PopSub: Improving Resource Utilization in Distributed Content-based Publish/Subscribe Systems

Pooya Salehi
Middleware Systems Research Group,
Technical University of Munich
salehip@in.tum.de

Kaiwen Zhang
Middleware Systems Research Group,
Technical University of Munich
zhangk@cs.tum.edu

Hans-Arno Jacobsen
Middleware Systems Research Group,
Technical University of Munich
jacobsen@in.tum.de

ABSTRACT
Distributed content-based publish/subscribe systems provide a selective, scalable, and decentralized approach to data dissemination. In a pub/sub overlay network, hop-by-hop routing allows brokers to correctly forward messages without requiring global knowledge. However, this model causes brokers to forward publications without knowing the volume and distance of matching subscribers, which can result in inefficient resource utilization. In order to raise the scalability of pub/sub, we introduce Popularity-Based Publication Routing for Content-based Pub/Sub (PopSub), which is specifically designed to raise the resource utilization efficiency. We define a utilization metric to measure the impact of forwarding a publication on the overall delivery of the system. Furthermore, we propose a new publication routing algorithm that takes into account broker resources and publication popularity among subscribers. Lastly, we propose three approaches to handle unpopular publications. Based on our evaluations, using real-world workloads and traces, PopSub is able to improve resource efficiency of the brokers by up to 62%, and reduce delivery latency by up to 57% under high load.

CCS CONCEPTS
• Software and its engineering → Publish-subscribe / event-based architectures; Client-server architectures;

KEYWORDS
Publish/subscribe, Content-based routing, Tree overlay, Scalability, Performance, Efficient routing, Gossip

ACM Reference format:
DOI: http://dx.doi.org/10.1145/3093742.3093915

1 INTRODUCTION
Publish/subscribe (pub/sub) is widely used as a loosely-coupled, asynchronous, and selective communication substrate between information producers (publishers) and consumers (subscribers) [5, 9, 24]. Clients can subscribe to fine grained data using content-based subscriptions and receive notifications whenever matching publications are disseminated by publishers. In a distributed pub/sub system, the task of matching and forwarding messages to all interested clients are divided and allocated to a network of brokers, collectively called the pub/sub overlay network. In many systems, this overlay is organized as a tree determining the neighborhood relation of brokers and the links that connect them [5, 9, 24]. In order to avoid centralized routing of publications, reverse path forwarding (RPF) is used to establish paths between publishers and subscribers [5].

RPF decouples brokers from the knowledge of all end-to-end paths in the overlay since each broker needs to route messages only to the next hop towards the recipients. This hop-by-hop routing scheme, while scalable since it limits the knowledge required by each broker to operate, conversely hinders their capacity to apply optimizations that rely on global path knowledge. As a consequence, established pub/sub optimizations are primarily local in nature, such as subscription covering [23].

We identify a problem associated with RPF which we call the binary cost model problem. Given an incoming publication to be routed at a broker, this publication must be forwarded to every next hop which contains at least one matching subscription. In other words, the cost to forward a publication to a next hop is the same, no matter the actual number of subscribers downstream.

Hop-by-hop forwarding therefore does not consider the overall system performance. Consider the example in Figure 1. binary cost problem Publisher \( P \), connected to broker \( B_1 \), is sending a publication destined for subscribers \( s_1 \) to \( s_{11} \), all outlined in red. In order to reach all matching subscriptions, broker \( B_1 \) must forward the publication to both broker \( B_2 \) and \( B_3 \). We observe that for the same amount of work, forwarding to \( B_2 \) is used for 10 deliveries, while forwarding to \( B_3 \) produces only one delivery. In addition, sending to \( B_3 \) does not directly reach \( s_{11} \); another forwarding to \( B_4 \) is needed. Therefore, the utility of sending to \( B_2 \) is 20 times greater than sending to \( B_3 \). As pub/sub systems are designed to sustain high publication throughput [30], prioritizing publication forwarding with high utility improves resource utilization in the overlay and the efficiency of the system as a whole.

From the point of view of a broker, forwarding efficiency is then dependent upon two factors: distance to reach subscribers and number of subscribers (i.e., popularity). On each broker, we want to prioritize forwarding publications with high popularity and nearby

Figure 1: Example of binary cost problem
subscribers. In contrast, subscribers which are located far away with unpopular interests, drag down the efficiency of the whole system by attracting publications down a long path of brokers with low utility. Because it only takes one outlying subscriber to match a publication, addressing the binary cost problem will raise the scalability of the system, even under the presence of a small number of disorderly subscriptions.

The typical solution to raise system efficiency is to construct optimal overlays [7]. While this approach works for topic-based systems, it is not adequate to capture the complex relationships between content-based subscriptions, nor is it designed to create tree topologies, which are the focus of our work.

In this paper, we present Popularity-Based Publication Routing for Content-based Pub/Sub (PopSub). PopSub seeks to address the shortcomings of hop-by-hop forwarding by considering its utility on the end-to-end paths, while retaining the desired property of knowledge decentralization. PopSub measures the impact of one publication forwarding in terms of the volume and distance of subscriptions that can be satisfied. Publications are then prioritized based on this popularity-based metric. Publication popularity estimation is performed in a scalable way which does not require any change to the RPF, nor global knowledge of all paths. Furthermore, it is a lightweight mechanism which does not require overlay reconfiguration. We propose three alternatives to the main dissemination tree, namely Direct Delivery, Batching, and Gossiping, to deliver less popular publications to improve system performance. The contributions of this paper are as follows:

1. Identify a metric which measures the gain of forwarding a publication based on distance and popularity of downstream subscribers.

2. Provide a lightweight and scalable way to estimate the popularity of publications among subscribers at each broker without requiring global knowledge and by piggybacking on existing pub/sub traffic.

3. Propose three alternative dissemination mechanisms for handling low priority publications: Direct Delivery, Batching, and Gossiping.

4. Evaluate PopSub and demonstrate its scalability through the improved efficiency of the overlay network, and show the relative benefits of using the proposed alternative dissemination mechanisms.

Next, Section 2 provides a review of the related work. Section 3 gives an overview of the system model, defines the binary cost model for routing, and describes the identified flaw innate to pub/sub systems. Section 4 describes our solution, which includes metric collection, publication prioritization, alternative dissemination mechanisms, and traffic handover. The results of our evaluation are presented in Section 5. We finally conclude in Section 6.

2 RELATED WORK

In this section, we discuss the related work in the following categories: self-organizing overlays and overlay reconfiguration, efficient publication routing, and QoS-aware pub/sub.

In managed overlays, the topology is designed for efficient publication routing and high throughput. In contrast, self-organizing overlays rely on gradually constructing and periodically reconfiguring the overlay to achieve these properties. Several existing works use self-organizing overlays to avoid costly publication forwarding by identifying middle steps which are pure forwards (brokers with no local interest) [1, 6, 8]. Scribe is a DHT-based peer-to-peer multicasting system which can route publications via interested peers and thereby avoid unnecessary publication forwards. Spidercast [8] relies on its own protocol to construct an efficient overlay for topic-based dissemination. These works only target multicast or topic-based systems with self-organizing overlays, whereas PopSub addresses content-based pub/sub using managed overlays.

Some studies suggest overlay reconfiguration to improve publication delivery efficiency in content-based systems with managed overlays [2, 29]. In these approaches, brokers monitor and compare their subscriptions: brokers with similar interests are then connected together to reduce number of publication forwarding operations required to deliver the publications. PopSub differs from self-organizing and reconfiguration works in three ways. First, performing these reconfigurations is costly as they require to buffer publications and stop routing while paths are updated [31]. Therefore, overlay reconfiguration temporarily increases publication delivery latency [29] which may not be tolerable in all use cases. Furthermore, overlay construction and transformation may also not be achievable when the network is geographically-aware [19], or when the subscription churn is high [16]. Secondly, overlay reconfiguration is optimized for the benefits of large clusters of similar subscriptions without considering isolated subscriptions. For example, in Figure 2, since majority of subscriptions on B1 are black, clustering brokers and recon figuring the overlay accordingly is more likely to benefit black subscriptions. Consequently, assuming a uniform distribution of subscriptions on brokers, there will always be isolated subscriptions. PopSub on the other hand, can break down subscriptions on each broker and use different dissemination mechanisms for a subset of a broker's subscriptions. In other words, while self-organizing and reconfiguration can benefit subscribers of popular contents, PopSub is still applicable in such systems to identify unpopular content and improve the overall publication routing cost even more.

Lastly, existing works can only improve routing efficiency by avoiding pure forwarding brokers. In scenarios where middle forwarding brokers have only one subscription matching the routing publication, they cannot improve the routing efficiency and therefore suffer from the binary cost model problem (Section 3.3).

Opportunistic multipath forwarding (OMF) [18] improves publication routing efficiency in content-based pub/sub systems by building and maintaining extra links on top of the managed overlay topology. These links are used to bypass pure forwarding brokers and improve delivery latency and throughput. For example, in Figure 2, using OMF, B1 should forward publications directly to brokers B4, B8 and B9, since B2, B3, and B7 have no local black, red or blue subscriptions. However, this increases number of publication forwards that B1 needs to perform in order for all subscribers to receive their matching publications. In topologies with a higher fanout degree and for more popular publications, this can result in a high number of forwards for B1. To avoid overwhelming forwarding brokers, OMF performs these opportunistic bypasses.
considering the free capacity available at the forwarding broker. For brokers which are utilized more than a certain threshold, OMF reverts back to forwarding publications only via the tree, assuming the managed overlay topology is designed to tolerate such loads. In comparison, PopSub considers the overall performance using the publication gain metric and can improve publication delivery even under high loads.

Atmosphere [15] identifies situations where clients suffer from several intermediate forwarding hops due to the present overlay topology. For each publisher, Atmosphere identifies the most relevant subscribers to be directly served by the publisher, forming overlays. Therefore, Atmosphere allows faster publication delivery to subscriptions served by direct links. In contrast, PopSub does not involve changing client connections and does not rely on client resources to improve delivery latency. While Atmosphere provides faster than overlay deliveries, PopSub prevents delayed publications due to congestion and improves overall resource utilization.

Despite the different scenarios addressed by OMF and Atmosphere, the idea of directly delivering publications to improve resource utilization and publication delivery latency is applicable in our popularity-based routing approach as well. Therefore, one of the proposed methods to handle unpopular publications, Direct Delivery, is based on these two works. We evaluate and compare the performance of such a bypassing mechanism on our proposed popularity-based routing scheme.

QoS-aware pub/sub systems, similar to PopSub, aim to augment the knowledge of brokers about existing subscribers without relying on global knowledge and without degrading scalability [3, 4]. Nonetheless, the aim in these systems is to satisfy precise requirements received from clients (QoS guarantees) by estimating and provisioning the resources required. PopSub on the other hand, aims to improve the overall resource utilization of the pub/sub system without involving clients.

There exists a number of works which handle overload situations in highly congested pub/sub systems. These works either throttle the publication rate [17] or use admission control to accept incoming subscriptions [13]. In contrast, PopSub does not assume control over the publisher and subscriber clients. Our work processes all incoming publications and subscriptions. In overload situations, PopSub employs a novel popularity metric to prioritize the appropriate data flows.

3 MODEL AND PROBLEM DEFINITION

In this section, we first provide an overview of the pub/sub model we are employing. We then describe the binary cost model for publication routing and the associated weakness inherent to existing distributed pub/sub systems.

3.1 Terminology

In this paper, we use publisher and subscriber to refer to edge brokers which are responsible for processing and propagating publications and subscriptions of their clients. Therefore, the source and destination of all messages are brokers that are connected to clients which are the actual publishers and subscribers. If necessary, clients are explicitly referred to as publisher client or subscriber client. Furthermore, we refer to the data dissemination acyclic graph connecting all brokers in the overlay as tree.

3.2 Pub/Sub Background

PopSub operates on an advertisement-based content-based pub/sub protocol [5]. In this protocol, publisher clients send advertisements about messages they intend to publish to the broker they are connected to (edge broker). These advertisements are flooded in the overlay. Hence, all brokers are aware of all advertisements in the system. Advertisements have a type (or class) and a set of associated predicates defined over the attributes of the messages that the advertiser intends to publish. During advertisement propagation, each broker stores the advertisement and the identity of the previous broker (called next hop) in the subscription routing table (SRT).

Subscriber clients interested in receiving notifications, send their subscriptions to their edge broker. Similar to advertisements, subscriptions have a type and contain a set of predicates over the attributes of publications they are interested in. Subscriptions are propagated by routing them based on the SRTs. Therefore, each subscription matching an advertisement is propagated towards its advertiser. During subscription propagation, each broker stores the subscription and their next hop in the publication routing table (PRT). Upon receiving a publication, brokers forward it to the next hop of any matching subscription recorded in the PRT. Unsubscribe- tisations and unsubscriptions are the reverse operations used to remove advertisements and subscriptions from the system.

3.3 Binary Cost Model Problem

The goal of publication routing in a pub/sub system is to correctly deliver publications while minimizing routing cost and avoiding
unnecessary communication [27]. In an overlay-based pub/sub system, a publisher \( P \) and a subscriber \( S \) are connected by a tree \( T_{P \to S} = \langle B, L \rangle \) consisting of brokers \( B \) and overlay links \( L \). At each hop, the cost of routing a publication from \( P \) to \( S \) consists of two parts: matching the publication on broker \( B_i \in B \) to find the next hop leading to \( S \), and forwarding the publication across the link \( L_i \in L \) to the next hop. Therefore, the cost of delivering a publication from \( P \) to \( S \) is \( C_{P \to S} = \sum M(B_i) + \sum F(L_i) \) where \( M(B_i) \) is the cost of matching publication \( P \) against existing subscriptions on broker \( B_i \) and \( F(L_i) \) is the cost of forwarding the publication on link \( L_i \) towards \( S \).

This cost model can be generalized to scenarios where a set of subscribers \( S \) are interested in publisher \( P \). The cost of routing a publication via \( T_{P \to S} \) is \( C_{P \to S} = \sum M(B_i) + \sum F(L_i) \), where \( B_1 \) is the set of brokers receiving the publication \( i \) hops away from the publisher and \( L_i \) is the set of links that the publication traverses in step \( i \) of delivery to \( S \).

Since publication routing is performed in a hop-by-hop fashion, the cost of routing a publication from \( P \) to \( S \) is the sum of the costs on each hop. Therefore:

\[
C_{P \to S} = C^1_{P \to S} + \ldots + C^n_{P \to S}
\]

where \( C^i_{P \to S} = M(B_i) + F(L_i) \) and \( n \) is the maximum distance between \( P \) and any \( S \in S \). In existing routing algorithms, in step \( i \) of routing a publication, broker \( B_i \) ’s decision to incur the cost \( C^i_{P \to S} \) solely depends on whether there exists at least one subscriber in \( S_{n-i} \), the subset of \( S \) reachable from broker \( B_i \) in the next \( n-i \) hops:

\[
C_{P \to S} = k_1C^1_{P \to S} + \ldots + k_nC^n_{P \to S}, \quad k_i = \begin{cases} 0, & \text{if } S_{n-i} = \emptyset. \\ 1, & \text{otherwise.} \end{cases}
\]

This cost model is oblivious to the cardinality of \( S \), which results in all publications being considered equal and incurring the same cost regardless of the number of subscriptions that they match. We call this cost model the binary cost model for publication routing.

The binary cost model is essential for guaranteeing correct publication delivery. However, routing publications based on this cost model can result in inefficient use of resources and consequently increasing delivery latency or reducing the overall publication delivery of a pub/sub system. Furthermore, under high load, publications that have many subscribers can experience the same high latency as publications with lower popularity. Therefore, it is beneficial to prioritize publications based on their impact on the overall publication delivery in order to improve resource utilization and to mitigate the impact of delayed or dropped publications when the system is under high load. In the next section, we propose a metric to estimate the impact that routing a publication can have on the total publication delivery.

4 SOLUTION COMPONENTS

We now present the various aspects which compose PopSub. First, we introduce a metric to measure publication popularity in a pub/sub system, along with a popularity-based cost model for publication routing. We then explain how PopSub evaluates publication popularity. Then, we discuss different alternatives to handle unpopular publications.

4.1 Popularity-Based Cost Model

In order to address the routing problem associated with the binary cost model, we need to estimate the resources required by a publication and its impact on the overall publication delivery. Using these two factors, we can extend the binary cost model and give priority to publications that result in the highest publication delivery with the lowest resource requirement.

The amount of resources required to route a publication from \( P \) to all matching subscribers \( S \) is proportional to the number of hops that it takes to reach all \( S \in S \). Consequently, longer paths result in more brokers and overlay links involved in publication routing. However, two publications traversing the same path and incurring the same routing cost can have a different impact on the total publication delivery. For example, in Figure 2, delivering the blue and gray publications from \( B_1 \) to \( B_3 \) result in the same routing cost. Nonetheless, in this case, routing the gray publication can result in four times more publication delivery.

In order to prioritize publications with the highest number of matching subscribers and lowest resources required for routing, we introduce a gain ratio that determines for each publication \( p \) the benefit of routing that publication one hop on the overall publication delivery of the system. We define the gain ratio of routing publication \( p \) as:

\[
G_p = \frac{|\text{sub}(S)|}{|T_{P \to S}|}
\]

where \(|\text{sub}(S)|\) is the number of matching subscriptions in \( S \) and \(|T_{P \to S}|\) is the average path length from publisher of \( p \) to \( S \). \( G_p \) estimates the maximum number of deliveries that can result from routing publication \( p \) one hop further. Since \( G_p \) depends on the number of matching subscriptions and average path length from the current broker, each routing broker on the path from publisher to subscribers has a different estimate for \( G_p \). Therefore, on broker \( B_1 \), \( G_p \) is the local estimate of \( B_1 \) for routing \( p \) one hop further.

For example, in Figure 2, \( B_1 \) publishes 3 different sets of publication, blue (\( A_3 \)), red (\( A_2 \)) and gray (\( A_1 \)). On \( B_4 \) there are two matching subscriptions, \( S_10 \) and \( S_{11} \), matching \( A_2 \). On \( B_5 \) there are three subscriptions, \( S_7 \) and \( S_8 \), matching \( A_2 \), and \( S_9 \) matching \( A_3 \). Therefore, on \( B_1 \):

\[
G_p(\text{red}) = \frac{4}{(2 + 2 + 3 + 3)/4} = \frac{4}{2.5}, \quad G_p(\text{blue}) = \frac{1}{3}
\]

This means on \( B_1 \) routing a red and blue publication one hop towards \( B_2 \) results in increasing the overall publication delivery by \( \frac{4}{2.5} \) and \( \frac{1}{3} \), respectively. Note that these numbers are just local estimates calculated by the broker and only provide a normalized metric to compare the gain of routing different publications, as fraction of a delivery does not exist and in this example routing both publications one hop would not result in any delivery. Furthermore, only distant subscriptions are taken into account for calculating publication gain. For example, while routing a red publication on \( B_4 \), \( G_p(\text{red}) = \frac{4}{7} \) rather than \( \frac{1}{3} \), as the local subscriptions can be delivered to, without any further forwarding.

The estimated \( G_p \) depends on the location of the broker routing \( p \) and popularity of \( p \), i.e., the number of matching subscriptions reachable from that broker. Therefore, a broker might have different popularity estimates for a publication across each outgoing overlay link since each link leads to a different subset of subscribers.
Furthermore, popularity of publications belonging to different advertisements are unrelated to each other and must be estimated separately. On each broker, PopSub estimates the publication gain ratio per advertisement per link. For example, in Figure 2, $B_1$ maintains a list of publication gain estimates for each overlay link going to $B_2$ and $B_4$. The list of publication gain estimates for each link includes one estimate per advertisement.

Each broker $B_i$ maintains a table of publication gain estimates, $G_{B_i}$, where $G_{B_i}[i, α]$ records the gain of routing a publication $p$ across link $ℓ$ for each $p$ matching advertisement $α$. Since advertisements are flooded in the overlay and all brokers know about all advertisements, the number of publication gain estimates a broker $B_i$ needs to maintain is $|L_{B_i}| \times |A|$ where $L_{B_i}$ is the set of overlay links of $B_i$ and $A$ is the set of all advertisements in the system.

PopSub estimates publication gains on each broker during subscription propagation. Initially, all entries of $G_{B_i}$ are set to $0$. A subscription $s$ traversing towards its matching advertisement $α$ is required to record the number of hops it has passed. While going from broker $B_i$ to broker $B_j$, $G_{B_i}[i,j,α]$ is updated by incrementing the number of matching subscriptions and updating the average path length using $s$.hopcount. Consequently, all brokers on the path between the publisher and the subscriber update their publication gain estimate. Figure 2 shows the updated publication gain tables of all brokers after propagation of all subscriptions. The missing tables all have three entries, each initialized to $\frac{1}{3}$, and are omitted to improve readability. Unsubscriptions are processed similarly and are also required to record the number of hops they traverse.

Performing the publication gain estimation during subscription propagation has the benefit of not requiring global knowledge about all subscribers and not hindering scalability of the routing algorithm since the only additional requirement is recording hop count of subscriptions.

The binary cost model can be extended to use the local estimation of $G_p$ to reflect the publication gain in the cost. Therefore, the cost of routing publication $p$ on broker $B_i$ to the next hop is $\frac{1}{G_p} \times (\phi + c_p)$. In other words, the cost of routing $p$ to the next hop is inversely proportional to the estimated gain of $p$.

PopSub is also compatible with subscription covering [5] with the only requirement that subscriptions should get fully propagated to update the publication gain tables. Although there is no communication saving in this case, the main benefit of subscription covering, namely reduction of routing table size, can be preserved. In the next section, we explain how PopSub routes publications based on the popularity-based cost model.

### 4.2 Publication Popularity Evaluation

On each broker $B_i$, publications are categorized into popular and unpopular publications. Similar to the publication gain table, this categorization is performed for each outgoing link of $B_i$. Therefore, for each entry $G_{B_i}[i, α]$ in the publication gain table, a flag pop is stored indicating whether publications matching advertisement $α$ are popular or not in the subset of the overlay reachable via link $ℓ$. Initially all entries have their pop flags set to true. Furthermore, each broker calculates and updates the message rate for each advertisement on each link. This message rate is also stored for each entry of the publication gain table in a $rate$ field. $G_{B_i}[i, α].rate$ records the average number of publications matching advertisement $α$ that has been forwarded via link $ℓ$ per second. Initially, all $rate$ fields are set to 0. Broker $B_i$ updates this message rate periodically based on the publications that it routes.

Periodically, broker $B_i$ evaluates the popularity of all table entries ($ℓ, α$). This is achieved by filling the broker capacity with the publications that have the highest gain ratio. We consider the broker’s capacity as its throughput in terms of number of publications it can forward per second ($τ_i$). Therefore, regardless of whether $τ_i$ is limited by broker’s CPU, memory or network resources, the aim is to achieve the highest number of deliveries using the available capacity. This problem is similar to the knapsack problem where the weight of each item is equal to its message rate and the value of each item is equal to its gain ratio. Table entries that are chosen to fill up the broker capacity are deemed popular, the rest of the entries are considered unpopular. Algorithm 1 is the algorithm performed by each broker $B_i$ periodically to re-evaluate publication popularity.

**Algorithm 1 Evaluating publication popularity on $B_i$**

1: function $EVALUATE$($G_{B_i}$) 2: sort $G_{B_i}$ by descending gain ratio 3: filled = 0 4: for $g \in G_{B_i}$ do 5: if filled < $τ_i \times φ$ 6: $g.pop = true$ 7: filled += $g.rate$ 8: else 9: $g.pop = false$

In Algorithm 1, the entries of the $G_{B_i}$ table are first sorted by decreasing order of their gain ratio. Next, starting from the entry with the highest $G_p$, entries are marked as popular until we find an entry $g$ such that the sum of the message rates so far plus $g$’s message rate exceeds the broker capacity to be filled. All entries with a $G_p$ equal to or lower than $g$’s publication gain is marked as unpopular. Since the average number of links per each broker is bounded and usually small, the complexity of Algorithm 1 on each broker is $O(n)$, where $n$ is the number of advertisements in the system.

Algorithm 1 solves a simplified version of the knapsack problem since we do not require to minimize the remaining capacity. Parameter $φ$ allows changing the threshold for popularity evaluation. Higher values of $φ$ means that more entries in $G_{B_i}$ are marked as popular since a higher share of the broker capacity is dedicated to popular publications. The value of $φ$ must be chosen in such a way that reflects the popularity distribution of the workload. For example, in a Zipfian workload, typically most subscribers are interested in only 20% of the publishers [25, 28]. Therefore, the value of $φ$ should correspond to 0.2 or less. $φ$ should be chosen using the average system utilization and the CDF of the workload distribution. For example, in a workload with a normal distribution and average system utilization of 50%, $φ = 0.5 \times 0.3 = 0.15$, as in a normal distribution more than 68% of values lie within the first standard deviation ($σ$) and almost all values lie within 3 $σ$. While workloads with uniform distribution are very rare [21, 25, 28], in these cases, $φ$ simply defines the percentage of table entries that are considered popular. Lastly, in order to avoid overloading the broker, it is preferred to always leave some headroom during capacity planning ($φ < 0.8$).
While in this work we assume that all publications are of equal priority, in some scenarios there can be single publications with few subscribers that have a high priority. To accommodate these cases, the capacity filing algorithm can be modified to first fill up the broker capacity with high priority messages, and after that, divide the remaining capacity among popular and unpopular content.

Any publication that is not planned to be routed through the tree is considered unpopular. In the next three sections, we discuss three alternative approaches to disseminate unpopular publications, namely, Direct Delivery, Batching and, Gossiping.

### 4.3 Direct Delivery

In this approach, inspired by Atmosphere [15] and OMF [18], a publisher delivers unpopular publications directly to the subscribers without forwarding the publication through the overlay. For example, in Figure 2, if blue publications are unpopular on the link between $B_1$ and $B_2$, $B_1$ directly delivers blue publications to $B_2$. This requires subscriptions to record the broker they originated from. Therefore, each subscriber records its own address in the subscription before propagating it in the overlay. Note that, since the identity of the actual client is not used and the broker ID is only used within the overlay brokers, the anonymity of the client is preserved. Direct delivery, while effective and simple, is susceptible to scalability issues. The reason is that, in the worst case, a publisher must forward a publication to all other brokers in the overlay. OMF and Atmosphere address this problem by limiting number of out-of-overlay forwards considering available resources on the publishing broker. In PopSub, however, direct delivery is performed only for unpopular publications. In a skewed workload, typical for real-world scenarios, unpopularity of a publication results in limited number of interested subscribers. This prevents Direct Delivery to encounter scalability issues. We evaluate scalability of Direct Delivery in section 5.

### 4.4 Batching

Batching messages to improve resource utilization is a common practice and supported in many existing message passing and pub/sub systems [14, 20]. Next, we explain a batching approach to handle unpopular publications in a content-based pub/sub system.

In this approach, upon receiving a publication $p$ to publish (Algorithm 2), the broker checks the popularity of $p$’s advertisement, $\alpha$, on each outgoing link $\ell$. If the entry $(\ell, \alpha)$ is marked unpopular, the publication gets buffered rather than forwarded. The set of buffered publications $B$, are batched in one publication and forwarded via the tree whenever one of the following conditions is satisfied:

- $|B| \geq |\mathcal{G}_{B_1, \min} / \overline{G}|$, where $\mathcal{G} = \{\text{gain}(p) | p \in B\}$ and $\overline{G}$ is the average gain of buffered publications and $\mathcal{G}_{B_1, \min}$ is the minimum gain among popular entries.
- $T$ seconds has passed since the first publication was buffered.

The first condition ensures that enough publications are batched to maintain the minimum publication gain on the broker and the timeout condition provides an option to prevent starvation of subscribers to unpopular content. Note that, publications are only buffered at the publisher and once published as a batch publication, they are not buffered again by any forwarding broker downstream.

### Algorithm 2 Publishing publications on $B_1$ (Batching)

```
1: function Publish(p)
2: \( \alpha = \text{matchAdv}(p) \) \hspace{1em} \text{\# find matching advertisement}
3: \( \mathcal{L}^p_{B_1} = \text{matchSubs}(p) \) \hspace{1em} \text{\# find links to forward } p
4: \text{for all } \ell \in \mathcal{L}^p_{B_1} \text{ do}
5: \text{if } \mathcal{G}_{\ell, \alpha}.\text{pop is true}
6: \text{send}(p, \ell)
7: \text{else } \text{send}(p, \ell) \quad \text{\# } p \text{ is unpopular}
8: \mathcal{B}(\ell, \alpha).\text{add}(p) \quad \text{\# add to buffer for } (\ell, \alpha)
9: \text{if } |\mathcal{B}(\ell, \alpha)| \geq 1
10: \text{timer.run}(T, \text{sendBatch}(B, \ell))
11: \text{else if } |\mathcal{B}| \geq |\mathcal{G}_{B_1, \min} / \overline{G}|
12: \text{timer.cancel}
13: \text{sendBatch}(B, \ell)
```

### Algorithm 3 Receiving publications on $B_1$ (Batching)

```
1: function Receive(bp)
2: \( \alpha = \text{matchAdv}(bp) \) \hspace{1em} \text{\# find advertisement (same for all)}
3: \( \mathcal{L}^p_{B_1} = \text{matchSubs}(p) \)
4: \text{for all } \ell \in \mathcal{L}^p_{B_1} \text{ do}
5: \text{matchSubs}(\ell, bp)
6: \text{for all } \ell \in \mathcal{L}^p_{B_1} \text{ do}
7: \text{add } p \text{ to the batch publication created for } \ell
8: \text{send created batch publications across each } \ell
```

### Gossiping

Gossiping (a.k.a. epidemic protocols) provides a scalable dissemination mechanism without requiring building and maintaining an overlay. PopSub utilizes an existing gossiping protocol, namely Lightweight Probabilistic Broadcast (lpbcast) [11] to disseminate unpopular publications via gossiping.

Next, we provide a short overview of lpbcast, a gossip-based broadcast protocol which provides a scalable approach for large-scale data dissemination. lpbcast uses partial views to perform message routing and membership management in a decentralized
and scalable fashion. Therefore, processes can avoid global knowledge and rely only on a partial view of the system. This partial view consists of a subset of the processes in the system with a fixed maximum size. On each process b, besides the partial view \( \mathcal{V}_b \), each process maintains the following four sets:

- \( \mathcal{M}_b \): set of messages to be gossiped by b
- \( \mathcal{D}_b \): a digest of messages gossiped and seen by b
- \( \mathcal{S}_b \): set of subscribers (process IDs) that b is aware of
- \( \mathcal{U}_b \): set of unsubscribers that b is aware of

Periodically, b creates a gossip message \( g = (\mathcal{M}_b, \mathcal{D}_b, \mathcal{S}_b, \mathcal{U}_b) \) and sends it to a random subset of processes in \( \mathcal{V}_b \). A receiving process, \( b' \), uses \( \mathcal{S}_b \) and \( \mathcal{U}_b \) to update its partial view. This allows gradual removal of processes not interested in receiving messages anymore and gradual integration of new processes. Any message in \( \mathcal{M}_b \) not previously seen by \( b' \) is eligible for delivery and gossipping in the next round. The digest received from \( b \) is used to update the knowledge of \( b' \) of published messages and retrieval of missing messages. \( \text{lpbcast} \) also supports periodical retrieval of missing messages.

A straightforward approach to use \( \text{lpbcast} \) for dissemination of unpopular publications is to run one \( \text{lpbcast} \) instance on each broker. Each broker \( B_i \) maintains a partial view of all brokers in the system and participates in gossiping publications of all types. Upon routing a publication, popular publications are routed through the tree based on the matching results and unpopular publications are added to \( \mathcal{M}_b \) to be gossiped along with other pending unpopular publication in the next gossip round. Utilizing \( \text{lpbcast} \) in this way can result in unnecessary gossip overhead on brokers. Since \( \text{lpbcast} \) is a broadcast protocol, all brokers receive all unpopular publications, even if a broker is not a publisher, subscriber or forwarding broker (located on the path between a publisher and subscriber).

In order to reduce the gossip overhead, \( \text{PopSub} \) creates broadcast gossip groups for each advertisement on demand. Furthermore, each group includes as members only brokers that send or receive publications belonging to that advertisement. Therefore, \( \text{Gossiping} \) provides an alternative which combines \( \text{Direct Delivery} \) and \( \text{Batching} \). Similar to \( \text{Direct Delivery} \), unpopular publications are not processed by uninterested middle brokers. Furthermore, batching publications in one gossip and periodically disseminating them allows brokers to benefit from batch processing and forwarding. Furthermore, any received gossip only needs to be matched against local interests of the broker and unlike batching, \( \text{Gossiping} \) does not require the forwarding broker to match the publications with all subscriptions known to the broker. Lastly, \( \text{Gossiping} \) provides a tunable dissemination mechanism. For example, by limiting history of events to gossip, it prevents overloading and drops messages if necessary. \( \text{Gossiping} \) provides a probabilistic delivery guarantee and increased delivery latency of unpopular publications. These are tradeoffs to achieve higher scalability and performance.

Next, we explain our proposed on demand gossip group creation, and, routing and propagation of unpopular publications.

Upon receiving publication \( p \), publisher \( B_i \) uses Algorithm 4 to route \( p \). \( B_i \) first finds the matching advertisement \( \alpha \) and the set of links \( \ell \) leading to the next hop. Next, if \( p \) is popular on \( \ell \) (line 5), \( B_i \) forwards \( p \) to \( \ell \) (line 8). If not, \( p \) needs to be routed via gossip.

In order to inform subscribers that this publication is unpopular, \( B_i \) sets a switchToGossip flag in \( p \) to allow subscribers to prepare for receiving the rest of the unpopular publications belonging to the same advertisement via gossip. Therefore, if such a flagged publication has not been sent on \( \ell \) for an unpopular advertisement \( \alpha \) (line 9), \( B_i \) sets this flag (line 10-11) and starts a new gossip group for \( \alpha \) if necessary (lines 13). Next, the broker includes itself as a partial view of the gossip group in \( p \) (line 14) and forwards \( p \) on \( \ell \). This ensures that the receiving subscriber knows of at least one member of the gossip group for \( \alpha \) in case \( \text{group}_\alpha \) was just created and is empty. This is crucial to make sure that no gossip group experiences partitioning, since, in the worst case, all members at least know the publisher. Note that \( \text{group}_\alpha \) can already exist if for example, \( \alpha \) was already unpopular on another link \( \ell' \) in \( L_p \). In that case instead of the publisher, its partial view is included.

An unpopular advertisement can become popular due to the periodical re-evaluation of publication popularity (Algorithm 1). \( B_i \) controls this by checking for publications that are popular but \( B_i \) has already sent a flagged publication indicating their unpopularity. In this case a flag indicating switching delivery to the tree is set in \( p \) before sending it on \( \ell \).

After sending the first flagged publications, future unpopular publications belonging to advertisement \( \alpha \) on link \( \ell \) are simply added to the set of messages of the \( \text{lpbcast} \) process to be gossiped on the next round (line 17). Using the gossip broadcast, unpopular publications are batched together and sent as one message representing the new unpopular publications since the last gossip round. All subscribers with a subscription matching \( \alpha \) receive all unpopular publications without any filtering since the group uses a broadcast protocol.

Subscribers process publications based on Algorithm 5. If \( p \) includes a flag to switch to gossip dissemination and a gossip group for \( \alpha \) does not exist, \( B_i \) starts an \( \text{lpbcast} \) process and joins the broadcast group via the partial view included in \( p \). Processing a publication with a flag to switch back to the tree is simply gossipping an unsubscribed and stopping the \( \text{lpbcast} \) instance on \( B_i \). A subscriber receiving a publication \( p \) via gossip has to match \( p \) against its local subscriptions before delivery.

Any broker \( B_i \) receiving publication \( p \) for routing, which is not a publisher or subscriber of \( p \), simply forwards \( p \) on all \( \ell \in L_p \).

---

**Algorithm 4: Publishing Publications on \( B_i \) (Gossiping)**

```
1. function PUBLISH(p, groupp, flagp)
2.  α = matchAdv(p)  // find matching advertisement
3.  ℓ = matchSubs(p)  // find links to forward p
4.  for all \( \ell \in L_p \) do
5.      if \( G_{\ell}(\alpha) \) pop is true
6.          if flag\{\ell, α\} is true
7.              p.switchToTree = true
8.              send(p, ℓ)
9.      else if flag\{\ell, α\} is false
10.         \( p\).switchToGossip = true
11.        if groupp ∪ \{\ell\} \in groupp
12.            groupp = new \text{lpbcast}(α)
13.        p.view = \{\ell\} ∪ groupp \cup \{\ell\}
14.        send(p, ℓ)
15.  else \( p \) is unpopular and flag\{\ell, α\} is set
16.      groupp, M = \{\ell\} ∪ groupp ∪ M
```

The above algorithm allows for a dynamic and scalable dissemination of unpopular publications, while avoiding overloading and message drops. It enables the system to efficiently manage and distribute content based on user preferences and popularity, ensuring a better user experience and improved resource utilization.
without considering the popularity of \( p \). This means that gossip groups consist only of publisher and subscriber brokers.

**Algorithm 5 Receiving publications on \( B_i \) (Gossiping)**

1. function Receive\( (p, group_p) \)
2. \( a = \text{matchAdvs}(p) \) \( \rightarrow \) find matching advertisement
3. if \( p.\text{switchToGossip} \) is true
4. if \( group_p \notin \text{group}_p \)
5. \( \text{group}_p = \text{new lpbcast}(a) \)
6. \( \text{group}_p, \forall = p.\text{view} \)
7. else if \( p.\text{switchToTree} \) is true
8. \( U = \{ B_i \} \cup U \) \( \rightarrow \) Unsubscribe from gossip group
9. send gossip \( \rightarrow \) Required to propagate unsubscribe
10. remove group\( _a \)
11. find matching local subscriptions and deliver

The frequency of switching a stream of publications between gossiping and the tree is a function of the frequency of running Algorithm 1. For publications which are the last few popular ones, it is possible to switch between gossip and overlay upon every re-evaluation, for example due to changes in message rate of a more popular publication. We avoid this thrashing effect by allowing a 10\% threshold while planning broker capacity. When there is no change in the gain ratio table, PopSub maintains the same capacity as the last round, as long as the current filled capacity is within 10\% of the capacity to fill. Changes in subscriptions and gain ratios are affected in the next re-evaluation.

While switching the dissemination mechanism, it is possible for a subscriber to miss some publications or receive duplicates. In order to guarantee complete delivery, each \( B_i \) needs to maintain a cache of the publications it publishes and ID of publications it has seen. These two sets are used for retransmitting missing publications and detecting duplicates. The size of these caches must be large enough to cover the period of time it takes for a flagged publication to reach its subscribers. For example, in Figure 2, after \( B_i \) sends a flagged publication to \( B_2 \), it does not forward any more publication over link \( f_{12} \) but rather keeps them for gossiping. Therefore, a broker must cache publications published from the time the flagged publication is sent until the subscriber receives the publication and subsequently creates and joins the gossip group. This time period can be assumed in the worst case as long as the longest path from a publisher to a subscriber (i.e., the diameter of the overlay). Furthermore, publication sets that have a subset of them delivered via gossip might not maintain the publisher order. This is also the case for Direct Delivery. Therefore, PopSub provides per-publisher ordering if a publication set is all delivered via the main dissemination tree (i.e., is popular or uses Batching). Per-publisher unique IDs for publications and reordering buffers can be used to provide per-publisher ordering guarantees for unpopular publications as well.

## 5 EXPERIMENTAL EVALUATION

In this section, we evaluate our approach via simulation using real-world workloads, latency values, and traces. We implemented our approach and a baseline, as part of a content-based pub/sub system simulator written in Java. We evaluate PopSub using the three proposed dissemination mechanisms for unpopular publications, namely Direct Delivery (based on Atmosphere and OMF), Batching, and Gossiping. Furthermore, our baseline is a popularity-agnostic pub/sub system which handles all publications using the tree.

### 5.1 Workload

We evaluate PopSub using hierarchical topologies which are used for high-throughput pub/sub systems [5, 10]. We use real-world traces which provide latency values across 1715 machines, connected via the Internet and estimated using the King method [12, 26]. We generate overlay topologies of different sizes from this graph as follows. We select one random node in the graph and perform a BFS traversal on the graph, with the selected node as root, until we have collected a subgraph with the same size as the desired topology size. Next, we calculate the shortest-path tree from the selected root to all other nodes in the subgraph and use this tree as the overlay topology. Furthermore, any out-of-overlay connection uses the latencies provided in the subgraph. Using this process, we create five different sets of topologies using different seeds.

Since the popularity of publishers is an important factor that PopSub can take advantage of, we use a workload based on Twitter traces [22]. These traces provide the follower relationship among 81306 users where each user has at least one follower (subscriber). We generate our workload based on the popularity distribution extracted from this trace. We sort users based on the number of followers they have and calculate each user’s normalized popularity as the percentage of followers from the total user count. In each experiment there are 200 publisher clients and \( N \) subscriptions. The number of subscriptions per publisher client is selected based on the Twitter follower distribution.

Publisher clients are connected to the top of the tree (root and its children) and each publish for one minute at a rate of 10 pub/sec. Subscriber clients each have one subscription and are distributed among all brokers of the tree topology based on a random uniform distribution. The number of advertisements is 200 (one per publisher client) and in total there are 20 different classes of advertisements. The number of advertisements per class is based on a random uniform distribution. Each message has two attributes and the range of each attribute is 0 to 1000. Publication and subscription attribute values are chosen based on a random uniform distribution over the content space. Each experiment is run with 5 different topologies and 5 different workloads (25 runs per results) and the values are averaged. In order to study the impact of publication popularity on our metrics, we also use a synthesized workload where the popularity of the advertisement classes among the subscribers follows a Zipf distribution with different Zipf exponent values.

Another important factor for our evaluation is broker throughput and performance gains achieved by batch processing and forwarding publications (batch gain). Rather than using random values, we use an existing content-based pub/sub system to run benchmarks, collect throughput values and measure batch gains. We use two brokers on a Dell PowerEdge R430 server. Each broker runs in a VM with 4 CPU cores and 10GB RAM. The publishing broker publishes 3000 publications. On each run the publisher batches publications in different sizes from 1 to 256. Each benchmark is repeated 5 times with different seeds, and the results are averaged. The publication...
throughput per second for each publisher is calculated, and throughput values of different batch sizes is normalized with respect to batch size 1. The collected set of throughput measurements for batch size 1, is used as the throughput value of brokers in the simulation. In the simulations, each time a broker forwards a batch publication, \( bp \), of size \( n \), the number of throughput units required to forward \( bp \) is looked up in the batch factor table.

Each PopSub broker re-evaluates its publication popularity metrics every 2 seconds based on the message rates collected in the last interval. \( ipbcast \) is configured with a view size of 10 and a fanout of 2. This means in each gossip round, each gossip is sent to 2 brokers randomly selected from the 10 members of the partial view. The gossip interval is every 10 seconds. Furthermore, \( Batching \) timeout is set to 3 seconds.

5.2 Metrics

**Publication Gain ratio** is the average gain ratio of publications that are routed through the tree. This measures the resource efficiency of the pub/sub system. A higher gain ratio means a higher number of successful deliveries to resource utilization. Each broker that evaluates the publication popularity metrics, records the gain ratio of publications that the broker decides to route via the overlay. The values collected by all brokers is averaged. While for baseline, \( Batching \), and \( Direct \) Delivery all publications are routed via the overlay, for \( Gossiping \), this metric only represents popular publications.

**Pure forwards** is the total number of publications forwarded by the overlay brokers where the forwarding broker has no local subscriber matching the publication. Reducing number of pure forwards improves the routing efficiency.

**Unpopular deliveries** is the total number of publications delivered via \( Direct \) Delivery, \( Batching \), or \( Gossiping \). This metric is always zero in the baseline. **Publication delivery latency** (PDL) is the time that it takes to deliver a publication to a subscriber. The \( 99^{th} \) \%ile of delivery latency of all successfully delivered publications is defined as the PDL. We report PDL values for popular and unpopular publications separately. **Popular PDL** represents latency of publications delivered via the tree and **unpopular PDL** represents latency of deliveries via \( Direct \) Delivery, \( Batching \), or \( Gossiping \).

**Gossip false positive ratio** represents the percentage of publications that were received via gossip but did not match any local subscription and therefore did not result in a publication delivery. Furthermore, **Match per gossip** is the average number of publications delivered per each received gossip. These two metrics show the effectiveness of gossiping.

In order to study the scalability of \( Direct \) Delivery, we count number of forwards per publication on each publisher, each time an unpopular publication is published. This represents the number of direct publication forwards required to deliver unpopular publications. The presented value is the \( 99^{th} \) \%ile of all the collected values. **Number of gossip groups** shows the number of publishers that have unpopular publications in all or a subset of the overlay.

Lastly, **average queue size** is the average length of queue of publications buffered due to overloaded brokers. The size of this queue directly influences the publication delivery latency. The measurement is performed as follows: each time there is a publication arriving at a broker to be forwarded, and due to overload, the publication has to be queued, the length of the queue is recorded. All collected queue lengths are averaged to represent the average queue size of an approach.

5.3 Experiments

**Impact of \( \phi \).** In this experiment, we study the impact of \( \phi \) on our metrics. \( \phi \) determines the percentage of broker capacity allocated to handling publications via the tree. Increasing \( \phi \) results in more publications being considered popular. The overlay consists of 200 brokers and the number of subscriptions (clients) is 22,000. Figure 3a shows that for our Twitter-based workload, determining popularity threshold by filling 5-10\% of publisher broker capacity, results in 29-62\% higher gain ratio compared to baseline. This means that for \( Batching \) and \( Gossiping \), each publication forward in the overlay, results in up to 4.3 deliveries on average. This value is 2.65 for the baseline. **Direct Delivery** results in gain ratios similar to the baseline because all unpopular deliveries have a gain ratio of 1, i.e. one forward results in one delivery, and this brings down the total average. While the total average gain ratio of the three approaches is different, the average gain ratio of popular publications is similar in all three approaches and is the same as \( Gossiping \). By increasing the threshold for popularity evaluation, all approaches result in a similar gain ratio since all publications are considered popular. This is also evident from Figure 3b, where increasing \( \phi \) decreases number of publications delivered via \( Direct \) Delivery, \( Batching \), or \( Gossiping \). While **Direct Delivery** shows a gain ratio similar to the baseline, all three approaches benefit from the popularity evaluation algorithm as shown in Figure 3c. Here, identifying unpopular publications and delivering them via one of the three proposed approaches, can reduce number of pure forwards by up to 59\% compared to the baseline. As mentioned previously, \( \phi \) can be used to tune the popularity evaluation algorithm to match the popularity of the workload. Since the Twitter-based workload has a Zipfian distribution, smaller values of \( \phi \) result in better performance. The local gain ratio estimation and relaxed capacity filling threshold (to prevent thrashing) results in different number of unpopular deliveries and consequently different pure forward counts for the three approaches. This experiment shows that choosing \( \phi \) according to the popularity of the workload can result in up to 62\% improvement in gain ratio and up to 59\% lower number of pure forwards.

**Impact of overlay size.** In this experiment, we study the impact of the overlay size on our metrics while the number of subscriptions is fixed at 22,000 and \( \phi = 0.05 \). Figure 3d shows that increasing number of brokers while the number of clients is the same, reduces the gain ratio. This is inevitable since larger overlays result in longer paths. As number of subscriptions is the same, the gain ratio decreases. Consequently, increasing number of brokers with the same number of clients, decreases resource utilization of the system. The increasing number of pure forwards (Figure 3e) in all approaches also confirms longer paths between publisher and subscribers with no interested forwarding brokers.

The three proposed approaches to handle unpopular publications, provide different tradeoffs. While **Direct Delivery** does not improve the average gain ratio, it reduces number of pure forwards (Figure 3e) and provides an unpopular PDL similar to that of the popular publications (The popular PDL of all approaches is around 135ms). On the other hand, **Batching** and **Gossiping** improve the average gain ratio and reduce pure forwards at the cost of higher
unpopular PDL. Figure 3f shows that, compared to Direct Delivery, unpopular deliveries via Batching and Gossiping result in PDL of up to 10 times and 200 times slower, respectively. The large difference however only applies to a small portion of deliveries since compared to the 99th percentile, the 90th percentile of PDL is much smaller. Figure 3g shows that 90th percentile of unpopular PDL for Batching and Gossiping is up to 2 times and 70 times higher, respectively. Gossiping trades off the increased latency for higher scalability in comparison to Batching. This tradeoff is studied in the last experiment with an overloaded pub/sub system. Note that, unpopular PDL of Direct Delivery and Gossiping does not increase despite larger overlay sizes and longer paths, since these approaches do not use the tree.

This experiment shows that PopSub provide better performance in systems which are not underutilized. Furthermore, we showed the tradeoffs provided by the three proposed approaches.

Impact of number of subscriptions. In this experiment, we study the impact of number of subscriptions on our metrics. We use an overlay of size 200, $\varphi = 0.05$ and change number of subscriptions from 20,000 to 25,000. Figure 3h shows that increasing subscriptions increases gain ratio in all approaches. The reason is that higher number of subscriptions means a higher number of matching subscriptions on each forwarding broker on average. Since the overlay size and hence the average path length is fixed, the gain ratio increases.

Similar to previous experiments, Figure 3h shows that Batching and Gossiping can improve average gain ratio by up to 62% compared to the baseline. Note that Gossiping can slightly outperform Batching, because batched publications still have to go through the tree and potentially pass through uninterested brokers. This reduces the average gain ratio compared to Gossiping which gossips unpopular publications only among brokers with a matching local subscription. The lower number of pure forwards for Gossiping compared to Batching in Figure 3i, confirms this. Compared to the baseline, Gossiping, Batching and Direct Delivery reduce pure forwards by up to 56%, 37%, and 42%, respectively.

As shown in Figure 3j, increasing number of subscriptions does not impact unpopular PDL. The reason is that since the overlay size and hence number of subscribing brokers is the same, only number of subscriptions per broker changes. Consequently, number of unpopular deliveries increases. However, as the system is not overloaded, all approaches are able to scale to larger number of subscriptions. The impact of increasing subscriptions on an overloaded system is studied in the last experiment.

As discussed previously, Direct Delivery is scalable in skewed workloads since it only forwards unpopular publications. Figure 3k shows the 99th percentile of number of publication forwards required to deliver unpopular publications. Direct Delivery requires up to 29 to 36 publication forwards to deliver unpopular publications since a publisher must send the publication directly to subscribers with one or more matching subscriptions. Since this number is much higher for popular publications, using Direct Delivery for popular publications is not scalable. Furthermore, increasing number of subscriptions results in a sub-linear increase in number of direct forwards per publisher. The reason is that increasing number of
subscriptions increases the number of matching subscribers for all publishers. Consequently, unpopular publications also have a higher number of subscribers and the publisher needs to forward each unpopular publication to a larger number of subscribers. In contrast, number of groups in Gossiping is not affected by number of subscribers since the number of publishers with unpopular publications is related to the workload and client distribution. Furthermore, unlike Direct Delivery, a publisher publishes unpopular publications by only gossiping the publication to two members of the gossip groups, regardless of its number of subscribers.

This experiment shows that while PopSub can improve the gain ratio of the pub/sub system, the three proposed approaches greatly reduce number of pure forwards in the overlay. Furthermore, we showed that Direct Delivery is scalable for scenarios where the workload is skewed and subscribers do not follow publishers based on a uniform distribution.

**Impact of workload skewness.** In this experiment, we study the impact of workload skewness on our metrics using a synthetic workload that follows a Zipf distribution. We change workload skewness by increasing the Zipf exponent ($s$) which results in more subscriptions to popular advertisements and reduction of the number of subscriptions to unpopular advertisements. While $s = 1$ results in a skewed workload, were almost 80% of subscriptions are interested in a small set of popular advertisements, an exponent value of 0.05 results in a uniform-like distribution of subscriptions among advertisements. The overlay size and number of subscriptions are fixed at 200 and 22,000, respectively and $\varphi = 0.05$.

Figure 31 shows the impact of workload skewness on each approach’s gain ratio. All approaches result in a higher gain ratio for workloads where distribution of subscriptions matching publishers is more skewed (increasing $s$). The reason is that in such non-uniform workloads, there actually are popular and unpopular publications and PopSub is able to take advantage of this to improve gain ratio. This is also evident from Figure 3m where the higher popularity of some publications in more skewed workloads, results in reduced pure forwards for all approaches.

Two important observations in Figure 31 are the following. First, Batching can improve gain ratio by up to 40% regardless of the workload skewness. The reason is that while Direct Delivery and Gossiping move unpopular publications out of the tree to improve gain ratio of the tree, Batching always uses the tree. Therefore, by combining several unpopular publications, Batching can always improve the gain ratio. This is the inherent performance gain of batching. However, prioritizing based on the introduced metric allows identifying slightly more popular publications and limiting batching latency to a predictable subset of the publications which are slightly less popular. Therefore, the more skewed the workload, the less number of clients are affected by this batching latency.

Secondly, Gossiping improves gain ratio in skewed workloads ($s > 0.5$). The reason is that in such workloads, the popularity evaluation algorithm can identify popular publications and by keeping unpopular publications out of the tree, improve the gain ratio. In uniform workloads, the popularity evaluation algorithm merely chooses a subset of the publishers which may only slightly be more popular, since there is no real popular publisher in a uniform workload. While Direct Delivery shows gain ratios similar to baseline due to direct deliveries with a gain ratio of 1, the gain ratio of Direct Delivery for only popular publications is similar to Gossiping.

Figure 3n shows that besides a gain ratio improvement in skewed workloads, Gossiping performance also improves. Gossip false positives decrease by up to 19% and publication match per gossip improves by up to 18%. The reason is that in a skewed workload, number of subscription to unpopular publishers decreases as majority of subscribers are interested in popular publishers. Therefore, in a smaller gossip group the likelihood of receiving publications that do not match the local interest of the broker decreases. This is also evident from the increasing publication match per each received gossip. Figure 3o shows that due to the decreased number of subscriptions to unpopular advertisements, the number of gossip groups slightly decreases in skewed workloads. Furthermore, the decrease in the number of publication forwards required for direct delivery shows that in skewed workloads Direct Delivery is scalable and does not overwhelm publishers.

This experiment shows that PopSub improves gain ratio and reduces pure forwards regardless of the distribution of the subscriptions among publishers. While Direct Delivery should be used only in skewed workloads to avoid scalability issues, Batching and Gossiping do not have this limitation. Furthermore, Batching can improve gain ratio even in uniform workloads.

**PopSub in an overloaded system.** In this experiment, we study the performance of PopSub in an overloaded pub/sub system. Here, in an overlay of 300 brokers, 200 publishers publish at a rate of 15 pub/sec and we increase number of subscriptions from 30,000 to 50,000. Figure 4a shows that similar to previous experiments, PopSub can improve gain ratio and increasing number of subscriptions increases the gain ratio. Figure 4b shows the average queue size of the overlay brokers. As evident from the baseline, the pub/sub system is overloaded which results in high number of messages queued to be processed. However, Batching, Direct Delivery and Gossiping reduce the average queue size by up to 33%, 60% and 98%, respectively. Direct Delivery results in smaller queue sizes because unpopular publications do not go through the overlay. In contrast, Batching routes all publications through the overlay and therefore results in a higher load on forwarding brokers. In comparison to Direct Delivery which puts higher load on already overloaded publishing brokers, Gossiping avoids this by distributing the dissemination of unpopular publications evenly among all interested subscribers, an inherent property of gossip protocols.

Figures 4c and 4d show the 99th %ile of PDL for popular and unpopular publications. Direct Delivery and Gossiping reduce popular PDL by up to 40% and 57%, respectively. Similar to Figure 4b, Gossiping outperforms Direct Delivery by avoiding overloading publishing brokers. Note that, while due to the overloaded brokers, in all approaches, popular and unpopular publications are affected, using Gossiping, popular publications have 17% to 33% lower PDLs compared to unpopular publications. This is due to the fact that by involving only subscribers to unpopular advertisements and avoiding overwhelming publishers, Gossiping can reduce the impact of unpopular publications on the deliveries made through the overlay. Lastly, The higher PDL of Batching is due to routing all publications through the overlay, which in comparison to the other two approaches results in a higher load on the brokers. This experiment shows that prioritizing publications based on their gain ratio can
reduce load on overlay brokers. Furthermore, handling unpopular publications via Direct Delivery and Gossiping improves publication delivery latency.

Based on our experiments, Direct Delivery is suitable for systems which are not over-utilized and cannot tolerate PDLs higher than the overlay average. Although Batching and Gossiping can always improve scalability and resource utilization, Batching is more suitable for scenarios where the subscription distribution among publishers is not skewed and Gossiping can be used in scenarios where the system may be overloaded.

6 CONCLUSIONS

In this paper, we presented PopSub, our approach to increase resource utilization of a pub/sub system by prioritizing publications based on their popularity and handling less popular publications with approaches that require fewer resources. The three proposed approaches for handling unpopular publications provide different tradeoffs and are suitable for different scenarios. Furthermore, by identifying unpopular publications and using the proposed alternative dissemination approaches only for such publications, PopSub is able to maintain the same publication delivery latency for a majority of publications while increasing resource utilization of the system. The result of our evaluations, using real-world workloads and traces, confirms that PopSub is able to improve resource efficiency of the system by up to 62%, reduce unnecessary publication forwards by up to 59%, and reduce popular publication delivery latencies by up to 57% in an overloaded pub/sub system. Additionally, these improvements can be achieved with workloads following any distribution of subscriptions on the advertisements.

7 ACKNOWLEDGMENTS

This research was supported by the Alexander von Humboldt Foundation.

REFERENCES