HyperCBR: Large-Scale Content-Based Routing in a Multidimensional Space

Stefano Castelli
University of Trento, Italy
castelli.stefano@gmail.com

Paolo Costa
Vrije Universiteit, Amsterdam, The Netherlands
costa@cs.vu.nl

Gian Pietro Picco
University of Trento, Italy
picco@dit.unitn.it

***Abstract*—Content-based routing (CBR) is becoming increasingly popular as a building block for distributed applications. CBR differs from classical routing paradigms as messages are routed based on their *content* rather than their destination address, which fosters decoupling and flexibility in the application's distributed architecture. However, most available systems realize CBR by relying on a tree-shaped overlay network and adopt a routing strategy based on broadcasting subscription requests, thus hampering applicability in very large-scale networks.**

In this paper, we observe that a fundamental underpinning of any CBR protocol is for messages and subscriptions to "meet" at some points in the network. In the approach we propose here, called *HyperCBR*[1], we enforce this topological property in a multidimensional space, by routing messages and subscriptions on different, albeit intersecting, partitions. We derive an analytical model of HyperCBR, validated through simulation, and use it to evaluate our approach in two relevant CBR contexts—content-based searches in peer-to-peer networks, and content-based publish-subscribe. The results show that our protocol achieves efficient CBR even in *very* large scale settings (e.g., millions of nodes) while at the same time opening up intriguing opportunities for deployment-time tuning based on the expected traffic profiles. The analytical evaluation is complemented by simulation results relying on a CAN-based implementation, showing that HyperCBR generates a small forwarding and matching load, and that it is able to tolerate high churn with low overhead.

I. Introduction

In content-based routing (CBR), senders do not specify message recipients using a unicast or multicast address. Instead, they simply inject messages in the network, which determines their routing based on the nodes' interests. These identify the relevant classes of messages based on their content, e.g., using key-value pairs or regular expressions. Therefore, in CBR it is the receiver that determines message delivery, not the sender.

This implicit, content-based style of communication increases decoupling and fosters flexibility in the resulting distributed architecture. Indeed, CBR found application in many contexts, including publish-subscribe [1], distributed databases [2], file sharing [3], and data collection in wireless sensor networks [4]. Hereafter, we adopt the terminology made popular by publish-subscribe, and refer to interests as *subscriptions*, to application messages as *events*, and to filters enabling content-based matching as *patterns*.

Protocols for CBR typically rely on a tree overlay interconnecting the application-level routers, called *brokers*. As for routing, the most common strategy is arguably *subscription forwarding*, used in Siena [5], that establishes the routes followed by matching events by broadcasting subscription requests. However, hardware miniaturization and networked embedded systems, along with peer-to-peer networks and grid computing, are enabling visions of pervasive large-scale systems involving millions if not billions of devices. It is unclear whether the aforementioned mainstream CBR approaches can keep the pace with these *very* large-scale systems.

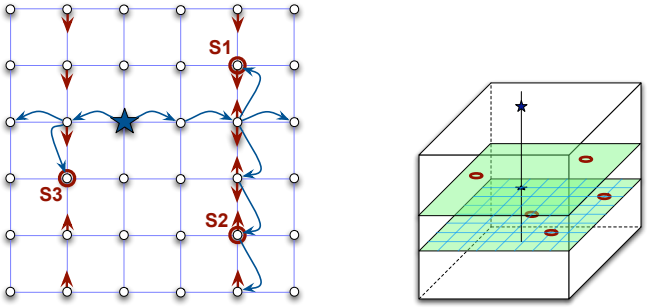
In this paper we illustrate *HyperCBR*, our proposal to enable CBR in very large-scale networks. HyperCBR, described in Section II, exploits some topological requirements about the routing of events and subscriptions, and relies on an application-level routing infrastructure shaped as a *multidimensional* grid. Subscription and events are routed on distinct, albeit intersecting, partitions of a d-dimensional hyperspace. Besides improving scalability, this enables tuning of HyperCBR for different traffic profiles by changing the shape of these partitions, e.g., to optimize the routing of subscriptions over events, or vice versa. Moreover, HyperCBR is independent from the format of events and subscriptions.

Given our focus on very large-scale systems, we characterize HyperCBR analytically, to enable its evaluation in networks whose size is well beyond what allowed by simulation. Section III illustrates an analytical model of traffic, validated in Section IV through simulation in scenarios up to 100,000 nodes. The traffic estimates derived analytically in most cases come within 1% of the simulated ones, therefore allowing us to evaluate, in Section V, the effectiveness of our routing in two CBR application domains—content-based searches in peer-to-peer file sharing and publish-subscribe—with networks of millions of nodes. Results show that HyperCBR provides scalability (orders of magnitude) higher than mainstream CBR approaches, due to its massive decentralization. The analytical evaluation of traffic is complemented by simulations showing that HyperCBR imposes only a limited load on brokers. In Section VI, we discuss how we rely on CAN [6] for implementing the hyperspace, illustrate our solution for dealing with dynamic topologies, and show through simulation that HyperCBR tolerates high levels of churn with small overhead. The paper is completed by related work in Section VII and concluding remarks in Section VIII.

II. HyperCBR: An Overview

Rationale. Our approach stems from the observation that a CBR protocol must guarantee that the routes followed by

[1]Pronounced HyperCyBeR.



(a) $d = 2$. (b) $d = 3$.

Figure 1. The basic idea behind HyperCBR. Small circles represent nodes. A circle around a node denotes a subscriber, the star denotes a publisher. Straight arrows show the routing information established by subscription propagation. Curvy arrows show the route followed by the event generated by the publisher.

published events and by subscriptions intersect in at least one node. For example, in subscription forwarding (e.g., [5]) subscriptions are sent to all the brokers, and therefore the published event meets subscriptions immediately at the publishing node. Conversely, in flooding-based peer-to-peer protocols like Gnutella [7] events (queries) flood the network and meet subscriptions (files) only at the node hosting them.

We generalize this notion by assuming that the dissemination of events and subscriptions is performed through different partitions of the nodes. Let $\mathcal{N}$ be the set of nodes in the system. Subscriptions are disseminated through partitions $\mathcal{S}_i$, such that:

$$\forall i, j: \quad \mathcal{S}_i \subseteq \mathcal{N}, \quad \mathcal{S}_i \cap \mathcal{S}_j = \oslash, \quad \bigcup_i \mathcal{S}_i = \mathcal{N}$$

i.e., they cover without overlapping the whole set of system nodes. Similar definitions hold for the event partitions $\mathcal{E}_i$. The two types of partitions are related by the following constraint:

$$\forall i, j: \quad \mathcal{E}_i \cap \mathcal{S}_j \neq \oslash$$

i.e., each event partition must intersect *all* subscription partitions, and vice versa. When these properties holds, an event sent to *all* the nodes in a partition $\mathcal{E}_i$ is received by at least one node in each subscription partition $\mathcal{S}_j$.

Several choices are possible. Here, we focus on partitions determined by the structure of a d-dimensional space.

HyperCBR: Basic Concepts. Figure 1(a) illustrates our approach on a bidimensional grid ($d = 2$) where we route subscriptions along columns ($\mathcal{S}_j$) and events along rows ($\mathcal{E}_i$).

Subscriptions, and unsubscriptions, are routed similarly to the *subscription forwarding* strategy used in Siena [5]. Differently from Siena, however, its use is restricted to a single subscription partition $\mathcal{S}_j$, instead of the whole system. When a subscription to a new pattern p is issued by a node (e.g., S_1 in Figure 1(a)), it propagates along the entire column the node belongs to. If a second subscription to p is issued by another node on the same column (e.g., S_2) this propagates along it only up to the closest subscriber to p (S_1 in our case). However, if this second subscription is issued on a column different from the first one (e.g., by S_3), the subscription propagates independently throughout that second column.

As for events, our routing strategy is made of two constituents. First, events are routed through their own partitions, rows in our case. Our partitioning choice satisfies the constraints above: an event disseminated along a row crosses *all* the columns containing matching subscriptions, as long as it is routed along the whole row. Second, when an event hits a column containing matching subscriptions, it is "captured" by that column and duplicated along it as shown in Figure 1(a), by following the path established by subscriptions. In this respect, HyperCBR behaves exactly like subscription forwarding, and ensures that the path traveled by an event is the minimal one.

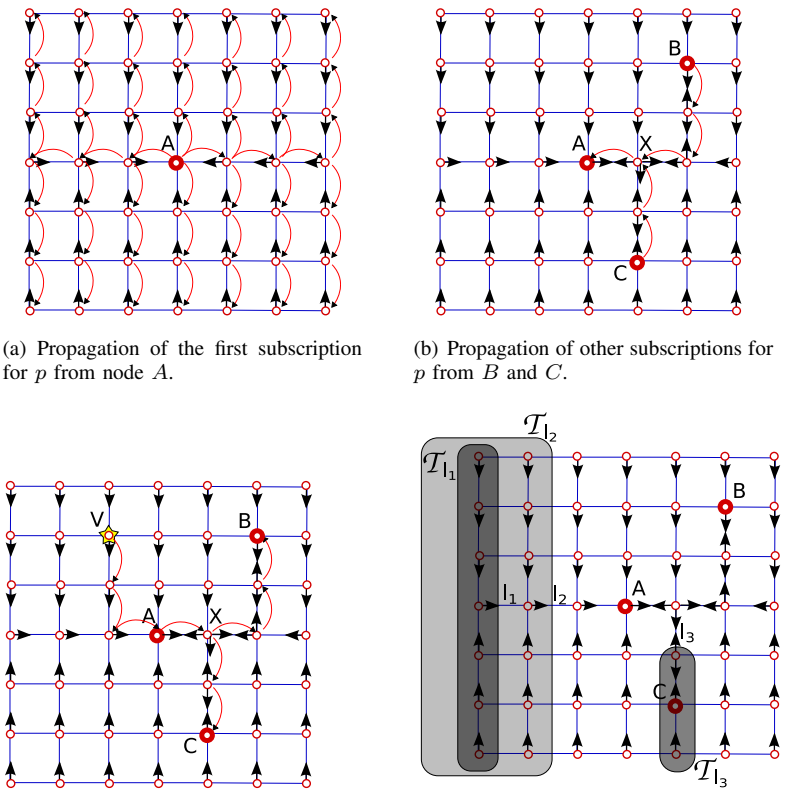
The concept can be extended to an arbitrary number of dimensions. Figure 1(b) visualizes a configuration with $d = 3$ dimensions, where subscription partitions are planes slicing through the cube and parallel to one of the sides, and event partitions are lines orthogonal to such planes. This example is used throughout this section for illustration purposes, as it is easier to reason up to three dimensions. In practice, however, d may be higher than 3, as discussed in Section V.

Routing in Hyperspace. Multidimensional partitions require some additional machinery to enable efficient subscription dissemination and subsequent event routing. As shown in Figure 2(a), which depicts the bottom subscription plane $\mathcal{S}_j$ of Figure 1(b), in this case a subscription is first propagated along one dimension of the plane (e.g., a row in the figure). At each node, the subscription is subsequently propagated along the other dimension (a column in our case), and so on along all the dimensions of the partition, until all the nodes in the partition received the subscription and updated their routing table accordingly. Effectively, the propagation of the first subscription sets up a tree rooted at the first subscriber, A in our case. We call this tree the partition's *subscription tree*. The propagation of other subscriptions for the same pattern p proceeds as discussed above, by laying only the routes connecting the new subscriber to the old ones, similar to the bidimensional case. Figure 2(b) provides an example.

A subscription for a different pattern p' is instead propagated by "reusing" the subscription tree established by the first subscriber for p on the same partition. Essentially, the subscription tree is used as a sort of overlay spanning the partition. The subscription for the new pattern is first routed to the root of the subscription tree, leveraging the routes already in place. From there, the subscription tree is then used to reach all the nodes in the partition, which insert the new pattern in their routing tables. This scheme ensures that all the patterns are disseminated in the same way in $\mathcal{S}_j$, and no loops are created. Unsubscriptions are handled analogously, as in Siena.

As for events, when they intersect a subscription partition (a plane in our case) they simply follow the routes present on it, as depicted in Figure 2(c). As in the bidimensional case, the strategy used for subscription propagation ensures that events follow the minimal route. These concepts are easily generalized to spaces with $d > 3$.

More dimensions = More Degrees of Freedom. Interestingly, hyperspaces with $d > 2$ provide alternatives for system



(a) Propagation of the first subscription for p from node A.

(b) Propagation of other subscriptions for p from B and C.

(c) Event propagation. The grid shown in the figure is the bottom plane of Figure 1(b).

(d) The shadowed rectangles represent the subtrees $\mathcal{T}_{l_i}$ associated to the nearby links l_i.

Figure 2. Routing in a d-dimensional hyperspace. Straight arrows denote routes, curvy arrows denote message (subscription or event) propagation.

deployment, by changing the size of subscription and event partitions. The choice we show in Figure 1(b), for instance, is likely to minimize the traffic caused by events, at some additional cost for routing subscriptions. However, the inverse choice may be beneficial in a system where subscriptions outnumber events. Mainstream CBR approaches do not provide this flexibility, as they route events and subscriptions on the same overlay, typically a tree. The advantages and tradeoffs of our approach are discussed in more detail in Section V, based on the analytical model of network traffic we introduce next.

III. An Analytical Model of Traffic

We consider a system with n nodes, organized in a hyperspace of d dimensions. We assume that the space is *full*, i.e., there is exactly one node in each point of the hyperspace, whose side contains exactly $\lambda = n^{\frac{1}{d}}$ nodes. Moreover, we assume that the system topology is static. These assumptions are motivated by the need to simplify analytical treatment: their practical impact is discussed and evaluated in Section VI.

We assign $0 \leq d_s \leq d$ dimensions to subscriptions, and $0 \leq d_e \leq d$ to events. According to the properties in Section II, $d_s + d_e \geq d$ must hold; to minimize traffic, we require that $d_s + d_e = d$. For instance, in Figure 1(b), $d_s = 2$ and $d_e = 1$. We define $S = |\mathcal{S}_i| = n^{\frac{d_s}{d}}$ and $E = |\mathcal{E}_i| = n^{\frac{d_e}{d}}$ as the number of nodes in a subscription and event partition, respectively. Interestingly, we observe that a d-dimensional space contains E distinct subscription partitions $\mathcal{S}_i$, and similarly S distinct $\mathcal{E}_i$ partitions. For instance, in our example there are $S = n^{\frac{2}{3}}$ vertical lines, (event partitions $\mathcal{E}_i$) and $E = n^{\frac{1}{3}}$ horizontal planes (subscription partitions $\mathcal{S}_i$). Finally, note that, as a consequence, E and S are related by the equation $n = S \times E$.

Any node may act as a publisher and/or subscriber, i.e., publishers and subscribers are uniformly spread in the hyperspace. The traffic depends on parameters characterizing the application profile. We define N_p and N_e as, respectively, the number of distinct patterns and events observed in the system during a given time interval. Moreover, we define $\sigma(p)$ as the probability of a node of being a subscriber for a pattern p, and $\mu(e)$ as the probability of a node to be a receiver for event e. (When unambiguous, we drop the indexes for readability.) The value of μ depends on the number of patterns in the system matching e, as well as on the number σn of dispatchers subscribed to them. However, μ cannot be determined simply as the product of these two parameters, as a dispatcher could be subscribed to multiple patterns matching e, by virtue of content-based matching. Clearly, a receiver is also a subscriber for at least one of the patterns matching e.

In the case of a uniform distribution for patterns, (i.e., each pattern p occurs with the same probability), $\sigma(p) = k_p$, where $0 \leq k_p \leq 1$ is some constant. In general, however, different

patterns have a different probability. For instance, in the case of the commonly used Zipf distribution:

$$\sigma(p) = P\frac{c}{(p^\alpha)} \tag{1}$$

being P the average number of subscribed patterns per node, α the exponent of the Zipf distribution, and c a parameter such that $\sum_{p=1}^{N_p} \frac{c}{p^\alpha} = 1$. Similar considerations hold for events.

A. Cost of Routing a Single Pattern p

We can estimate the routing cost to disseminate subscriptions for a pattern p by counting the number of routes in $\mathcal{S}_i$, represented by arrows in Figure 1. Each link can be part of at most two routes, one towards the first subscriber (i.e., the root of the subscription tree in $\mathcal{S}_i$) and one towards the other subscribers. For instance, in Figure 2(d) link l_3 contains two routes, one towards A (the root) and the other towards C. Instead, l_1 and l_2 both contain only one route because they are used only to reach the root subscriber A.

We denote with $\rho_l(p)$ and $\eta_l(p)$ the probability of a link l to be part of a route towards the root and other subscribers, respectively. Hereafter, to simplify the model description, we assume[2] that the first subscriber (the root) is always located in the center of $\mathcal{S}_i$. Therefore, the resulting tree is symmetric along all dimensions, and propagation of the first subscription always begins from a line's center. Based on these observations, the number of routes on a single line can be easily computed[3] as $2\sum_{l=1}^{\frac{\lambda-1}{2}} \rho_l(p) + \eta_l(p)$.

This result allows us to compute the traffic $T_{\mathcal{S}_i}$ generated in a single partition $\mathcal{S}_i$ by considering all lines on all dimensions:

$$T_{\mathcal{S}_i}(p) = \sum_{k=1}^{d_s}(L_k \times 2\sum_{l=1}^{\frac{\lambda-1}{2}}(\rho_l(p) + \eta_l(p))) \tag{2}$$

In this expression, the coefficient L_k takes into account that the total number of available lines varies according to the dimension k, $1 \leq k \leq d_s$. For instance, consider the tree in Figure 2(a). Here, the first dimension (rows) contains only one line, while the second (columns) contains $\lambda = 7$ lines. If we were to set $d_s = 3$, the tree would include the $\lambda^2 = 49$ lines realizing the third dimension by intersecting the plane. It is therefore easy to see that $L_k = \lambda^{k-1}$.

To compute the two probabilities ρ_l and η_l, we observe that a link l contains a route between a father u and a child v if and only if the set $\mathcal{T}_l$ of nodes belonging to the sub-tree rooted at v contains *at least* one subscriber. Similarly, a route from v to u exists if and only if the set $\mathcal{S}_i \setminus \mathcal{T}_l$ of nodes *not* belonging to the sub-tree rooted at v contains *at least* one subscriber. For instance, link l_3 in Figure 2(d) has two routes, since there is a subscriber both in $\mathcal{T}_{l_3}$ and outside of it.

Recall that σ is the probability that a node is a subscriber for a pattern p, and $\bar{\sigma} = 1 - \sigma$ the complementary probability, i.e., the probability of *not* being a subscriber for p. Therefore, $\bar{\sigma}^{|\mathcal{T}_l|}$ is the probability that no subscriber exists in l's subtree and η_l is simply its complementary probability, i.e., the probability that there exists at least one subscriber in l's sub-tree:

$$\eta_l = 1 - \bar{\sigma}^{|\mathcal{T}_l|} \tag{3}$$

Along the same lines we can derive the expression for ρ_l as the probability that at least one subscriber exists outside the l's subtree, i.e. among the $S - |\mathcal{T}_l|$ nodes outside $\mathcal{T}_l$:

$$\rho_l = 1 - \bar{\sigma}^{S-|\mathcal{T}_l|} \tag{4}$$

We observe that $|\mathcal{T}_l|$ depends on the dimension k on which l is located and its position i, $1 \leq i \leq \lambda$, inside such dimension. Consider Figure 2(d), and assume that positions inside lines are computed starting from the lower left corner of the plane. On the first dimension (rows), $\mathcal{T}_{l_2}$ contains 14 nodes, i.e., l_2's position ($i = 2$) times the size of the second dimension ($\lambda = 7$). Instead, $|\mathcal{T}_{l_1}| = 7$, since the position of l_1 is $i = 1$. Finally, $|\mathcal{T}_{l_3}| = 3$ is determined by l_3's position ($i = 3$) along the vertical dimension. Generalizing, $|\mathcal{T}_l| = i\lambda^{d_s-k}$.

Finally, to obtain the overall traffic generated by the subscription messages for a single pattern p, we need to multiply $T_{\mathcal{S}_i}(p)$ by the number of distinct partitions $\mathcal{S}_i$. This, as observed earlier, is equal to $E = |\mathcal{E}_j|$, since by construction there is one and only one partition $\mathcal{S}_i$ per each node in $\mathcal{E}_j$. Therefore, recalling Equation (2), the expression of the total traffic generated by subscriptions for a given pattern p is:

$$T(p) = ET_{\mathcal{S}_i}(p) = 2E\sum_{k=1}^{d_s}(\lambda^{k-1}\sum_{i=1}^{\frac{\lambda-1}{2}}(2 - \bar{\sigma}^{|\mathcal{T}_l|} - \bar{\sigma}^{S-|\mathcal{T}_l|})) \tag{5}$$

where $S = |\mathcal{S}_i| = \lambda^{d_s}$ and $|\mathcal{T}_l| = i\lambda^{d_s-k}$. The equation above correctly yields $T(p) = 2E(S-1)$ (i.e., all links in all $\mathcal{S}_i$ are bidirectional) when $\sigma = 1$ (i.e., all nodes are subscribers).

B. Cost of Routing a Single Event e

Similar to subscriptions, we focus on the dissemination of a single event. First of all, we observe that the event traffic is determined by a fixed contribution $T_{\mathcal{E}_i}$, given by the event dissemination inside the partition $\mathcal{E}_i$ containing the publisher, plus all the contributions $T_{\mathcal{S}_j}(e)$ due to event forwarding outside $\mathcal{E}_i$ based on the routing information in the $\mathcal{S}_j$ subscription partitions crossed by $\mathcal{E}_i$. Therefore:

$$T(e) = T_{\mathcal{E}_i}(e) + ET_{\mathcal{S}_j}(e) \tag{6}$$

The fixed contribution is simply $T_{\mathcal{E}_i}(e) = |\mathcal{E}_i| - 1 = E - 1$. To estimate the additional traffic $T_{\mathcal{S}_j}$ we observe that, by construction, each node in $\mathcal{E}_i$ belongs also to a partition $\mathcal{S}_j$. Therefore, if there is at least a subscriber in $\mathcal{S}_j$, the nodes lying in the intersection between $\mathcal{E}_i$ and $\mathcal{S}_j$ can be regarded as a sort of *virtual publisher* in $\mathcal{S}_j$. In other words, these are the nodes that begin the dissemination of the event inside $\mathcal{S}_j$, according to the routes established by its subscribers. Figure 2(c) illustrates the concept pictorially.

We observe that a link l is traversed by an event e if and only if the sub-tree $\mathcal{T}_l$ contains at least one receiver and the

[2]This assumption can be easily implemented by defining the underlying overlay network to be a torus, and having the subscriber propagate its subscription on the d_s dimensions only for $\frac{\lambda}{2}$ hops along each direction.

[3]If λ is even, the number of routes is actually $(2\sum_{l=1}^{\frac{\lambda}{2}} \rho_l(p) + \eta_l(p)) - 1$. Hereafter, for the sake of readability we assume that λ is odd.

virtual publisher of e is not in $\mathcal{T}_l$ or, vice versa, the virtual publisher is in $\mathcal{T}_l$ and there is at least one receiver outside $\mathcal{T}_l$.

We define the probability $\pi_l = \frac{|\mathcal{T}_l|}{n}$ as the probability that the virtual publisher lies in $\mathcal{T}_l$, and $\bar{\pi}_l = 1 - \pi_l$ as the probability that it lies outside $\mathcal{T}_l$. Therefore, the probability $\psi_l(e)$ that a link at level l is traversed by an event e is

$$\psi_l(e) = \pi_l(1 - \bar{\mu}^{S-|\mathcal{T}_l|}) + \bar{\pi}_l(1 - \bar{\mu}^{|\mathcal{T}_l|}) \tag{7}$$

recalling that $|\mathcal{T}_l| = i\lambda^{d_s-k}$ and $S = \lambda^{d_s}$.

We compute the number of links traversed by e analogously to Section III-A. Equation (2) can be "reused" by noting that an event can traverse a link only in one direction, and therefore ρ_l and η_l are replaced by the probability $\psi_l(e)$:

$$T_{\mathcal{S}_j}(e) \quad = \quad \sum_{k=1}^{d_s}(L_k \times 2\sum_{i=1}^{\frac{\lambda-1}{2}}\psi_l(e)) \tag{8}$$

Recalling Equation (6), the overall traffic generated by e is:

$$T(e) = E{-}1{+}2E\sum_{k=1}^{d_s}(\lambda^{k-1}\sum_{i=1}^{\frac{\lambda-1}{2}}\pi_l(1{-}\bar{\mu}^{S-|\mathcal{T}_l|}){+}\bar{\pi}_l(1{-}\bar{\mu}^{|\mathcal{T}_l|})) \tag{9}$$

The equation above is easily verified for $\mu = 1$ (i.e., all nodes in the system are receivers), correctly yielding $T(e) = n - 1$.

C. Total Message Traffic

With a uniform distribution of patterns and events, the total traffic is simply $T = N_pT(p) + N_eT(e)$. Non-uniform distributions, such as the Zipf one in Equation (1) force us to take into account different values of σ and μ for each pattern and event. Therefore, in general

$$T = \sum_{p=1}^{N_p} T(p) + \sum_{e=1}^{N_e} T(e) \tag{10}$$

IV. Validating the Model

We validated our model using PeerSim [8], a discrete event simulator expressly designed for large-scale distributed systems. We feed both the model and the simulator the same scenario parameters, whose default values are shown in Table I, and compute the (percentage) difference between the traffic derived analytically and through simulation, defined as $\%\Delta T = 100 \cdot (T_{model} - T_{sim})/T_{sim}$. In both cases, we use a full multidimensional space, as this is not only the assumption we used to derive our model, but also the worst case in terms of traffic generated, as shown in Section VI-A. Results are averaged over 50 simulation runs with different seeds.

Table II shows the results w.r.t. an increasing system size n. In Table II(a) we kept the space dimension fixed to $d = 4$, therefore causing a corresponding increase in λ, the side of the hyperspace. Dually, in Table II(b) we fixed $\lambda = 10$ as we incremented the number of nodes from 100 to 100,000 (the highest scale we could simulate), increasing d accordingly from 2 to 5. The values of n differ in the two tables, to maintain a full hyperspace. Also, to avoid a bias due to the fixed component $T_{\mathcal{E}_i}$ of event traffic, we set $d_s = d$.

Parameter	Default value
n	10000
$\sigma(p)$	2%
$\mu(e)$	10%
d	4
d_s	4
λ	10

Table I
DEFAULT PARAMETERS USED IN THE VALIDATION.

	n=256 ($\lambda = 4$)	n=4096 ($\lambda = 8$)	n=20736 ($\lambda = 12$)	n=65536 ($\lambda = 16$)
$\%\Delta T(p)$	0.82	0.03	0.02	0.02
$\%\Delta T(e)$	0.37	0.14	0.12	0.19

(a) d constant.

	n=100 ($d = 2$)	n=1000 ($d = 3$)	n=10000 ($d = 4$)	n=100000 ($d = 5$)
$\%\Delta T(p)$	1.04	0.28	0.39	0.18
$\%\Delta T(e)$	-0.44	1.66	0.91	0.34

(b) λ constant.

Table II
SIMULATED VS. THEORETICAL TRAFFIC W.R.T. THE SYSTEM SIZE n.

d_s	1	2	3	4
$\%\Delta T(p)$	1.74	0.36	0.20	0.39
$\%\Delta T(e)$	4.26	-0.078	0.93	0.91

Table III
SIMULATED VS. THEORETICAL TRAFFIC W.R.T. d_s.

Both experiments demonstrate that our model approximates very precisely the values generated by the simulator. Indeed, the highest difference ΔT between theoretical and simulated traffic is 1.66%. Moreover, as expected the model becomes more accurate as n increases, since the distribution of subscribers and publishers in space is increasingly approximated by a uniform one. Also, we note that in almost all cases our model yields a conservative estimate by excess.

In Table III, we remove the assumption $d_s = d$ and show the results of our comparison for increasing values of d_s, therefore validating our model against different choices of $\mathcal{S}_i$ and $\mathcal{E}_i$. Again, our model provides a very precise estimate of the traffic generated. In general, the difference is below 1%. The only exception, still below 4.5%, is for $d_s = 1$. The reason lies again in our assumption about uniform distribution, which in this case is challenged by the small size of λ.

Finally, Table IV assesses the impact of the distribution of patterns and events. To enable a fair comparison and remove bias, we set $\sigma = \mu$, essentially assuming that all subscriptions match only one event. This way, we can easily evaluate $T(p)$ against σ and $T(e)$ against μ. Table IV(a) shows the case of a uniform distribution of patterns, while Table IV(b) uses a Zipf distribution. Once more, our model is very precise, with a difference w.r.t. simulated results below 3%, and in most cases below 1%. For what concerns Table IV(b), it is worth noting that the best results are obtained for $\alpha \leq 1$, which represents the values commonly found in literature.

We can conclude that our model provides a very good ap-

σ, μ	0.001	0.01	0.1	0.5	0.8	0.9
$\%\Delta T(p)$	0.003	0.11	0.68	0.67	0.16	0.02
$\%\Delta T(e)$	1.87	1.92	2.97	1.50	0.34	0.05

(a) Uniform distribution.

α	2	1	0.5
$\%\Delta T(p)$	-0.67	-0.03	-0.01
$\%\Delta T(e)$	-2.67	-0.72	-0.51

(b) Zipf distribution.

Table IV
SIMULATED VS. THEORETICAL TRAFFIC W.R.T. THE DISTRIBUTION OF PATTERNS AND EVENTS. WE ASSUME $\sigma(p) = \mu(e)$.

proximation of the results available through simulation, which is commonly used to evaluate CBR protocols. Moreover, our results evidence that the precision of our model improves as the system size increases. We exploit this fact in the next section, by comparing HyperCBR against mainstream approaches in networks whose scale could not be easily simulated.

V. EVALUATION

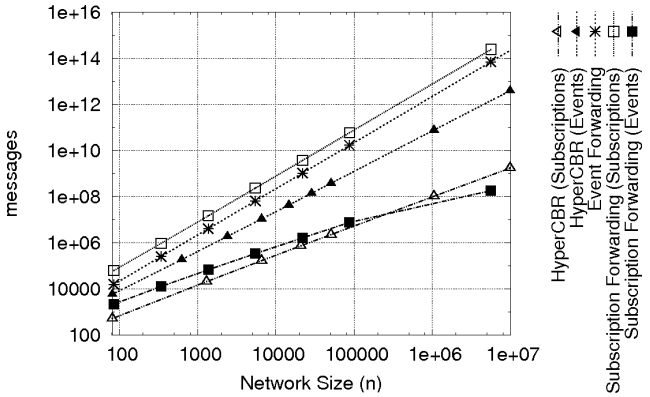
We exploit the model defined in Section III to evaluate the performance of routing in HyperCBR. We compare against Siena [5], since it is probably the best known CBR system, and is paradigmatic of the design choices of many others. Siena assumes the existence of an overlay network connecting all nodes in a single unrooted tree and uses the subscription forwarding routing strategy mentioned earlier. The comparison between tree-based subscription forwarding and HyperCBR relies on a model of the former we recently published [9].

Moreover, we consider also *event forwarding* where, dually to subscription forwarding, events are always broadcast to all brokers and matched locally against subscriptions, which instead are never propagated. Notably, in HyperCBR $\langle d_e = 0, d_s = d\rangle$ yields subscription forwarding (albeit implemented in a d-dimensional space), while $\langle d_e = d, d_s = 0\rangle$ yields event forwarding. It is fair to say that our approach subsumes the two above, arguably the most common in the literature.

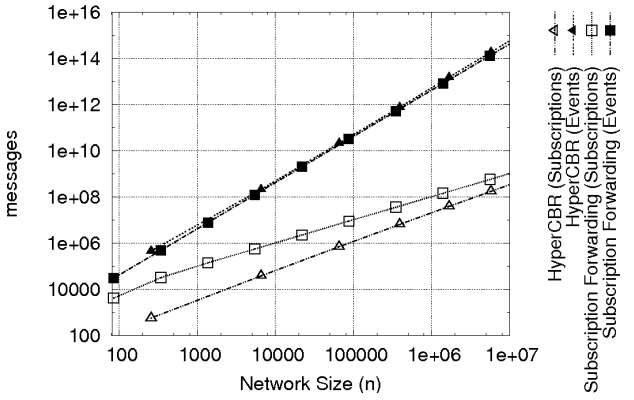
We evaluate HyperCBR in two scenarios: a *file sharing* application and a *publish-subscribe* system. These represent two extremes in terms of traffic, as the former is characterized by a huge number of subscriptions while the latter is dominated by events. This shows the flexibility of HyperCBR that, due to its ability to privilege events or subscriptions, blends well with many different workloads. Moreover, as we show later in this section, HyperCBR not only reduces the network traffic but also achieves a better balancing of the (computational, network, and memory) load, through higher decentralization.

File Sharing. As suggested in [3], CBR can be used to provide scalable data search and retrieval in a peer-to-peer network. Nodes can advertise their shared resources (e.g., MP3 files) by means of subscriptions and retrieve data from others by injecting in the network an event (query) containing the pattern (e.g, a regular expression) to match intended items.

The parameters we used for characterizing this scenario, shown in Figure 3(a), are derived from traces collected on



(a) File sharing ($N_p = 8n$, $N_e = 2.25n$, $P = 3.2$, $\alpha = 0.4$).



(b) Publish-subscribe ($N_p = 100$, $N_e = 20n$, $P = 0.75$, $\sigma = 0.75\%$, $\mu = 10\%$).

Figure 3. Comparing HyperCBR and subscription forwarding.

an EDonkey server over 24 hours [10]. σ is computed using Equation (1). We set $\mu = \sigma$, as in EDonkey downloaded files are always shared, thus the popularity of downloaded and shared file is essentially the same. Finally, we set $d = 4$.

In this scenario, subscriptions are the main traffic contributors. Therefore, to reduce the overall traffic, we chose $d_s = 1$, i.e., subscriptions are propagated only along a line. Subscription forwarding, instead, must propagate all the subscriptions to all nodes, therefore dramatically increasing traffic. This yields the situation depicted in Figure 3(a) where, on a scale of $n = 10^7$, the subscription traffic of subscription forwarding is more than two order of magnitude higher than our approach (note the logarithmic scale on the y-axis). Event traffic is indeed dwarfed by subscriptions. Moreover, in this chart we do not consider unsubscriptions: if we did, our performance would be even better, since unsubscriptions propagate basically like subscriptions.

As another point of comparison, we plot the performance of event forwarding, adopted by some file sharing applications, notably Gnutella [7]. Since no subscriptions are propagated, the only contribution comes from event (i.e., query) traffic. However, as the chart shows, the traffic due to events is very high since each event must reach all nodes in the network.

We note that subscription forwarding has no way to affect

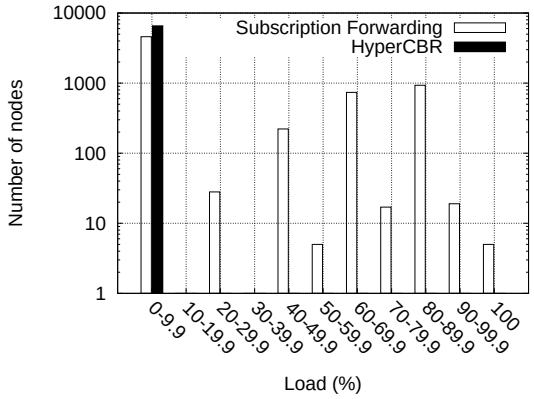
the subscription traffic. Indeed, the only "knob" available in subscription forwarding is provided by the tree degree, set to $f = 4$ in this scenario. Nevertheless, increasing (or decreasing) f bears no significant effect, since the vast majority of subscriptions need to be forwarded to all nodes, regardless of the value of f. Similar arguments hold for event forwarding. HyperCBR outperforms both precisely because, besides changing the degree by varying d, it can also redistribute the load between subscriptions and events by tuning d_s properly.

Publish-Subscribe. There is no established and commonly agreed benchmark scenario for content-based publish-subscribe—let apart when the scale is very high as in our case. Therefore, we borrowed some parameters mentioned in the literature [11] and set others to reasonable values, as shown in Figure 3(b). In particular, we set event messages to be 20 times more frequent than subscriptions. This is an extreme scenario particularly suitable for subscription forwarding, as this strategy is very effective in optimizing event traffic—albeit with the side-effect of increasing the number of subscriptions. In principle, we can set $d_s = d$ and achieve essentially the same performance of subscription forwarding, again reasserting the versatility of our approach even in unfavorable situations. Instead, Figure 3(b) shows the results obtained with $d_s = 2$. Although this choice has not a relevant impact on traffic, it significantly reduces the load on the nodes, as discussed next.

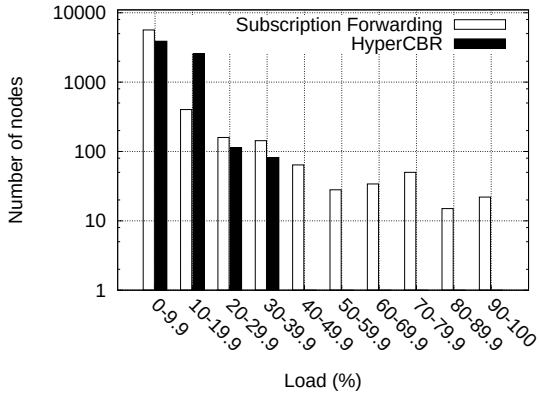
Node Load. The emphasis of this paper is on network traffic. Nevertheless, our approach also drastically reduces the load on the individual nodes—another key aspect of scalability. Indeed, as an extreme, centralized solutions often ensure lower traffic than decentralized ones, but at the expense of concentrating the entire load on one node.

Since we currently do not capture this aspect in our model, we further investigate it by means of simulation. We use the aforementioned publish-subscribe scenario with 6,500 nodes, and evaluate the load in terms of messages dispatched by each node. As shown in Figure 4(a) and 4(b), in HyperCBR the load is equally distributed among nodes, while in subscription forwarding the nodes at the highest levels of tree must handle most of the messages. This is also evident in Figure 4(b) where we plot the distribution of events: in subscription forwarding a subset of nodes "sees" *all* the events published in the systems while in multidimensional a node *at most* forwards 32% of them. This means that when the scale of the system, and accordingly the number of events, is increased, subscription forwarding is likely to experience serious scalability issues as some nodes become bottlenecks. Conversely, HyperCBR exhibits very good scalability because event propagation involves each time a different set of nodes according to the partition $\mathcal{E}_i$ of the publisher. Note how these remarkable results are derived in a scenario where the traffic costs being compared were similar. In the file sharing scenario HyperCBR works even better, by reducing also the absolute value of traffic.

Besides reducing the number of messages handled per node, HyperCBR improves also w.r.t. the memory requirements for subscription tables and the computational overhead caused by



(a) Subscriptions.



(b) Events.

Figure 4. Distribution of the subscription and events messages dispatched by each node in the publish-subscribe scenario.

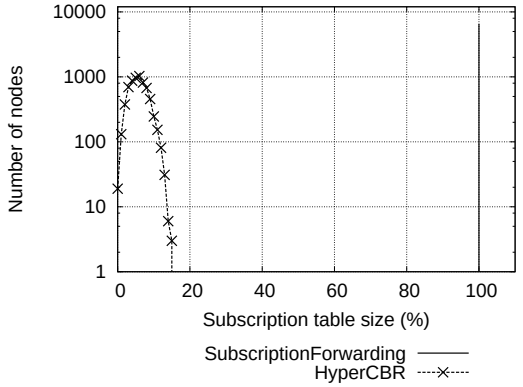


Figure 5. Subscription table size.

their inspection during event matching. Indeed, subscription forwarding requires each subscriber to know about each of the N_p distinct patterns in the system. In our protocol, instead, a node knows only the patterns in the partition $\mathcal{S}_i$ it belongs to. Therefore, the size of its subscription table, and consequently the computational overhead to inspect it, are sensibly lower. The difference between the two approaches in terms of memory requirements can be appreciated in Figure 5, where we report the distribution of table sizes in the simulated scenario

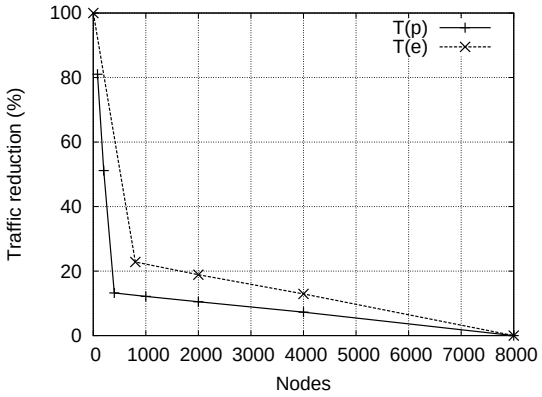


Figure 6. Traffic reduction w.r.t. a full hyperspace.

described above. The chart confirms our expectations, showing that the subscription tables in HyperCBR are a remarkable 80% smaller than in subscription forwarding.

These results confirm that HyperCBR is well-suited to the large scale scenarios we target, and a good choice for supporting CBR in a wide range of application domains.

VI. Dealing with Sparse Networks and Churn

In Section III we assumed that the hyperspace is full, i.e., each point in space is always taken by a node. In reality, this is not always the case. More generally, implementing HyperCBR entails building an overlay network supporting the multidimensional grid we base our routing upon, and maintaining it in the face of nodes joining and leaving. In the following, we discuss and evaluate our current solutions, which rely on the foundation provided by CAN [6].

A. Sparsely Populated Networks

In CAN, each node is assigned a *zone* that spans several points. We use the term *point* to indicate a position in the multidimensional space, and *node* to refer to the real host. A node is therefore responsible for many points. Moreover, a node keeps track of the IP address of its neighbors (i.e., of the owners of adjacent zones) as well as of their "direction" in space (e.g., whether they are on the same horizontal plane or on the same vertical line).

HyperCBR selects one of the points assigned to a node as its *identity*, i.e. the point which issues subscriptions and receives matching events on behalf of the node. Routing of events and subscriptions occurs as described in Section II, by relying on the topology information above. The only (significant) difference is that a message exchanged between two points owned by the same node does not generate any traffic. For instance, Figure 7(a) shows our reference example with the addition of the nodes responsible for points, denoted by boxes. An event published in A is delivered to the three subscribers based on the existing routes. However, only 4 network messages are exchanged across the nodes involved instead of the 8 messages that would be required among points: the remaining hops are "performed" locally to nodes.

Evaluation. Interestingly, in practical cases where the hyperspace is not full, the model we derived in Section III provides an upper bound for traffic. In Figure 6, we analyze the traffic (events and subscriptions) generated by the publish-subscribe scenario of Section V in a hyperspace of 8,000 points. The plot shows the percentage traffic reduction generated by a non-full hyperspace, in comparison with a situation where every point is associated to a single node. To enable fair comparison, we maintain the number of subscribers and receivers constant across the different sizes.

The x-axis shows the number of nodes in the system. The case $n = 8000$ corresponds to a full space and no reduction is possible. At the other extreme, a 100% reduction is observed when only one node exists, as all points are assigned to it and no traffic is generated. In intermediate situations, a traffic reduction is always observed (e.g., about 10% with $n = 4000$).

B. Churn and Route Reconfiguration

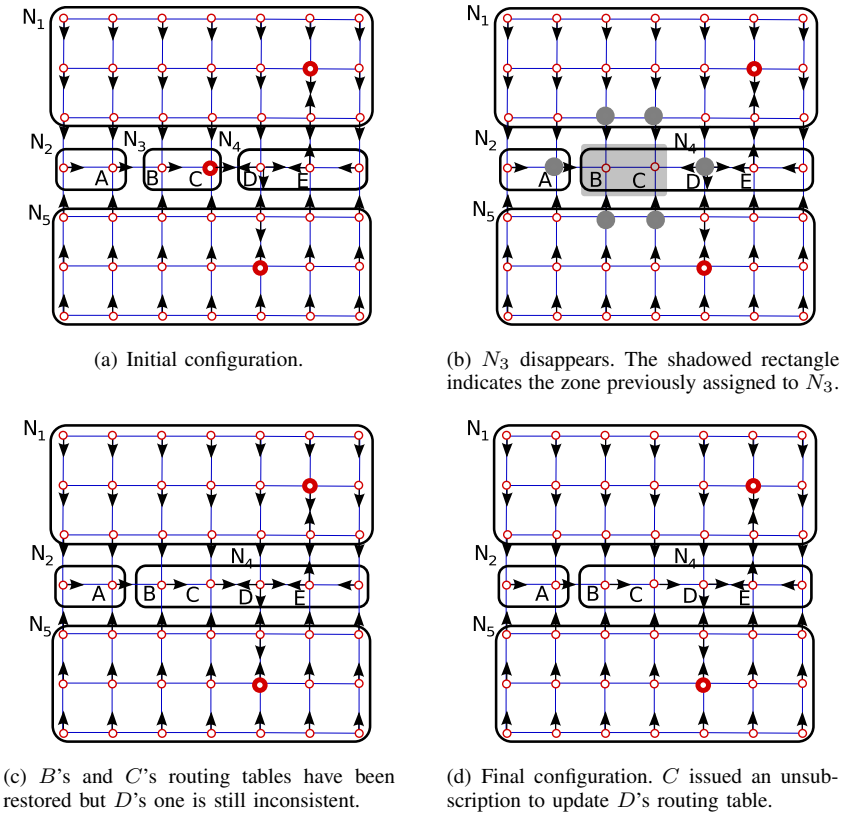
In CAN, a node joins by picking a random point in space, routing to the zone containing the point, and splitting the zone with its current owners. A node leaving is dealt with by having the owner of one of the neighboring zones take over the one owned by the departing node. However, reconfigurations must be handled not only w.r.t. the overlay network, but also w.r.t. routing, by properly reconciling routes with network changes.

Join. When a node N_{new} joins the network it become responsible of a zone $\mathcal{Z}$, previously assigned to another node, say N_{old}. In this case, the route reconfiguration process is rather simple, and consists of N_{old} forwarding to N_{new} all the information concerning the points in $\mathcal{Z}$. N_{new} chooses one of these points as its identity, and starts disseminating its subscriptions. In case N_{old}'s identity is among these points, N_{old} selects one of its current points as its new identity and unsubscribes from the previous one.

Leave. Soft failures can be dealt with analogously to joins. Before disconnecting, a node can notify the one taking over its zone and forward the routing tables of its points. Hard failures are more complex because when a node disconnects abruptly the routing table of its points are lost. To explain how they are recovered we refer to the example in Figure 7. In Figure 7(b), N_3 disappears abruptly and its zone, depicted by a shadowed area, is assigned to N_4. However, the routing tables associated to points B and C are lost and must be recovered.

Routing information is reconciled by having the new zone owner contact[4] the points neighboring along the d_s dimensions with the zone originally associated with the departed node, and determining whether their subscriptions require the insertion of a local one. This occurs when the neighboring points hold subscriptions that point "outwards" w.r.t. the vanished zone (i.e., B and C). In the example, the points being contacted are those marked in gray. Point A has only one subscription pointing towards the vanished zone (specifically, towards B), therefore no new route is required. Conversely, D holds a

[4]We use the term *contact* to indicate both *remote* (points belonging to different nodes) and *local* (points belonging to the same node) interactions.



(a) Initial configuration.

(b) N_3 disappears. The shadowed rectangle indicates the zone previously assigned to N_3.

(c) B's and C's routing tables have been restored but D's one is still inconsistent.

(d) Final configuration. C issued an unsubscription to update D's routing table.

Figure 7. Reconfiguration example ($d_s = 2$). Capital letters indicate *points* while N_i denote *nodes*. The rectangles represent the set of points assigned to each node. Subscribers are identified by double circles.

subscription towards E, pointing away from the vanished zone (i.e., from C). Therefore, a subscription must be added to reconnect the path followed by events. This subscription is then propagated in the usual way to the other points in the zone (only B in the example), leading to the situation in Figure 7(c).

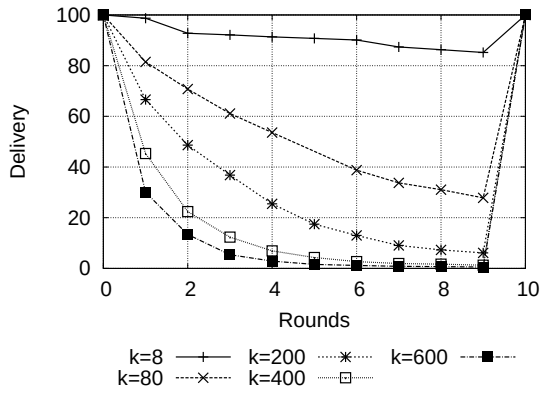
This configuration, however, is still inconsistent because D has a subscription towards C that is now obsolete. Nevertheless, this situation can be easily detected by scanning all points, including neighbors, to see whether there are some subscriptions that enter the vanished zone but never exit. In our example, the subscription from D to C does not have a complementary one from B to A and must be removed. This is accomplished by issuing an unsubscription from C, bringing the system in the final, correct configuration of Figure 7(d).

Evaluation. To assess the effectiveness of our reconfiguration solution, we stress HyperCBR by removing nodes from the network at regular intervals, measuring event delivery (i.e., the ratio between the events actually delivered and those expected) during and after the reconfiguration. We consider a network with $n = 8000$ and $d = 3$ and use the workload of the publish-subscribe scenario described in Section V. At regular intervals (rounds) we remove k nodes from the system, with k varying from a minimum of 8 to a maximum of 600 nodes per round. This yields an overall failure ranging from 1% to 75% of the nodes. Note how nodes are removed also during the recovery process, possibly creating overlapping reconfigurations. Figure 8(a) shows the event delivery. As expected, delivery drops while nodes are being removed, due to misconfigured routing tables. However, as soon as we stop removing nodes delivery goes back to 100%, confirming that the protocol returns to a correct configuration.

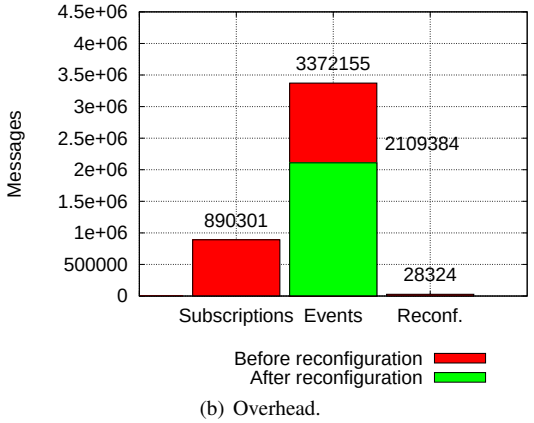
Figure 8(b) shows the cost of reconfiguration in terms of messages exchanged to update the routing tables, for $k = 40$. The chart confirms that the reconfiguration traffic is largely negligible w.r.t. the number of messages exchanged to disseminate subscriptions and events. As the chart shows, it is even smaller than the reduction caused by the fact that nodes are removed and the hyperspace becomes emptier, based on the previous considerations. This low overhead stems from the locality of the interactions required to reconstruct the routing tables. As shown in the example, only neighboring nodes are contacted, therefore generating a low traffic.

VII. Related Work

Mainstream tree-based approaches, as discussed in Section V, provide the highest efficiency in event routing but subscription traffic negatively affects their performance. Recent work in the context of publish-subscribe (e.g., [12], [13]) exploit DHTs to support CBR. Unfortunately, they assume that events and subscriptions are based on attributes, thus they do not unleash all the expressive power of arbitrary content-based matching. Moreover, some of these approaches require the



(a) Delivery.



(b) Overhead.

Figure 8. Dealing with churn in the publish-subscribe scenario ($n = 8000$). At each round, k nodes are removed from the system.

existence of a so-called *rendez-vous* node where subscriptions meet events. In HyperCBR, every node can act as a rendez-vous point, therefore enabling higher scalability through a more decentralized scheme and minimizing bottlenecks. Finally, none of the aforementioned approaches offers mechanisms to tune the system towards to different subscription and event workloads, and therefore they cannot be applied in different contexts (e.g., peer-to-peer networks). The only exceptions to this respect are represented by [14], [15], which split the event space on different nodes to redistribute the effort. Compared to HyperCBR, these works achieve flexibility in a more coarse-grained fashion. Also, they rely only on simulation, while our analytical model provides a powerful tool to investigate the tradeoffs involved in very large-scale scenarios that cannot be analyzed through simulation.

In the related field of peer-to-peer networks, several efforts tackled scalable search services. Early solutions exploit either a centralized approach [16], [17] or a flooding-based one [7], [18]. More recent ones (e.g., [19]) exploit DHTs to implement keyword search. Although these protocols are routinely used in most file sharing application, the quality of the search is very poor when compared against content-based systems.

Finally, recently the database community focused on implementing distributed DBMSes in a peer-to-peer fashion, under the name of *semantic overlay networks* [20], [21]. Nevertheless, these approaches suffer from the same issues evidenced above for peer-to-peer and publish-subscribe systems, since they do not support the full power of pattern-based search (e.g., through regular expression) and fields are mapped onto single nodes, thus easily creating bottlenecks.

VIII. Conclusions

In this paper we proposed HyperCBR, a new CBR approach expressly targeting very large-scale scenarios. HyperCBR relies on a multidimensional space where subscriptions and events are routed along distinct, albeit intersecting, partitions. Our routing technique leads to higher scalability: we provided evidence for this claim through simulation and through an analytical model of traffic. Interestingly, we evaluated our approach in paradigmatic scenarios with very different characteristics: indeed, increased flexibility in deployment is another distinctive trait of HyperCBR. Finally, we showed how HyperCBR can be realized on top of CAN, and how it can tolerate high levels of churn with low overhead.

References

[1] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *Computing Surveys*, vol. 2, no. 35, 2003.
[2] A. Bulut, A. K. Singh, and R. Vitenberg, "Distributed data streams indexing using content-based routing paradigm," in *Proc. of* 19^{th} *IEEE Int. Parallel and Distributed Processing Symposium (IPDPS)*, 2005.
[3] D. Heimbigner, "Adapting Publish/Subscribe Middleware to Achieve Gnutella-like Functionality," in *Proc. of SAC*, 2001.
[4] C. Intanagonwiwat *et al.*, "Directed diffusion for wireless sensor networking," *IEEE/ACM Trans. Networking*, vol. 11, no. 1, 2003.
[5] A. Carzaniga, D. Rosenblum, and A. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service," *ACM Trans. on Computer Systems*, vol. 19, no. 3, pp. 332–383, 2001.
[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content addressable network," in *Proc. of SIGCOMM*, 2001.
[7] http://www.the-gdf.org.
[8] http://peersim.sourceforge.net.
[9] S. Castelli, P. Costa, and G. Picco, "Modeling the Communication Costs of Content-based Routing: The Case of Subscription Forwarding," in *Proc. of the* 1^{st} *Int. Conf. on Distributed Event-Based Systems*, 2007.
[10] S. Le-Blond, J.-L. Guillaume, and M. Latapy, "Clustering in P2P Exchanges and Consequences on Performances." in *Proc. of the* 4^{th} *Int. Wkshp. on Peer-to-Peer Systems*, 2005.
[11] A. Carzaniga, M. Rutherford, and A. Wolf, "A routing scheme for content-based networking," in *Proc. of INFOCOM*, 2004.
[12] P. Pietzuch and J. Bacon, "Hermes: A Distributed Event-Based Middleware Architecture," in *Proc. of the* 2^{nd} *Int. Wkshp. on Distributed Event-Based Systems*, 2002.
[13] A. Gupta *et al.*, "Meghdoot: content-based publish/subscribe over P2P networks," in *Proc. of the* 5^{th} *Int. Conf. on Middleware*, 2004.
[14] Y. Wang *et al.*, "Subscription partitioning and routing in content-based publish/subscribe systems," in *Proc. of the* 16^{th} *Symp. on Distributed Computing*, 2002.
[15] F. Cao and J. Singh, "Efficient event routing in content-based publish/subscribe service network." in *Proc. of the* 23^{rd} *INFOCOM*, 2004.
[16] http://www.napster.com.
[17] http://www.edonkey2000.com.
[18] http://www.kazaa.com.
[19] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *Proc. of the* 1^{st} *Int. Wkshp. on Peer-to-Peer Systems*, 2002.
[20] M. Cai and M. Frank, "RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network," in *Proc. of the* 13^{th} *Int. Conf. on World Wide Web*, 2004.
[21] M. Harren *et al.*, "Complex Queries in DHT-based Peer-to-Peer Networks," in *Proc. of the* 1^{th} *Int. Wkshp on Peer-to-Peer Systems*, 2002.