-based broadcast is typically used here to disseminate the information from the publishers to the interested subscribers. Since different nodes are interested in different types of information, avoiding the delivery of unwanted information usually goes through mapping different types of information to different broadcast groups. Any node may belong to more than one broadcast group, and this number varies as nodes dynamically subscribe to new types or cancel previous subscriptions. Given that the resources at each node are limited, every node has to dynamically divide the available resources among the groups it belongs to.

Gossip-based broadcast algorithms must be able here to adapt to situations where each node has different and varying amounts of resources. To our knowledge, none of the gossip-based algorithms we knew [1, 3, 4, 9, 8, 6] of includes any form of dynamic feedback mechanism. They typically discard messages in overload conditions, without providing to the source (the sender) any feedback regarding the reliability of the operation. In fact, even with the simple setting of a single broadcast group with persistent and uniform buffer resources at each node, the message emission rate might vary, in particular with several senders. The rate of new messages in the system is unpredictable and depends on the sum of the individual emission rates. It is non trivial to estimate the global congestion and control the message emission at each sender accordingly. As we show in the paper, without such a control scheme, the reliability decreases significantly.

This paper proposes a novel adaptive mechanism for gossip-based broadcast algorithms. The idea is to disseminate and gather information about the resources available in a broadcast group such that every sender can adjust its emission accordingly. The challenge consists in ensuring that senders are able to perceive the quality of the algorithm operation, in terms of reliability with the current system configuration, without interacting explicitly with other nodes of the system; such interaction would hamper the distinctive scalability of gossip-based broadcast algorithms.

The intuitive idea underlying our mechanism is the following. We periodically evaluate the available resources in every broadcast group. In each period, nodes gossip the minimum buffer size in the group for that period. They do so by maintaining and gossiping the minimum of their own buffer size with the value received in gossip messages for that period. The value computed at the end of a period is used as the estimation for that period and maintained for a predefined number of periods Δ. Then, during the normal operation of the gossiping protocol, each sender computes the average *age* of messages stored locally that would be discarded if the local buffer was the smallest in the group. The age of a message is the number of times it has been forwarded from one node to another and is directly related with the level of dissemination among nodes. If the average age of messages that would be discarded by members with low

[1] Using one of the several paradigms proposed for this sort of systems, such as subject-based [12], content-based [13] or type-based [2] subscriptions.

buffers is lower than the required age to ensure reliability, the sender decreases its transmission rate. If the average age of discarded messages is higher than needed, then the sender is allowed to increase its transmission rate.

This adaptation mechanism is highly scalable as it does not require nodes to maintain information about every other node in the system. It also does not require additional messages to be exchanged: it relies on a small amount of control information that is included in the header of normal data messages. The mechanism takes into account the dynamic nature of the system and continuously adapts to changing operational conditions. As conveyed by our performance measures, without such a mechanism, the reliability of message dissemination in a large scale gossip-based setting can hardly be sufficient in a practical setting. To evaluate the reliability benefits of using our mechanism, we consider a specific gossip-based algorithm but the idea is general and can be similarly applied to other gossip-based algorithms [1, 3, 4, 9, 8, 6], as we discuss in Section 5.

The rest of the paper is organized as follows. Section 2 presents a brief overview of existing gossip-based broadcast algorithms and discusses the limitations of these algorithm when nodes have limited, heterogeneous and dynamic resources. Section 3 introduces our adaptive algorithm and Section 4 presents its experimental evaluation. Section 5 gives some related work. Section 6 concludes the paper.

2 Background

In gossip-based broadcast algorithms, messages are not disseminated in a deterministic manner. Instead, each group member participates in message propagation by forwarding received messages to a random subset of other group members (i.e., gossiping). Different variants of these algorithms exist [1, 3, 11] and differ in the concrete strategies used to select gossip targets and to bound the number of times each message is forwarded. It can be shown that, given adequate resources, gossiping can be configured to obtain high reliability such that a message is delivered to all processes with a high probability. Basically, both the probability that *(i)* a message is delivered to some but not all processes and that *(ii)* a message that is broadcast by a correct process is never delivered by any process, can be made as small as required, providing almost as much reliability as any deterministic approach.

The decentralized nature of gossip-based dissemination results in algorithms that are scalable to a large number of nodes without overloading any member of the group. The algorithms sustain stable high throughput in large groups, despite node failures, performance perturbations, and lost packets. However, the probabilistic reliability guarantees stand on the assumption that enough buffering resources are available and that message loss in the network is independently distributed. If not, reliability guarantees can be significantly compromised.

2.1 Gossip-based Broadcast

Although our discussion and proposals apply to gossip-based algorithms in general, we use here a concrete algorithm, to motivate the need for the adaptation and to illustrate our idea. We have selected the one of [3] which is depicted in Figure 1 and works as follows.

```
Initially:
    events, eventIds = ∅
every T ms:
    {Update ages}
    for all e ∈ events do
      e.age ← e.age +1
    for all e ∈ events: e.age > k do
      events ← events \ {e}
{Gossip}
    gossip.events ← events
    Choose f random members target_1 ... target_f
    for all j ∈ [1..f] do
      SEND(target_j,gossip)
upon RECEIVE(gossip):
    {Update events and ages}
    for all e ∈ gossip.events
      if e.id ∉ eventIds then
        events ← events ∪ {e}
        eventIds ← eventIds ∪ {e.id}
        DELIVER(e)
      if e' ∈ events such that
      e.id = e'.id and e'.age<e.age then
        e'.age ← e.age
{Garbage collect eventIds}
    while |eventIds| > |eventIds|_m do
      remove oldest element from eventIds
{Garbage collect events}
    while |events| > |events|_m do
      remove oldest element from events
```

Figure 1: Gossip-based broadcast algorithm.

Received events are buffered in *events* buffer. Periodically, at every interval T, a node forwards all stored events to a subset with size f of randomly selected nodes. The gossip period T, fanout f and buffer size $|events|_m$ are configuration parameters of the algorithm[2]. Upon receiving a gossip message, a node buffers each newly received event in *events* buffer and locally delivers it. Duplicates are avoided by keeping a set of identifiers of already received events in *eventIds*. To prevent exhaustion of local resources, the size of *events* is kept bound by some constant $|events|_m$ that is a configuration parameter (in the original algorithm, resources allocated to the algorithm are constant). Therefore, if *events* is full, some old events need to be discarded to make room for new events. In this case, the *age* of the event is used as the criteria to select which event to discard. Age represents how many times a message has been forwarded among nodes [7].

To achieve the desired reliability, each member of the group should be able to store the events it receives for sufficiently long to be retransmitted. The size of available buffers indirectly determines the number of times each event is

[2]The selection of values for these parameters is out of the scope of this paper and addressed in [3].

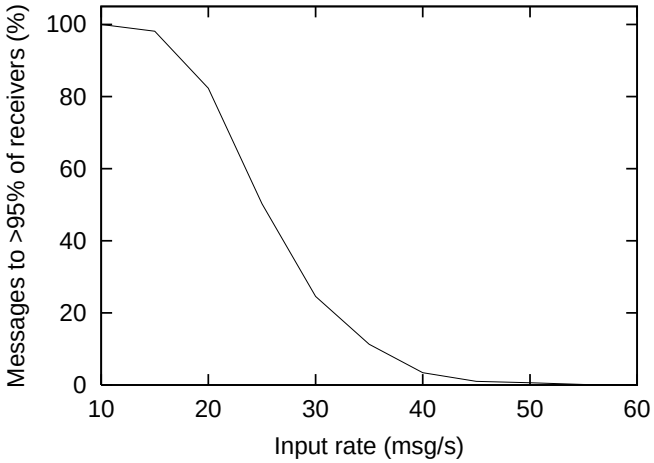


Figure 2: Reliability degradation.

included in a gossip message before being garbage collected and thus becomes a key factor in determining the maximum rate that can be reliably transmitted. Figure 2 shows how the reliability degradation occurs as the message rate increases (experimental conditions are described in detail in Section 4). Given a static configuration of resources, as the message emission rate increases the reliability decreases significantly. The loss of reliability is the consequence of messages being dropped sooner. In fact, from approximately 8.5 hops at 10 msg/s, the average age of messages dropped quickly falls to 3.7 at 30 msg/s and 2.7 hops at 60 msg/s. Correct operation of the algorithm thus requires that a process broadcasts messages only up to a safe rate [1]. This can be done by limiting messages broadcast by each process. Figure 3 illustrates a token-bucket interface that can be used to bound the load imposed on the system.

2.2 Dynamic Flow Control

In systems where the number of nodes is fixed and resources (buffer sizes) are statically allocated, it is possible to configure the application to prevent its load from exceeding the system capacity. However, this approach is not possible when resources change dynamically. Even if resources are fixed, a static configuration of the admissible load is not feasible when the number of senders may change in run-time. Configuring each node for the worst-case (when the maximum number of senders is active) would result in poor resource allocation in all other scenarios.

There are many reasons that may cause the available resources or number of nodes to change at run-time. For instance, a node may dynamically join and leave a group. It is also possible that the capacity of each node changes as it has to split its resources dynamically between multiple applications. For instance, with partially overlapping groups (e.g., subscription to different topics), resources of nodes participating in more than a single group have to be shared.

In a dynamic system, congestion can be avoided by evaluating the capacity of the system and adjusting the allowed emission rate in order not to exceed

```
Initially:
    tokens = max
every 1/rate ms:
    {Restore tokens}
    if tokens < max then
      tokens ← tokens + 1
upon BROADCAST(event):
    {Wait for available tokens}
    wait until tokens > 0
    tokens ← tokens − 1
    {Buffer event}
    events ← events ∪ {e}
    eventIds ← eventIds ∪ {e.id}
```

Figure 3: Bounding the input rate.

available resources. Two types of flow control techniques are used with deterministic broadcast algorithms: *window-based* and *rate-based*. We examine the applicability of each technique to gossip-based algorithms.

Window based flow control is based on imposing a fixed limit on the number of messages that can be in transit in any given moment. Rate is implicitly controlled by blocking senders which attempt to go over the limit. Progress depends on detecting stability and garbage collecting messages. Notice that even highly scalable stability tracking mechanisms [5] require feedback from all processes. This means that a full membership has to be known and state is proportional to group size.

Rate based flow control is more flexible, by explicitly adjusting the maximum rate of the sender according an evaluation of the system capacity. The challenge is thus to evaluate system capacity without feedback from each of the receivers. For instance, the desired rate could be calculated by observing the message rate that is being delivered to each receiver and propagating it back to senders, but this would not scale and would be sensitive to local performance perturbations unrelated to resource availability.

2.3 Intuition

Our proposal stems from the observation that the age of messages that are discarded is lower when the system is congested and that this can be observed without additional overhead by all nodes. Consider the following simulation results. For each buffer configuration in our test system, we experimentally determine the maximum input rate that results in good reliability guarantees (more precisely, the buffer configuration that was able to deliver messages to at least an average of 95% of participant processes). The results are presented in Figure 4. For each of these configurations, we record the average age of messages being dropped. Interestingly, the average age of messages being dropped when the system is about to become congested is the same for all buffer sizes and equals 5.3 hops in this system configuration. This can be observed inde-

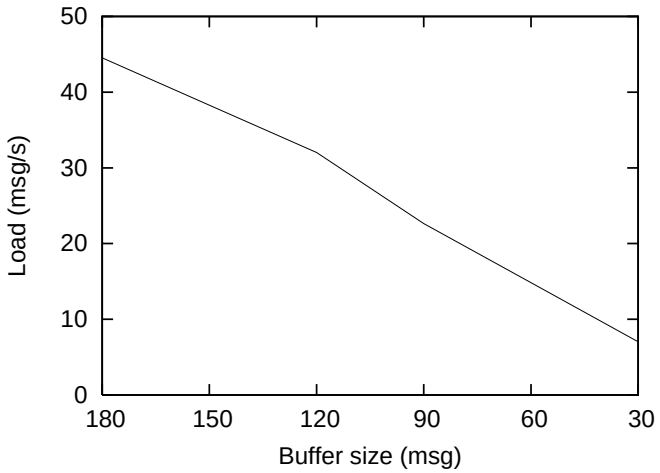


Figure 4: Maximum input rate.

pendently by each participant node with no additional protocol overhead. Our proposal is thus to use the average age of messages being dropped as a measure of congestion.

Notice that this is effective only when buffer availability is the same at all processes. If a node has a much larger buffer, it will observe a higher average age of dropped messages and thus will be unable to observe the actual state of the system. To obtain useful mechanism, it is required that a node estimates the age of messages being dropped at the node in the system with less available resources. A scalable mechanism to achieve this goal is presented and evaluated in the following sections.

3 Adaptive Mechanism

Since reliability can be compromised in case of buffer overflow, our goal is to provide each node with enough information such that the nodes can estimate the resource constraints of other nodes and adjust the rate of messages being broadcast accordingly. This is a challenging problem with gossip-based algorithms since there is no centralized control or global knowledge. Imposing explicit feedback on gossip-based algorithms would endanger both scalability and throughput stability.

There are two possible classes of mechanisms to compute the necessary information about remote resources without relying on a feedback mechanism: 1) distributed computation of resource availability using the gossip algorithm itself and 2) local estimation of resource usage by observing the traffic. Although distributed computation can accurately collect information from heterogeneous remote nodes, the latency induced to gather this information in a scalable fashion might imply long delays to react to changes. On the other hand, local observation of traffic allows low latency in evaluating resource usage. But symmetry between nodes in gossip-based algorithms makes local observations of traffic a good estimate of resource usage in distant nodes only when resource availability

is homogeneous. In Figure 5, we present an adaptation algorithm obtained by integrating our flow control technique in the algorithm of Figure 1. The new algorithm embodies the two types of mechanisms discussed in the previous paragraph: It uses a distributed mechanism to determine *resource availability* (i.e. the size of buffers), which changes only upon reconfiguration of nodes. A local mechanism is then used to determine *resource usage* (i.e. buffer occupancy), whose variation is far more being frequent and unpredictable as it is affected by the timing of senders and network delays. The resulting information from the combined mechanisms can then be used to adjust the rate at which each node is allowed to send messages. We examine each of these issues in the following sections.

3.1 Estimating Buffer Availability

Our approach uses a distributed mechanism to estimate the size of buffers available in remote nodes. Since we are interested in preserving the resilience of the protocol, we aim at the node that has less resources. Notice that the inherent redundancy in gossiping could overcome some message loss that results from a single node with less resources. This redundancy should however be used to cope with transient perturbations and thus preserved as a safety margin.

We denote the estimate of the size of the smallest buffer in the group by *minBuff*. Estimation of *minBuff* could be achieved by letting each node disseminate its own available resources and let every node in the group collect all these values and select the minimum of the set of collected values. However, such solution would not be scalable as it would require each node to gather values from every other node in the system. To circumvent this, we compute the value of *minBuff* in a decentralized manner using the same gossip messages used for data.[3] In detail, each process keeps a current known minimum value and adds it to all outgoing gossip messages. Upon reception of a gossip message, if the value received is lower than the currently known minimum, the local minimum is updated. Eventually, all processes discover the absolute minimum in the group.

Additionally, the algorithm must be able to cope with dynamic changes in the available resources at each node. For instance, if the process with less resources leaves the group, the estimate of resource availability becomes obsolete and should be forgotten to allowing full utilization of current resources. This is achieved by keeping a separate estimate of *minBuff* for each period of time, that depends only on the actual buffer sizes in that period.

The algorithm is presented in Figure 5(a) and works as follows. The interval for each estimate is called the sample period S. For each sample period s and in each process p, the algorithm computes $minBuff_p^s$. Each $minBuff_p^s$ is initially the size of the buffers available locally in p. In every gossip round, values s and $minBuff_p^s$ are included in the message header. Every time a node q receives a message from another process p, updates its own estimate of *minBuff* for period s, simply by setting $minBuff_q^s$ to $min(minBuff_q^s, minBuff_p^s)$. Notice that, using this approach, at the start of each period, nodes have an inaccurate estimate of remote resources in $minBuff_p^s$. The immediate use of this estimate would lead to fluctuations of the allowed input rate. In our algorithm, undesirable

[3] This is similar to an aggregation function [6].

Initially:
    s = Δ
    minbuff$_p^r$ = |events|$_m$, for all $1 \leq r \leq \Delta$
every T ms:
    ...
    {Add information to gossip message}
    gossip.s ← s
    gossip.minBuff ← minBuff$_p^s$
    ...
upon RECEIVE(gossip):
    ...
    {*Compute new known minimum*}
    if gossip.s=s ∧ gossip.minBuff < minBuff$_p^s$ **then**
      minBuff$_p^s$ ← gossip.minBuff
every S ms:
    emph{Enter new period}
    $s \leftarrow s + 1$
    minBuff$_p^s$ ← |events|$_m$
    minBuff = min(minBuff$_p^s$,...,minBuff$_p^{s-\Delta+1}$)

(a) Distributed discovery of resource availability.

Initially:
    avgAge = $(H + L)/2$
    lost = ∅
upon RECEIVE(gossip):
    ...
    {*Update congestion estimate*}
    while |events\lost| > minBuff **do**
      select oldest element e *from* events \ lost
      avgAge = α avgAge $+(1-\alpha)$ e.age
      lost ← lost ∪ {e}
{*Garbage collect events*}
    while |events| > |events|$_m$ **do**
      remove oldest element e *from* events

(b) Local estimation of congestion.

every T ms:
    ...
    {*Throttle sender*}
    if avgAge > H ∧ avgTokens<max/2 ∧
      ∧ rand > W **then**
      rate ← rate×$(1 + r_H)$
    if avgAge < L ∨ avgTokens>max/2 **then**
      rate ← rate×$(1 - r_L)$

(c) Rate adaptation.

Figure 5: Adaptive gossip-based broadcast.

fluctuations are avoided by selecting the minimum from a series recent values $minBuff_p^s$, $minBuff_p^{s-1}$, ..., $minBuff_p^{s-\Delta+1}$. This ensures that while a proper value is being computed for period s, the value in use takes into account the values in a pre-defined number of the previous periods. The value for Δ, as well as for S, are selected when configuring the algorithm.

This algorithm depends on loosely synchronized clocks to determine the sample period s. The required synchronization can easily be achieved by making each process advance s to $gossip.s$ upon reception of a gossip message from a later sample period.

3.2 Estimating Buffer Congestion

Given the estimate of the size of the smallest buffer in *minBuff*, it becomes possible to estimate congestion in remote nodes using only local information. This is done by accounting the age of messages which would have to be discarded in a buffer with size *minBuff*. The algorithm for this is presented in Figure 5(b) and works as follows. Upon receiving each gossip message, after storing events and updating their ages, *minBuff* is used as a threshold to select which events would have to be discarded. The age of such events is used to update a moving average *avgAge* which estimates the average age of messages being discarded by a process with exactly *minBuff* buffers. Events already accounted for are stored in *lost* to avoid using them twice in the calculation. The resulting value for *avgAge* can then be used whenever the process wants to adjust the sending rate.

Notice that although the average age is computed according to *minBuff*, the full size of the local buffer is used to store events, thus improving overall reliability. The sensitivity of *avgAge* to transient perturbations depends on the α parameter used to update the moving average which must be chosen when configuring the algorithm (see Figure 5(b)).

3.3 Adjusting the Sender's Rate

The input rate allowed to each sender can be adjusted by comparing the estimate of the age of messages being discarded, measured in *avgAge*, with the ideal age a_s, obtained analytically or experimentally (for instance, Section 2.3 describes how the critical age required to deliver messages to at least an average of 95% of participant processes in our system is calculated). Basically, the goal of the flow control mechanism is to have the senders decrease their rate if the estimated average age is less than the critical age value, and increase their rate otherwise. Senders can make this decision locally, by comparing their estimate of *avgAge* with the target critical age a_s.

The adjustment should also depend on the actual usage of the allowed rate by the application. If not, an application could temporarily reduce its sending rate below the allowed rate, causing *avgAge* to increase and its allowed rate to increase unbounded. Then, by suddenly increasing its sending rate to match the artificially inflated allowed rate, it would be able to congest the system. The usage of the allowed input rate can be estimated by observing the average number of tokens *avgTokens*.

To avoid that each sender changes its rate with every minor oscillation of *avgAge*, causing a continuous oscillation in the system, the algorithm presented

in Figure 5(c) uses two threshold values: a *low-age* mark, $L < a_s$ and a *high-age* mark, $H > a_s$. A sender decreases its rate if the average age goes below the low-age mark L or if the allowed input rate has been unused as indicated by a high value in *avgTokens*. The allowed rate increases again when the average age becomes higher than the high-age mark H and the previous allowed input rate has been fully used as indicated by a low value in *avgTokens*.

When the system is congested (i.e., the average age is below L), the sender reduces its rate by some amount denoted r_L. Similarly, when new resources are released in the system, and the rate can be increased (i.e., the average age is above H), the sender increases by an amount denoted r_H. Both the decrease and increase are relative to the actual sender rate at the moment the adjustment is performed.

Furthermore, in a group with a large number of senders, if all sources increase their rate at the same time, even if just by some small amount, it may happen that the load increases abruptly, causing the system to move from the low-age mark to the high-age mark very quickly and causing oscillations as a result of the adaptation mechanism. Therefore, we introduce some randomization in the rate increase procedure. In each round, if there are resources available to increase the rate, each sender uses randomization to decide if it increases its rate immediately or if it should wait for the next round (this is controlled by constant W).

3.4 System configuration

The configuration of parameters f, T, $|events|_m$, $|eventIds|_m$ and k used in the original algorithm of Figure 1, as well as determining the value for a_s are out of the scope of this paper and described elsewhere [3]. The configuration of *minBuff* computation parameters S and Δ, the parameter α used in computing moving averages, and of adaptation parameters L, H, r_L, r_H and W are discussed here.

Determining a value for the sample period S. For a message to propagate to all members it can take up to a_s gossip periods. Hence if a node with the minimum buffer size in the group gossips about it's buffer size, it takes a_s periods this message to be propagated to all other nodes. When every node comes to know about this minimum buffer size, all the nodes can set their buffer size appropriately. Therefore, setting S to a value not less than $a_s \times T$ ensures with high probability that in every sample period the minimum buffer size of a single node reaches all others. If it is known that the minimum value is shared by several nodes, a lower value for S can be used. Therefore, in our experiments we used $S = 2 \times T$.

Determining a value for Δ. The Δ represents a size of a time duration in which the values of *minBuff* are taken into account when deciding the actual buffer size by a process. A too high value (i.e. a bigger time duration) will cause under-utilization of the bandwidth in the broadcast group. A lower value might cause the global minimum buffer size to be changed abruptly. The value of Δ depends on the operation environment of the algorithm. For example in an environment where communication links and nodes become faulty and recover quickly, a higher value for Δ is suitable. In our experiments we have used $\Delta = 2$.

Determining a value for α. The parameter α is used to weight the computation of moving averages. To avoid sudden oscillations of *avgAge* and *avgTokens* when the inter-arrival time of messages has high variance, this should

be set close to 1. If the traffic is strictly periodic, this value can be lowered in order to improve reaction time. In the experiments presented in this paper we have used $\alpha = 0.8$.

Determining values for L and H. For emission rate to be stable without any oscillations, there should be a considerable difference between L and H values. For quick reactions to decrease in resources or global increase in emission rate, L should be close to a_s. On the other hand for better responsiveness in terms of increasing the emission rate, H should be also close to a_s. According to Section 2.3 we set $H = 7$ and $L = 5$.

Determining the value of r_H, r_L and W. Nodes react quickly for congestion by reducing their emission rate when the value of r_H is high. The nodes recover quickly and increase their emission rate when the r_L is high. As the nodes use local knowledge available to them to estimate congestion levels, it is possible that they mis-estimate the global congestion level for an instantaneous time duration. Though these cases are infrequent, having r_H and r_L closer to each other is more forgiving in terms of mis-estimations. In our experiments we have used $r_H = 5\%$ and $r_L = 5\%$. Also, we have set $W = 0.5$ (that is, on average only 50% of the nodes increase their rate).

Notice that the values chosen for the parameters discussed in the previous paragraphs are not independent. By setting Δ higher, one can reduce the value for S, as even if a minimum does not reach all nodes in a single period, it is unlikely that this happens for the same nodes in two consecutive periods. On the other hand, by setting the value of α higher, thus reducing the oscillations in *avgAge* one can make L and H closer to a_s. On the other hand, increasing both α and the distance of thresholds L and H to a_s, one can increase both r_H and r_L.

4 Experimental Results

In this section we evaluate our algorithm using simulations and experimental results obtained from our prototype implementation.

Experimental Settings. For the evaluation, we used a simple event-based simulation model as well as a full implementation. The simulation model allowed us to experiment with parameters of the algorithm and to obtain a detailed analysis of the behavior of the algorithm. The implementation, based on Java 2 Standard Edition, is used to validate simulation results in a real setting. Experimental results presented in this section were obtained using 60 processes which implement the gossip-based algorithm. These processes are deployed on 60 workstations which are connected by an Ethernet local area network. For the gossip-based algorithm, we use a fanout of 4 and a gossip period of 5 seconds, i.e., each process gossips to 4 other members in every 5 seconds. To compare the behavior of a conventional gossip-based algorithm with an adaptive variant of the algorithm, we run these two algorithms for a sufficiently long period of time with multiple senders.

Effectiveness of Rate Adaptation. In Section 2, we have shown that, for a given system and a given buffer availability, there is a maximum load above

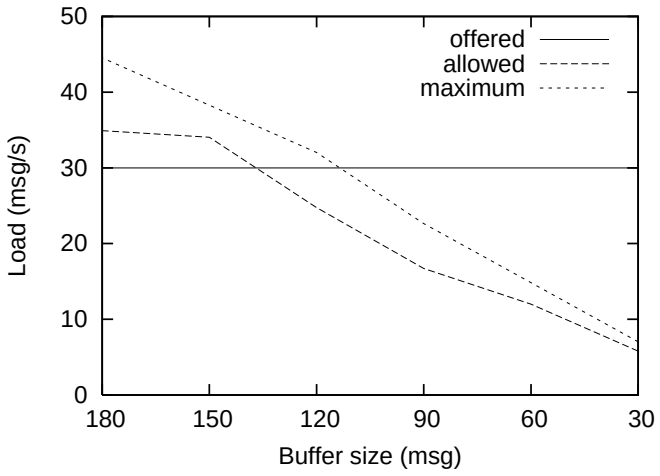


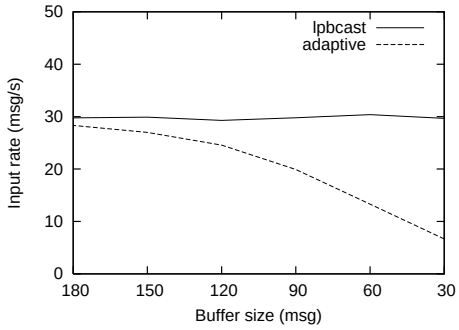
Figure 6: Ideal and adaptive rates.

which high reliability cannot be ensured. To promote reliability and a maximum resource utilization, our adaptive mechanism should allow the senders to approximate that "ideal" target load (note that the target load depends on the desired degree of reliability; we our case we have configured the system to allow messages to be delivered, on average, to 95% of the participants).

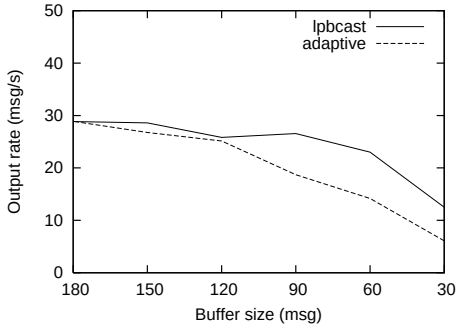
To show that our algorithm converges to the desired values, we have run a series of simulations with all processes using the same and progressively smaller buffer size. The offered load is constant and set to 30 msg/s. This was done both with the original algorithm as well as with the adaptive algorithm. Figure 6 shows the maximum load (the same as in Figure 4), the load being offered and the maximum allowed rate as dynamically computed by our algorithm in a configuration. When the intended load exceeds the system capacity (i.e. buffer size is less than 120 messages), the algorithm successfully approximates the "ideal" rate. When the load does not exceed system capacity (i.e. buffer size is greater than 120 messages), the offered load is accepted.

Rate Adaptation and Atomicity. To show the impact on reliability of our adaptive algorithm we use the same series of simulations with progressively smaller buffers sizes and a constant offered load of 30 msg/s. The results are presented in Figure 7.

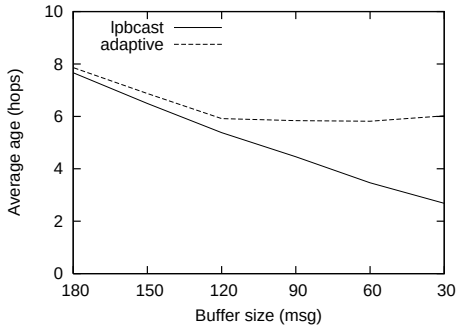
As shown in Figure 7(a), the original lpbcast algorithm does not bound its input rate and thus it equals the offered load of 30 msg/s. As this is higher than the maximum rate that can be transmitted reliably, many messages are lost, as can be seen in Figure 7(b) (in the case of lpbcast). It can also be observed in Figure 7(c) that the average age of dropped messages decreases significantly. In contrast, when using the adaptive algorithm the input rate equals the output rate, indicating that no messages are being lost. The impact of messages lost when no adaptation is being done is presented in Figure 8(a). Notice that the average number of receivers of each message remains constant when using the adaptive algorithm but degrades with low buffer sizes with the original algorithm. A better measurement of atomicity is presented in Figure 8(b),



(a) Input rate.

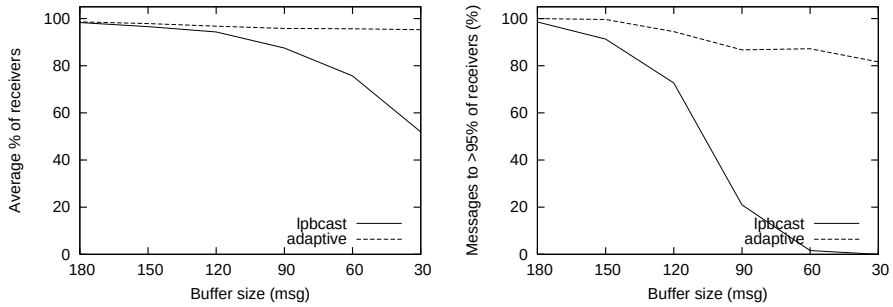


(b) Output rate (input-loss).



(c) Age of dropped messages.

Figure 7: Rates and average ages.



(a) Average number of receivers.

(b) Number of atomically delivered messages.

Figure 8: Reliability degradation.

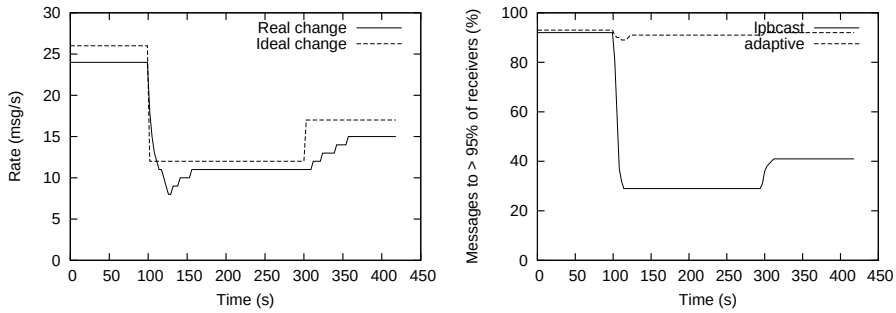
which shows that the percentage of messages delivered to almost all nodes drops sharply when using the original algorithm thus failing to meet bimodal guarantees.

Adaptation to Dynamic Buffer Size. The previous simulation results only show the behavior of the algorithm after convergence of the adaptive mechanisms. It is also interesting to study the dynamic behavior of the algorithm, i.e, how fast it reacts to changes in the system resources. It is also fundamental to validate the algorithm in a scenario where nodes have different resources. Finally, it is important to verify if a concrete implementation behaves as the simulation results predict.

To validate these points, we have subject the implementation to an experimental scenario where buffer resources change in run-time. The system is started in a configuration where the input load does not exceed the system capacity. Then, at a given point, 20 nodes (i.e., 1/3 of the nodes) reduce their buffer availability; more precisely, in these processes the buffer space is reduced from 90 to 45 messages. Later on, these nodes increase again their buffer availability but to a value still lower than the initial value (from 45 to 60 messages), and therefore unable to sustain the input load. In this scenario, senders try to impose the same input load but the adaptive mechanism adjusts their rate to a value close to the "ideal" value. Note that this scenario illustrates the behavior of the algorithm both in the case where the resources decrease and increase.

The results are illustrated in Figure 9. The decrease of buffer availability occurs at round $t = 100$ from a value 90 to 45 and then increase at $t = 300$ to a value of 60. The horizontal dotted lines depict, for each system configuration, the "ideal" maximum input load that preserves the target reliability measure. As it can be seen, the adaptive mechanism quickly moves the allowed input to value that is close to the target and then smoothly stabilizes until no instability can be observed around 60s after the configuration change.

Figure 9(b) shows the atomicity figures for the original lpbcast algorithm and for our adaptive algorithm in equivalent experimental runs. As expected, the atomicity figure of lpbcast drops when the resources cannot sustain the input



(a) Allowed Rates.

(b) Atomicity.

Figure 9: Dynamic Buffer Size

load while the adaptive mechanisms allows to preserve a satisfactory atomicity figure. It is also interesting to note that, as expected, the atomicity values are here slightly higher than in the simulations where all nodes decrease their buffer space. Namely, with buffers available for 60 messages we obtain an atomicity of 87% with simulation and of 92% with experiment. This illustrates that the algorithm effectively uses all the buffers available locally, thus the atomicity values benefit from the buffer capacity available at nodes that did not have their resources decreased.

5 Related Work

Gossip-based algorithm have recently been an active area of research [3, 4, 9, 8, 6] and several optimizations have been proposed. These optimizations target different goals and could be combined with our proposal, as we discuss below.

An efficient buffering technique has been proposed in [10] in the context of Bimodal multicast [1] to determine which group member should buffer messages on a long-term for recovery purposes. Each message, is associated with a fixed number of *bufferers* (this number depends on the size of the system and the probability of node failure) responsible for buffering a message for recovery purposes. The *bufferers* can be easily identified by hashing the message identifier. Nodes which need to recover from a message loss contact directly bufferers. Although this technique limits the amount of buffering required for a message, it targets recovery phases of gossip-based algorithms. Moreover, it assumes that each node has a full knowledge of the group membership since it might need to contact any node. Instead, our approach targets the buffering requirements and adaptation for the gossip-based algorithms to succeed in the first place. Moreover, our mechanisms could be applied to a gossip-based algorithm relying on a partial membership knowledge on each node.

Buffering management has also been addressed in [7] to efficiently purge gossip buffers and garbage collect messages. This approach purges messages according to their age, which is determined by the number of gossip rounds each instance of a message has been gossiped. The age of a message proves to be a

good estimate of level of propagation of a message. The goal of the work in [7] is to provide a good heuristic to efficiently manage buffers but no adaptation to varying resources availability is used there. However, in the approach described in this paper, we use an age-based heuristic to purge buffers.

Instead of reducing the sender's rate, if slower senders are expected to eventually recover, it is possible to repair lost messages from a log [14]. This has the inconvenient of requiring possibly very large buffers at logging servers and to deliver some messages much later to some processes.

Network congestion also results in correlated message loss thus degrading reliability. This is a potential weakness of the approach since, with gossip-based algorithms, the network usage is notably high in comparison with deterministic reliable broadcast algorithms. The usage of message semantics to discard obsolete messages in order to ensure reliability for recent messages has also been proposed [11]. Recognizing that the probability of the message being delivered to all processes grows with the size of the initial set of processes receiving the message, it has been suggested that an initial optimistic broadcast phase is used [1]. The gossip phase is then done by negative acknowledgments thus saving network bandwidth. Finally, given some knowledge about the network topology, it is possible to better spread network traffic across physical links, thus optimizing the usage of wide area networks [9].

6 Concluding Remarks

This paper presents a novel adaptation mechanism for gossip-based algorithms. This mechanism allows every node to adjust its gossiping rate according to the resources available within other nodes of the system and also to adjust the message emission rate according to the global congestion. As a result, the reliability properties of the broadcast service is increased. Our scheme allows gossip-based broadcast algorithms to be adapted to settings where nodes have heterogeneous resources that may change dynamically. In such settings, it is impossible to adjust the parameters of the gossip algorithm off-line, before the algorithm is deployed. Our mechanism is scalable, as it does not require any node to explicitly interact with or collect information about other nodes in the system. Instead, the dissemination of the information required to perform adaptation is embedded in the normal gossip of data messages. Our mechanism was validated experimentally, using both simulations and an implementation executing in a network of 60 workstations.

It is important to notice that the goal of our adaptation mechanism is not to recover from past message omissions but prevent future ones. That is why we advocate the need for quick adjustment of senders rate. Additional techniques have to be deployed to recover from lost messages.

Furthermore, our mechanism adapts to the node with the smallest amount of resources. It could also be extended to consider alternative criteria. For instance, the algorithm can be easily extended to compute not only the smallest, but the n smaller buffers in the system (or the n smaller buffers above a minimum threshold) to prevent a single node from affecting the performance of the whole group.

Acknowledgment

We are very grateful to Petr Kouznetsov and Patrick Eugster for their support in developing the lpbcast and its prototype.

References

[1] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, May 1999.

[2] P. Eugster, R. Guerraoui, and C. Damm. On objects and events. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 2001)*, October 2001.

[3] P. Eugster, R. Guerraoui, S. Handrukande, A.-M. Kermarrec, and P. Kouznetsov. Lighweight probabilistic broadcast. In *Proceedings of IEEE Intl. Conf. on Dependable Systems and Networks (DSN'2001)*, 2001.

[4] A.J. Ganesh, A.-M. Kermarrec, and L. Massoulie. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, Feb, 2003.

[5] K. Guo. *Scalable Message Stability Detection Protocols*. PhD thesis, Cornell University, Computer Science, May 1998.

[6] I. Gupta, R. van Renesse, and K. Birman. Scalable fault-tolerant aggregation in large process groups. In *Intl. Conf. Networked Computing and Applications (NCA'2001)*, 2001.

[7] P. Kouznetsov, R. Guerraoui, S. Handurukande, and A.-M. Kermarrec. Reducing noise in gossip-based reliable broadcast. In *Proceedings of IEEE Intl. Symposium on Reliable Distributed Systems (SRDS 2001)*, New Orleans, USA, October 2001.

[8] M. Lin, K. Marzullo, and S. Masini. Gossip versus deterministic flooding: low-message overhead and high-reliability for broadcasting on small networks. In *Proceedings of Intl. Symposium on Distributed Computing (DISC 2000)*, Toledo, Spain, October 2000.

[9] M.-J. Lin and K. Marzullo. Directional gossip: Gossip in a wide area network. Technical Report CS1999-0622, University of California, San Diego, Computer Science and Engineering, June 16, 1999.

[10] O. Ozkasap, R. van Renesse, K.P. Birman, and Z. Xiao. Efficient buffering in reliable multicast protocols. In *Proceedings of Intl. Workshop on Networked Group Communication*, November 1999.

[11] J. Pereira, L. Rodrigues, R. Oliveira, and A.-M. Kermarrec. Probabilistic semantically reliable multicast. In *Proceedings of Intl. Conf. Networked Computing and Applications (NCA'2001)*, 2001.

[12] D. Skeen. *Vitria's Publish-Subscribe Architecture: Publish-Subscribe Overview*. http://www.vitria.com, 1998.

[13] R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman, and M. Ward. Gryphon: An information flow based approach to message brokering. In *Proceedings of Intl. Symposium on Software Reliability Engineering (ISSRE '98)*, November 1998.

[14] Q. Sun and D. Sturman. A gossip-based reliable multicast for large-scale high throughput application. In *Proceedings of IEEE Intl. Conf. on Dependable Systems and Networks (DSN'2000)*, 2000.